Universidad Mayor de San Andrés

Facultad de Ingenieria

Carrera de Ingeniería Electrónica

Proyecto de Grado

ESTUDIO DE UNA RED (SDN) CON EL PROTOCOLO OPENFLOW Y EL CONTROLADOR RYU

Autor: Elizabeth Vino Duran

Asesor: Ing. Jose Campero

LA PAZ - BOLIVIA

2019

UNIVERSIDAD MAYOR DE SAN ANDRÉS
FACULTAD DE INGENIERIA

LA FACULTAD DE INGENIERIA DE LA UNIVERSIDAD MAYOR DE SAN ANDRÉS AUTORIZA EL USO DE LA INFORMACIÓN CONTENIDA EN ESTE DOCUMENTO SI LOS PROPÓSITOS SON ESTRICTAMENTE ACADÉMICOS.

LICENCIA DE USO

El usuario está autorizado a:

a) Visualizar el documento mediante el uso de un ordenador o dispositivo móvil.
b) Copiar, almacenar o imprimir si ha de ser de uso exclusivamente personal y privado.
c) Copiar textualmente parte(s) de su contenido mencionando la fuente y/o haciendo la cita o referencia correspondiente en apego a las normas de redacción e investigación.

El usuario no puede publicar, distribuir o realizar emisión o exhibición alguna de este material, sin la autorización correspondiente.

Resumen

Existe un gran crecimiento de las redes de información en la forma que se transportan los paquetes, como su adaptación a nuevas tecnologías. Mientras las nuevas redes crecen, las tradicionales no soportan nuevas aplicaciones, por lo que usuarios/empresas requieren servicios de red actualizados. Se debe otorgar soluciones eficaces a las empresas, afectando lo menos posible su estructura a pesar de los costos elevados que supone la implementación de nuevos equipos.

La virtualización de redes es una de las posibles respuestas a las necesidades mencionadas. Este procedimiento se encuentra en un momento de expansión y se enlaza con SDN (Software Defined Network) que plantea una nueva forma de manejar las redes al centralizar el software que maneja los flujos de paquetes.

El proyecto propone investigar el manejo de redes de SDN, exponiendo el flujo de paquetes, analizando su arquitectura desde el protocolo OpenFlow y mediante el controlador de código abierto RYU. Se considerará la topología del Instituto de Electrónica Aplicada para ejemplificar la red, mediante simulaciones en Mininet.

Agradecimientos

Quisiera expresar mi mas profunda gratitud a mis padres Feby Duran Mamani y Nemesio Vino Vargas por su continuo apoyo, aliento, cariño y sacrificio,

A mis hermanas Jhannet y Esther, mi hermano John Ever por siempre apoyarme.

A Dios por sostenerme y protegerme por todo mi camino, dandome fuerzas y superar obstaculos y dificultades a lo largo de mi vida.

Mi especial agradecimiento la carrera de Ingenieria Electronica, a sus docentes y adminitrativos por darme un segundo hogar en estos años de aprendizaje.

Al Ingeniero Jose Campero por la paciencia y tomarse el tiempo en este trabajo. Ha sido un privilegio porder contar con su guia y ayuda.

Y un particular a mis amigos por su compañía y soporte.

Dedicado a mis padres por haberme forjado como la persona que soy y me motivaron constantemente para alcanzar mis anhelos.

Contenido

Lista de Figuras

Lista de Tablas

Lista de Comandos

1. Introducción

Debido a el gran crecimiento de la Internet que se encuentra en desarrollo tanto en tráfico como complejidad. Los dispositivos manejan información a través de paquetes en una red (routers, switches, firewalls, etc.) y encaminan desde el dispositivo de origen hacia el dispositivo de destino. A medida que va creciendo la complejidad de los dispositivos para los usuarios también se incrementa la necesidad de interconectarse.

Se vio que las redes de datos son troncales al momento de manejar grandes cantidades de paquetes generados por dispositivos, existe la necesidad de optimizar las redes de información, debido a que las redes tradicionales necesitan más elementos para soportar los servicios requeridos.

El incremento de uso de la información nos enfrenta a nuevos posibles de errores en el flujo de paquetes, la arquitectura de SDN ha ganado mucho terreno en el ámbito de las redes de información dando soluciones a los problemas que tienen las redes tradicionales.

Para comprender la necesidad de SDN y OpenFlow es necesario explicar la tecnología relacionada con las redes.

Networking en tecnología

A finales de los 50 la compañía BELL realizó implementación del primer módem que permitía transmitir datos binarios sobre una línea telefónica simple. A raiz de esto se publicó la teoría sobre la utilización de la conmutación de paquetes para transferir datos.

A principios de los 60 se inició las investigaciones por parte de ARPA agencia del ministerio estadounidense de defensa, con el objetivo de conectar todos los equipos de las instalaciones del gobierno de Estados Unidos, llamada ARPANET dicha agencia durante los 70 se logró conectar 23 computadoras y el primer correo. Generando organizaciones como InterNetworking Working Group organización encargada de administrar la internet, y acoplando más naciones al uso de esta red.

Debido al gran interes del uso de la transmisión de datos, se puso en definición el protocolo TCP/IP y la palabra Internet, dando el fin a ARPANET.

hacia los años 90 la Internet dejo de ser una red exclusivamente académica para abrirse a la poblacion, motivada por diversos factores:

- Se anuncia públicamente la *World Wide Web*
- Aparece el navegador web *NCSA Mosaic* Primer buscador de la historia.
- Se conectan 10 millones de computadoras.

A principios de los 2000, las redes seguió expandiéndose debido al avance en su fiabilidad, velocidad, rendimiento y como consecuencia el nacimiento de tecnologías como ADSL que proporcionaban acceso a Internet,Dicho crecimiento facilitó el uso de diferentes servicios: correo electrónico, servicios de teleconferencia e intercambio de archivos multimedia de mayor tamaño entre usuarios.

"La programabilidad intentada en años anteriores se vio potenciada también por la concepción y gran desarrollo en un elemento de hardware básico como es el procesador de red"(Morgan Kaufmann, 2008). consiste en un circuito integrado programable diseñado para aplicaciones de red como el tratamiento de paquetes, funciones de control de acceso y seguridad o calidad de servicio, identificando diferentes tipos o clases de paquetes.

Arquitectura de una red tradicional

En una arquitectura de red tradicional se conectan varios dispositivos a grandes distancias y satisfacen las necesidades de conectividad de sus usuarios. Para realizar una operación todos los paquetes tienen que pasar por routers, switches, firewalls y demás elementos que toman decisiones individuales de enrutamiento, los cuales hacen difícil e ineficiente en su gestión.

La mayoría de los dispositivos como routers o switches no son de código libre, lo que hace costoso obtener una red confiable. Además de necesitar la compatibilidad con otros dispositivos para interactuar reenvió de información y toma de decisiones.

Generalmente estos dispositivos tiene un software que es exclusiva del proveedor, por lo tanto estan bloqueados y solo las empresas pueden modificar a nivel de software. En esta arquitectura tiene dos planos: plano de control y plano de datos, como se observa en la figura 1.1.

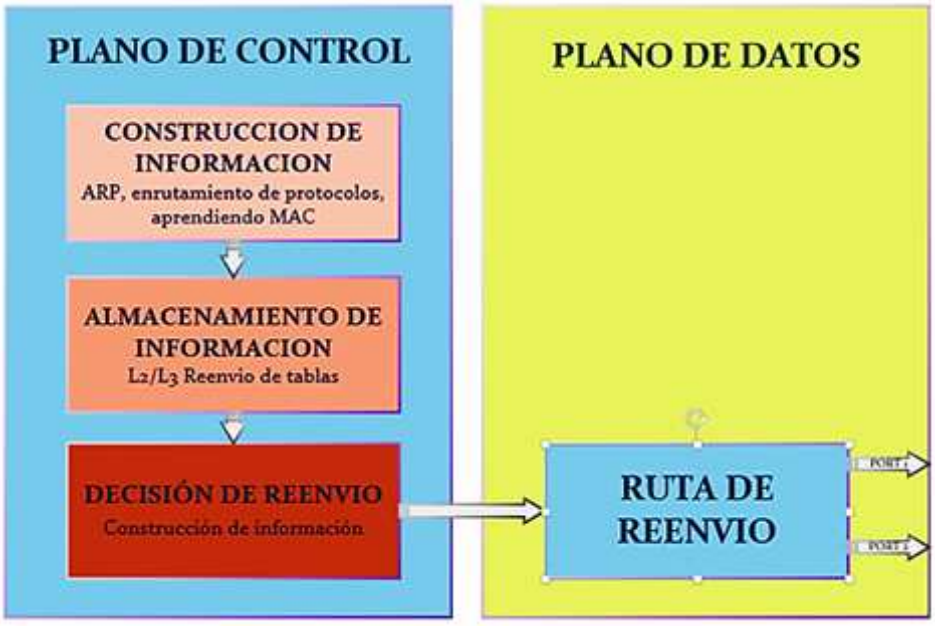


Figura 1.1: Arquitectura Tradicional
Fuente : Propia

Separando el plano de control y el plano de datos, se tienen las siguientes caracteristicas:

- El plano de control tiene software incorporado en el hardware del routers o switches, donde este ya esta determinado con respecto a las aplicaciones requeridas.
- El plano de datos es puramente conmutación de paquetes, debido a que el plano de control es el encargado de gestionar las órdenes.

En la red tradicional se combina el plano de control y el de datos en un solo dispositivo, siendo una red que no puede modificarse fácilmente haciendo de este un sistema rusticó.

Redes definidas por Software

"En 2011, Deutsche Telekom, Facebook, Google, Microsoft, Verizon y Yahoo! crearon la Open Networking Foundation (ONF)" (What is ONF, 2019), una organización sin ánimo de lucro dedicada a la promoción y adopción de SDN a través del desarrollo de estándares abiertos y que se apoyaba en el trabajo previo de investigadores de las Universidades de Stanford y Berkeley.

Ante necesidad de optimizar los recursos más frecuente se crean constantemente nuevos protocolos capaces de reducir costos y aumentar la eficiencia, tanto para los usuarios como para las empresas de telecomunicaciones.

Con la programación de redes se aborda varios problemas, facilitando la gestion de la red y servicios extremo a extremo. En la actualidad los protocolos de red se encuentran en evolución para optimizar y controlar el trafico de información, necesitando nuevas

formas de manejar la red y en SDN propone cambiar la manera rustica de manejar la red haciendola programable, donde pueden ser gestionadas remotamente, y de manera centralizada por un software controlador.

La separación evita esas tareas complejas al switch o dispositivo dando lugar a que el controlador encargado de automatizar y gestionar la red. Esta tecnologia pretende que las redes sean más eficientes, flexibles e incrementen su rendimiento permitiendo la configuración sin interrupción manual es decir, la posibilidad de programar a distintos requerimientos del usuario.

1.1. Antecedentes

Gracias a la revolución tecnologica se pudo transformar el entorno humano, ante la necesidad del alcance de información, se tuvo que añadir nuevas formas de comunicación conduciendo nacimiento del Internet.

"Inicialmente la información que se transmitía por la internet era mínima debido a los colapsos de la red que eran elevados, si un paquete de datos no podía atravesar algún punto de red se desviaba hasta encontrar otro camino por el cual poder llegar a su destino. En la actualidad los flujos de datos que se transmiten por red, como streaming de video, provocan problemas de colapso que hacen necesario actualizar la estructura de red." (Gonzalez, 2014)

"Al aumentar el flujo de información una red se vuelve más compleja, al aumentar la complejidad de una red aumentan las maneras de solucionar los distintos problemas. Por ello se formó numerosos grupos de investigación comenzaron el desarrollo de sistemas basados en este enfoque, actualmente existen varios protocolos para la comunicación entre el controlador y la infraestructura de red sobre los que se pueden desarrollar e implementar soluciones SDN. El despliegue operacional de este proyecto en la universidad de Stanford, comenzó la etapa de creación OpenFlow." (IETF 7149, 2014)

1.2. Situación Actual

Antes de abordar aspectos propios del proyecto, se expone el estado de situación, tendencias tecnológicas y otros trabajos académicos.

"En Bolivia el servicio de Acceso a Internet, era considerado como un servicio de valor agregado a los servicios tradicionales de voz, desde la llegada del servicio al país en el año 1996 con la creación de BOLNET hasta la gestión 2008 no tuvo un crecimiento significativo. Con la introducción de tecnologías 2.5G, 3G y 4G, la cantidad de conexiones de este servicio incrementó de forma acelerada." (ATT, 2018)

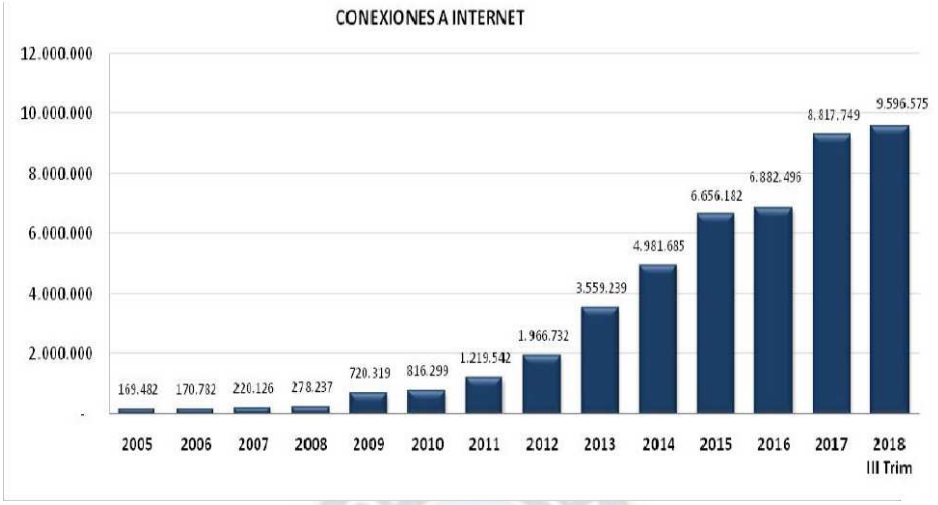


Figura 1.2: Evolución del Servicio de Acceso a Internet – a septiembre 2018
Fuente : Autoridad de Regulación y Fiscalización de Telecomunicaciones y Transportes

En 2018 el número de conexiones a Internet se incrementó en 778.826, de las cuales la mayor parte son establecidas a través de tecnologías 2.5G, 3G y 4G, fenómeno que se puede atribuir a la rebaja del precio de los Teléfonos Inteligentes (Smartphones) en el mercado boliviano y al desarrollo económico del país.

"Adicionalmente, las tecnologías de acceso fijo a Internet crecieron en 49,7 %. Se puede apreciar también en septiembre de 2018, aún permanecen un total de 451 conexiones por línea conmutada o dial up." (ATT, 2018)

Dando conocimiento a los datos proporcionados por la ATT (Autoridad de Regulación y Fiscalización de Telecomunicaciones y Transportes) observó que existe gran demanda para esta red que es el Internet; sin embargo denotamos que existe un ligero estancamiento de elementos de red.

Actualmente en Bolivia todavia no existen empresas que proporcione servicios SDN, evidenciando la falta de comunicacion y de inversion en esta tecnología.

1.2.1. Tendencias tecnologicas

"Existen dos nuevas tecnologias SDN (Software-Defined Networking) y NFV (Network Functions Virtualization) dirigidas a las redes en base en el software, para adaptarlas a las nuevas necesidades tecnológicas." (Leader, 2018)

Ambas complementarias se distancian de las redes de hardware para centrarlas en el software, generando un enfoque más flexible y ágil para el diseño, desarrollo y gestión de distintas situaciones:

- Soportar el creciente tráfico de datos.

- Satisfacer las necesidades de las nuevas tecnologías.
- Permitir la rápida innovación en redes.
- Reducir costos.
- Simplificar la gestión de red.

"SDN (redes definidas por software) nace de la experimentación universitaria. Se caracteriza por separar la capa de control de la de datos, para permitir un control centralizado, y ofrecer la capacidad de programar y automatizar el comportamiento de la red." (Leader, 2018)

De esta forma SDN es capaz optimizar el uso de los recursos de red, aumentar su agilidad y flexibilidad, permitir la innovación y el dinamismo de las redes, además de reducir gastos de capital y operativos.

Por su parte, NFV (virtualización de las funciones de red) surge de la iniciativa de proveedores de servicios de red como AT & T, BT, China Mobile, Deutsche Telekom, entre otros, permite implementar funciones de red mediante software donde virtualiza sólo sus funciones sobre servidores, switches y/o dispositivos de almacenamiento.

De esta forma, si queremos implementar; por ejemplo, una función de cifrado de red, en lugar de implementar con un nuevo dispositivo de hardware, podremos hacerlo con un software de cifrado en el servidor o en el switch, desde cualquier lugar dentro de la red.

NFV reduce la dependencia de dispositivos de hardware, aumentando la escalabilidad y personalización de la red de forma que sea más fácil mejorarla. Al evitar el hardware consigue reducir tanto el espacio que éste ocupa como su consumo energético, también de reducir los costes de mantenimiento de la red y su hardware.

Tanto SDN como NFV son tecnologías aún en desarrolló, representan el inicio de una nueva tendencia en networking, cuyo objetivo es, principalmente, adecuar la arquitectura de redes a las nuevas exigencias tecnológicas y las nuevas necesidades derivadas de la constante evolución tecnológica, simplificando la gestión de las redes.

En fin SDN y NFV son tecnologías y conceptos complementarios porque se dirigen a diferentes elementos de una solución basada en software.

1.2.2. Trabajos similares

Sobre la tecnología SDN existen diversos trabajos realizados pero se enfocan más en el controlador Floodlight debido a su completo repertorio de aplicación, a cerca del controlador RYU existen investigaciones fragmentadas, debido a su diseño basico.

A continuación mencionaremos los artículos y proyectos publicados en linea:

- Ariculo *"Uso de Mininet y Openflow 1.3 para la enseñanza e investigación en redes IPv6 definidas por software."* Line Yasmin Becerra-Sánchez, Bryan Valencia-Suárez, Santiago Santacruz-Pareja & Jhon Jairo Padilla-Aguilar, Facultad de

Ciencias Básica e Ingeniería, Universidad Católica de Pereira, Pereira, Colombia.

- Proyecto *"IMPLEMENTATION OF IEEE 802.1Q VLAN TAGGING USING RYU OPENFLOW CONTROLLER"* VARUN NAIR, Department of Electrical Engineering, Delft University of Technology

Ambos proyectos abordan diferentes temas y se enfoca en la modificación de los códigos para el uso del controlador.

1.3. Descripcion del problema

Actualmente en Bolivia las redes mas conocidas son eficaces, mas sus elementos son rígidos por lo que pronto no abastecerán a las nuevas nececidades tecnologicas. Ante esta situación se optó por crear nuevos dispositivos de red que solucionan problemas específicos, generando un problema más robusto la aparente solucion de añadir un dispositivo para cada problema y una solución diferente suponia añadir más elementos de red provocando gastos en infraestructura, entonces se propuso otras soluciones, la mas recomendada fue la implementacion de redes SDN.

1.4. Objetivos

1.4.1. Objetivos General

- Estudio y desarrollo del entorno de red SDN utilizando el protocolo OpenFlow y el controlador RYU.

1.4.2. Objetivos específicos

- Identificar los diferentes problemas a solucionar con el entorno SDN.
- Diseñar la red virtual para el Instituto de Electrónica aplicada mediante el uso de un software de virtualización en Mininet.
- Administrar el plano de control de los dispositivos, mediante un controlador Ryu, responsable de gestionar todo lo referente al comportamiento de la red.

1.5. Justificación

Con el aumento de nuevas tecnologías y el incremento de tráfico de información actual ha posibilitado el estudio redes definidas por software, que fueron pensadas para resolver problemas encontrados en el área de redes de datos.

"El desarrollo del proyecto SDN permitirá mejorar el rendimiento de las redes actuales, además se analizara conceptos nuevos, que aportaran el entendimiento de un nuevo tipo

de arquitectura." (Ramires, 2015)

"Con el estudio de las redes definidas por software se podrá solucionar las limitaciones que hoy en día presentan las redes de datos, tales como:" (Gonzalez, 2014)

- Configurar los equipos de red de manera individual.
- Soportes para nuevos servicios y capacidades.
- Dependencia de los usuarios a los fabricantes.

El estudio de distintas formas de manejar la redes de datos cobró gran importancia, ya que su uso incrementa a lo largo del tiempo con la llegada de la era digital.

Actualmente estan en auge iniciativas sobre la virtualización de la red que se centran en la configuración del software, mostrando avances rescatables, al emplear componentes de código libre que permiten estudiarlos con mayor libertad.

El proyecto se enfocó en el estudio de una red virtual simulada en SDN, en un entorno virtual Linux y con el sistema Ubuntu.

1.6. Alcances y limitaciones

De acuerdo a los objetivos planteados, para el proyecto se consiferan los siguientes alcances:

- Aprendizaje y caracteristicas de la tecnología SDN.
- Aprendizaje del manejo del controlador RYU, del protocolo OpenFlow y el emulador Mininet.
- Diseño de la red, en este caso la red del Instituto de Electronica Aplicada.
- Reconocimiento mediante el uso de los diferentes mensajes de OpenFlow y el Controlador RYU.
- Uso del controlador en Capa 2 y 3 del modelo OSI.

Tambien se consideran ciertas limitaciones:

- Los resultados se limitan a las pruebas del Simulador Mininet, lo que impide dar un diagnostico real para su aplicación, considerando también que el simulador Mininet se restringe al uso de un solo controlador.
- Debido a que es una tecnología emergente muchos de los equipos disponibles son incompatibles con OpenFlow v1.3 dificultando la aplicación del proyecto.
- La red del Instituto de Electrónica Aplicada se creó recientemente y la implementación de la tecnología se vio inejecutable debido a los switches existentes son limitados a la versión de OpenFlow v1.0.

1.7. Estrategia de Desarrollo

Para cumplir con los objetivos se trazó una estrategia de desarrollo que expone el proceso a realizar, plasmada en la siguiente figura.

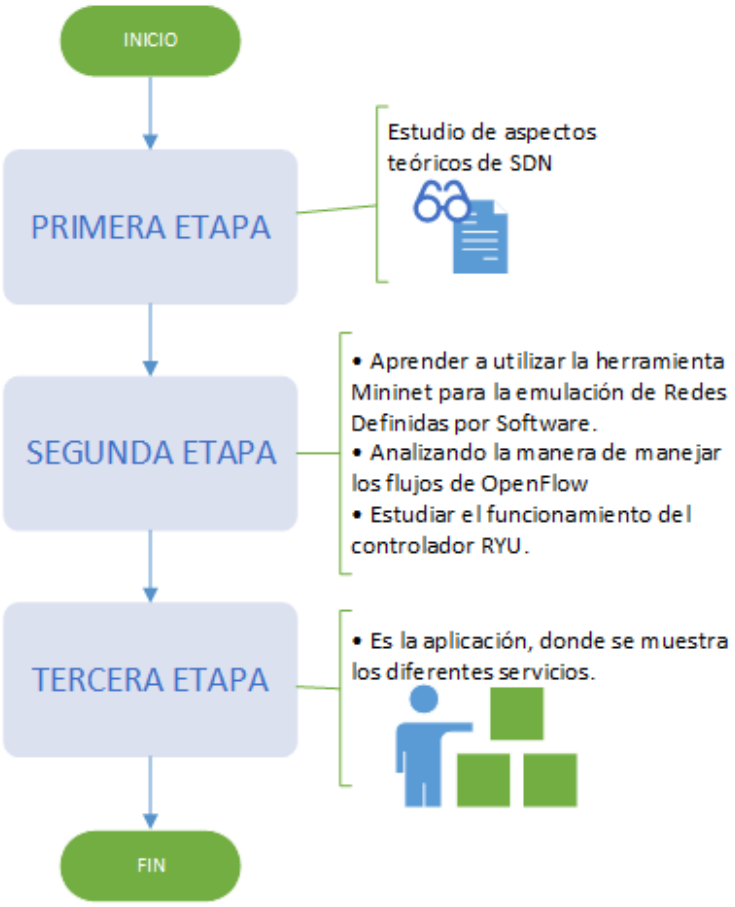


Figura 1.3: Diagrama de flujo para la estrategia de desarrollo
Fuente : Propia

2. Marco Teórico

Una red SDN nos presenta una nueva forma de crear una red volviendo a ella programable y manejable, debido a la nueva manera de establecer la arquitectura centrealizando el plano de control para gestionar y automatizar las funciones de red.

2.1. Arquitectura SDN

"Según la Open Networking Foundation, SDN es una arquitectura emergente que es dinámica, manejable, rentable y adaptable, por lo que es ideal para el alto ancho de banda. Esta arquitectura desacopla el funciones de control y reenvío de red que permiten que el control de la red se convierta directamente programable y la infraestructura subyacente a abstraer para aplicaciones y red servicios" (Cross, 2017). Las características más relevantes de SDN son:

- La separación del plano de datos y el plano de control.
- La centralización del plano de control, al que llamamos controlador y con la ayuda protocolo OpenFlow se conecta con el plano de datos.

En la arquitectura SDN muestra una red controlada remotamente y de manera centralizada, gracias a la separacion del plano de control y la de datos, dio inicio a innovadoras ideas con dirección a la evolución de redes. A continuación mostramos la separacion en la Figura 2.1.

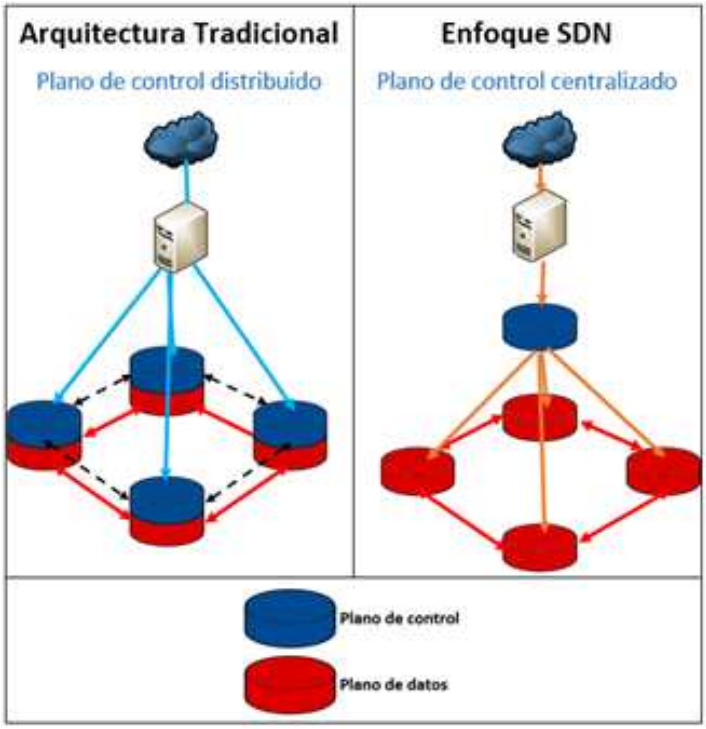


Figura 2.1: Arquitectura Tradicional VS Enfoque SDN.
Fuente : Propia

Al ser centralizado el plano de control, tuvo que incorporar el protocolo OpenFlow para conectarse al switch y para poder interactuar con el usuario, se dispone una capa de aplicación como se ve en la Figura 2.2.

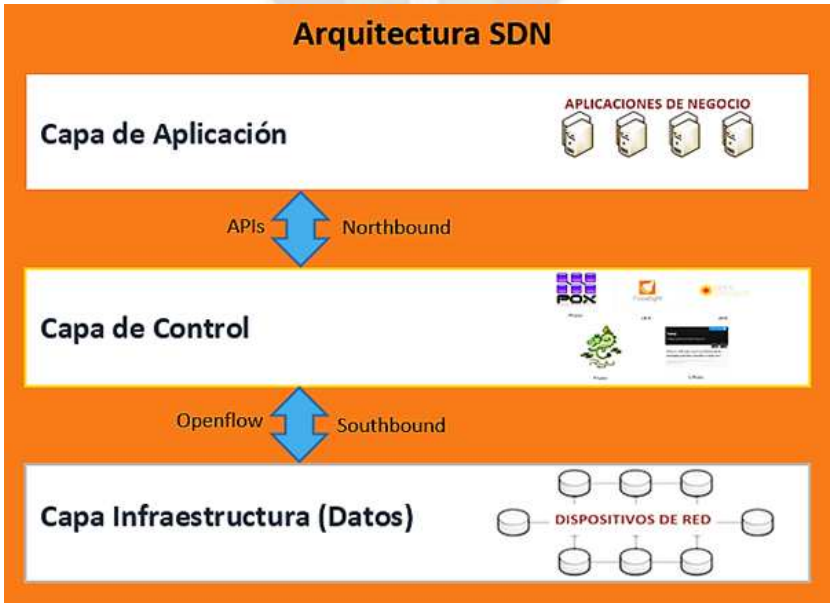


Figura 2.2: Arquitectura SDN
Fuente : Propia

2.1.1. Capa de Aplicación

La capa de aplicación es la capa de más alto nivel y está formada por aplicaciones de usuarios que inciden directamente en el controlador que nos permitirán simplificar y automatizar las tareas de configuración de la red; aprovisionar y gestionar nuevos servicios, así como extraer información de negocio, adaptandose automáticamente a las necesidades que vayan surgiendo. Esta capa no es capaz de comunicarse directamente con la capa de infraestructura. La capa se comunica con los controladores mediante APIs de la capa de control como REST, JSON, XML, etc.

Esta capa es principalmente para las necesidades del usuario, donde utiliza una interfaz denominada "Northbound API" para permitir comunicarse las aplicaciones con la capa de control.

2.1.2. Northbound

La interfaz northbound nos proporciona la comunicación o flujo que existe entre la capa de control y la de aplicación, es creada por el controlador que permite el soporte para la conmutacion, el enrutamiento, contrafuegos, etc.

2.1.3. Capa de Control

La capa de control es la capa más importante dentro de la arquitectura SDN, donde se gestiona la capa de aplicación e infraestructura y en ella esta ubicada la inteligencia de la red, donde es encargada de controlar y configurar los nodos de la red dirigiendo así los flujos de tráfico, es decir: el cerebro de la conmutación y enrutamiento de los paquetes de la red.

La comunicación entre el controlador y los dispositivos de la red a controlar, se realiza mediante la Southbound.

2.1.4. Southbound

La interfaz Southbound es la que comunica la capa de control y la capa de infraestructura. En este caso usamos el protocolo OpenFlow siendo el elemento esencial para separar la funcionalidad entre la capa de datos y de capa control.

2.1.5. Capa de Infraestructura(Capa de datos)

Es la capa más baja del modelo SDN y se compone de los dispositivos de red físicos que implementan la red mediante conexiones entre nodos, es un conmutador encargado de dar informacion a la interfaz Southbound.

2.2. Protocolo OpenFlow

Hasta el momento OpenFlow es el protocolo más aceptado e implementado en una arquitectura SDN, el protocolo ubica entre el controlador y un switch.

"El protocolo de OpenFlow proporciona una interfaz abierta para controlar la conectividad y los flujos dentro una SDN, es extensible y proporciona mecanismos para que los programadores en SDN definan elementos de protocolo adicionales (por ejemplo, nuevos campos de coincidencia, acciones, propiedades de puertos, etc.) para abordar nuevas tecnologías de red y comportamientos. Los patrones de tipo de Tabla OpenFlow es un vehículo donde describe una ruta de datos controlable, lo que permite a los proveedores de switches y controladores trabajar independientemente para crear productos SDN interoperables." (Goransson, Black, & Culver, 2014).

"Muchas grandes corporaciones dentro del mundo de las telecomunicaciones, entre las que se incluyen IBM, Google y HP, han dado su apoyo expreso a OpenFlow. Se trata de un protocolo estandarizado por la IETF (Internet Engineering Task Force) y gestionado por la Open Networking Foundation (ONF)" (What is OpenFlow? Definition and How it Relates to SDN, 2018)

- OpenFlow tiene la finalidad de dar comunicación del controlador al switch tanto de ida como de vuelta, estos mensajes son en forma de paquetes, permite al controlador crear flujos y programar los switches para conseguir un detallado control del tráfico.

- Permite a los switches encaminar el tráfico gracias a la información recibida del controlador y almacenada en sus tablas de flujos.

"En las redes SDN no existe la diferenciación entre routers, swicthes y firewall's, debido a que todos los dispositivos son considerados switches OpenFlow. Dentro del controlador se crearán reglas para los flujos según diferentes criterios, se comunicará a los switches mediante mensajes o paquetes OpenFlow. Estos criterios pueden ser de nivel dos (por MAC origen o destino), de nivel tres (por IP origen o destino) e incluso criterios de nivel cuatro (transporte) y nivel cinco (aplicación)" (SDN, 2017).

Openflow no sustituye a ninguno de los protocolos de la pila TCP/IP, cada mensaje va encapsulado en un segmento TCP, sus paquetes IP y trama Ethernet por lo tanto cualquier dispositivo de red puede encaminar el tráfico Openflow a su destino, no es responsable de encapsular y enviar ese tráfico, y sigue siendo una tarea que deben realizar los propios switches (y en ocasiones particulares el controlador) con los protocolos habituales.

Como todo Openflow es un protocolo que ha ido evolucionando con el tiempo y ha sufrido algunos cambios entre versiones, en el proyecto tomaremos la version 1.3 de OpenFlow

2.2.1. Tabla de flujos

Las tablas de flujo se componen de reglas, condiciones de coincidencia y acción, cuando se recibe un paquete; se analiza la tabla elemento a elemento para comprobar si se cumplen los requisitos de matching (coincidencia) en alguna de ellas. OpenFlow sigue un proceso general definido a la hora de recorrer las tablas de flujos en el switch. Este proceso o pipeline, es el siguiente.

- Las tablas de flujos en el switch se ordenan por números, empezando en 0.
- La tabla 0 debe existir siempre, ya que debe haber al menos una tabla de flujos en el switch.
- El proceso empieza siempre en esta primera tabla.
- Cuando entra un paquete se intenta asociar con alguna entrada de la tabla 0.
- En caso de encontrar una coincidencia, se añaden las instrucciones asociadas a ese flujo al denominado "action set" del paquete.

Action set es el conjunto de acciones que se aplican al paquete entrante, una vez han acabado de recorrerse las tablas de flujos. Si entre las instrucciones de la entrada se encuentra al pasar a otra tabla (Intrucción go to), se avanza a esa tabla y se repite el mismo proceso. El proceso completo se muestra en la siguiente figura:

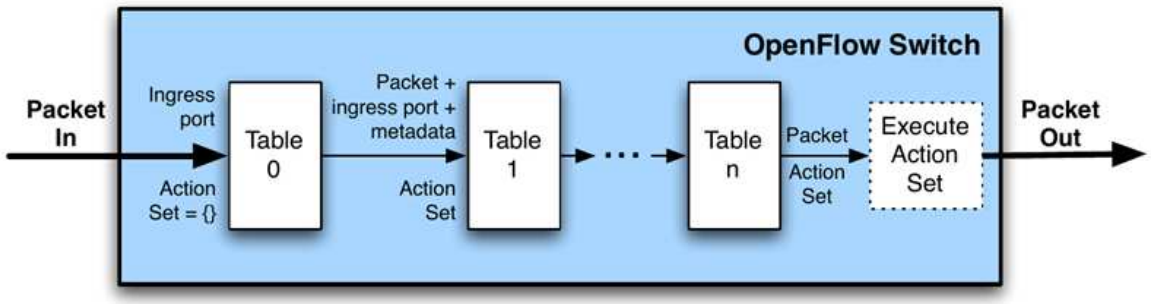


Figura 2.3: Proceso de las tablas de flujo o Pipeline
Fuente : Open Networking Foundation

En caso de que no exista una entrada asociada al paquete entrante en una tabla, se produce un fallo de tabla "table miss". Para este caso, el comportamiento dependerá de cómo se haya configurado la tabla. Se pueden añadir entradas en las tablas específicamente para indicar cómo procesar paquetes no equiparados (unmatched packets). Las opciones son:

- Enviar el paquete al controlador para que este se encargue de encaminarlo.
- Pasar el paquete a otra tabla del switch.
- Desechar el paquete.

Cuando el conjunto de instrucciones no tiene una instrucción de re dirección, el procesamiento se detiene en la tabla correspondiente el paquete se reenvía a algunos puertos de salida del switch.

El controlador OpenFlow tiene la capacidad de agregar, modificar o borrar la entrada de flujo en el switch según las instrucciones emitidas por el administrador de red (controlador).

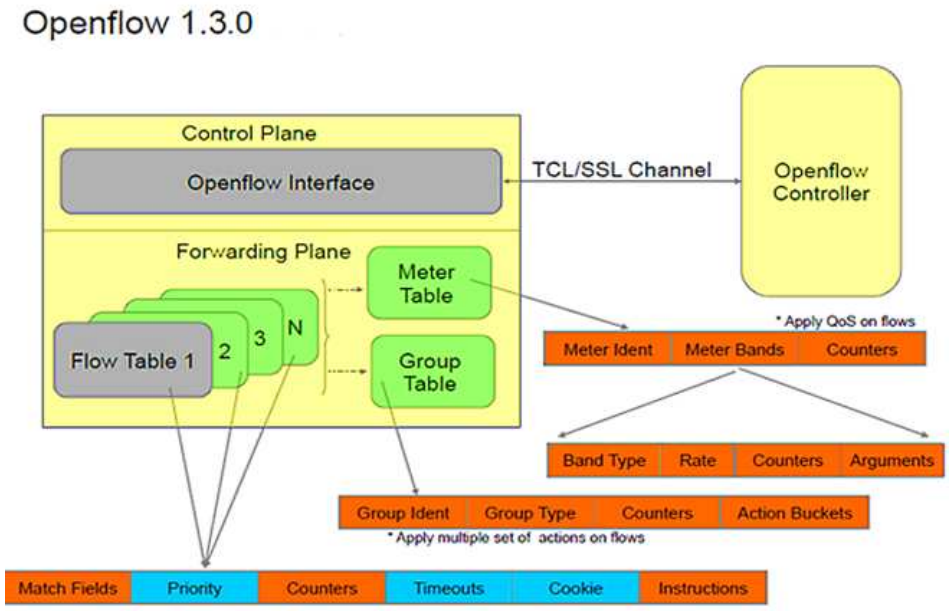


Figura 2.4: Arquitectura OpenFlow
Fuente : Analisis SDN con OpenFlow

- **Match Fileds:** Este campo contiene el parámetro coincidente con respecto al paquete, estos consisten en el puerto de ingreso y los encabezados de paquetes. Hay una variedad de condiciones que pueden especificar en un match, y cada vez que se actualiza OpenFlow se crean más en este caso se definen 40 tipos de match se verán en la Tabla 1.

Nro	Match field	Explicacion
1	`in_port`	Número de puerto del puerto receptor
2	`in_phy_port`	Número de puerto físico del puerto de recepción
3	`metadata`	Metadatos utilizados para pasar información entre tablas.
4	`eth_dst`	Dirección MAC de destino de Ethernet
5	`eth_src`	Dirección MAC de origen de Ethernet
6	`eth_type`	Tipo de trama de Ethernet
7	`vlan_vid`	ID de VLAN
8	`vlan_pcp`	VLAN PCP

Sigue en la página siguiente.

Nro	Match field	Explicacion
9	`ip_dscp`	IP DSCP
10	`ip_ecn`	IP ECN
11	`ip_proto`	Tipo de protocolo de IP
12	`ipv4_src`	Dirección IP de origen de IPv4
13	`ipv4_dst`	Dirección IP de destino de IPv4
14	`tcp_src`	Número de puerto de origen `tcp_src` de TCP
15	`tcp_dst`	Número de puerto de destino de TCP
16	`udp_src`	Número de puerto de origen de UDP
17	`udp_dst`	Número de puerto de destino de UDP
18	`sctp_src`	Número de puerto de origen de SCTP
19	`sctp_dst`	Número de puerto de destino de SCTP
20	`icmpv4_type`	Tipo de ICMP
21	`icmpv4_code`	Código de ICMP
22	`arp_op`	Opcode de ARP
23	`arp_spa`	Dirección IP de origen de ARP
24	`arp_tpa`	Dirección IP de destino de ARP
25	`arp_sha`	Dirección MAC de origen de ARP
26	`arp_tha`	Dirección MAC de destino de ARP
27	`ipv6_src`	Dirección IP de origen de IPv6
28	`ipv6_dst`	Dirección IP de destino de IPv6
29	`ipv6_flabel`	Etiqueta de flujo de IPv6
30	`icmpv6_type`	Tipo de ICMPv6
31	`icmpv6_code`	Código de ICMPv6
32	`ipv6_nd_target`	Dirección de destino del descubrimiento del vecino de IPv6
33	`ipv6_nd_sll`	Dirección de capa de enlace de origen del descubrimiento del vecino de IPv6
34	`ipv6_nd_tll`	Dirección de la capa de enlace de destino del descubrimiento del vecino de IPv6
35	`mpls_label`	Etiqueta MPLS
36	`mpls_tc`	Clase de tráfico MPLS (TC)
37	`mpls_bos`	MPLS BoS bit
38	`pbb_isid`	I-SID de 802.1ah PBB
39	`tunnel_id`	Metadatos sobre el puerto lógico
40	`ipv6_exthdr`	Pseudo-campo del encabezado de extensión de IPv6

Tabla 2.1: Match Fileds Fuente : RYU SDN Framework

- **Priority:** La tabla de flujo puede tener varias entradas de flujo para un paquete. Este campo establece la prioridad de la entrada de flujo para un paquete.

- **Counters:** Recuento de paquetes que llegan a la entrada de flujo. Los contadores se administran por puerto, por cola, por entrada de flujo, etc.
- **Timeouts:** Tiempo de validez para cada entrada de flujo o cantidad máxima de tiempo o tiempo de inactividad antes de que el interruptor expire el flujo.
- **Cookie:** Valor de datos opaco elegido por el controlador. Puede ser utilizado por el controlador para filtrar estadísticas de flujo, modificación de flujo y eliminación de flujo. No se utiliza al procesar paquetes.
- **Instruction:** El objetivo de la instrucción es definir qué sucede cuando se recibe un paquete correspondiente a la coincidencia. Se definen los siguientes tipos.

Nro	Instrucción	Explicacion
1	`Goto Table` (requerido)	La instrucción `Goto Table` indica la siguiente tabla de flujo en el pipeline, puede asumir el proceso de hacer coincidir paquetes con una tabla de flujo que se especifique. Por ejemplo, puede establecer una entrada de flujo como: Agregar un VLAN-ID a los paquetes recibidos en el puerto 1 y enviarlo a la tabla 2. El ID de tabla que especifique debe ser un valor mayor que el ID de tabla actual.
2	`Write Metadata` (opcional)	Establece los metadatos a los que se puede hacer referencia en la siguiente tabla.
3	`Write Actions` (requeridas)	Agrega una acción que se especifica en el conjunto actual de acciones. Si ya se ha configurado el mismo tipo de acción, se reemplaza con la nueva acción.
4	`Apply Actions` (opcional)	Aplica inmediatamente la acción especificada sin cambiar el conjunto de acciones.
5	`Clear Actions` (opcional)	Elimina todas las acciones en el conjunto de acciones actual.
6	`Meter` (opcional)	Aplica el paquete al medidor que se especifique.

Tabla 2.2: Instrucción
Fuente : RYU SDN Framework

Las siguientes clases correspondientes a cada instrucción se implementan en Ryu en funcion a la tabla 2.2.

- OFPInstructionGotoTable
- OFPInstructionWriteMetadata
- OFPInstructionActions

- OFPInstructionMeter

Las acciones Write/Apply/Clear se agrupan en OPFInstructionActions y se seleccionan en el momento de la creación de instancias.

La clase **OFPActionOutput** se usa para especificar el reenvío de paquetes que se usará en los mensajes de Emisión de paquetes y Modificación de flujo. Se especifica el tamaño de datos máximo (`{max_len}`) que se transmitirá al controlador y el destino en los argumentos del constructor, aparte del número de puerto físico del conmutador, algún valor definido.[9]

Nro	Acción	Explicacion
1	`OFPP_IN_PORT`	Reenviado al puerto de recepción
2	`OFPP_TABLE`	Se aplica a la primera tabla de flujo.
3	`OFPP_NORMAL`	Reenviado por la función de interruptor L2 / L3
4	`OFPP_FLOOD`	Inundado a todos los puertos físicos de la VLAN, excepto los puertos bloqueados y los puertos receptores.
5	`OFPP_ALL`	Reenviado a todos los puertos físicos excepto a los puertos receptores
6	`OFPP_CONTROLLER`	Enviado al controlador como un mensaje de entrada de paquete.
7	`OFPP_LOCAL`	Indica un puerto local del conmutador
8	`OFPP_ANY`	Está destinado a ser usado como comodín cuando selecciona un puerto utilizando los mensajes Flow Mod (eliminar) o Solicitudes de estadísticas, y no se usa en el reenvío de paquetes.

Tabla 2.3: OFPAction
Fuente : RYU SDN Framework

2.2.2. Entradas en la tabla de flujo

OpenFlow controla el tráfico agregando reglas, que es la combinación de identificadores de cada capa. Cada una de estas reglas se almacena en la tabla de flujo. Una entrada de flujo también consiste en acciones o instrucciones junto con las reglas match.

Cuando se recibe algún paquete en un switch openflow lleva a cabo diferentes instrucciones que se muestran en el diagrama de flujo:

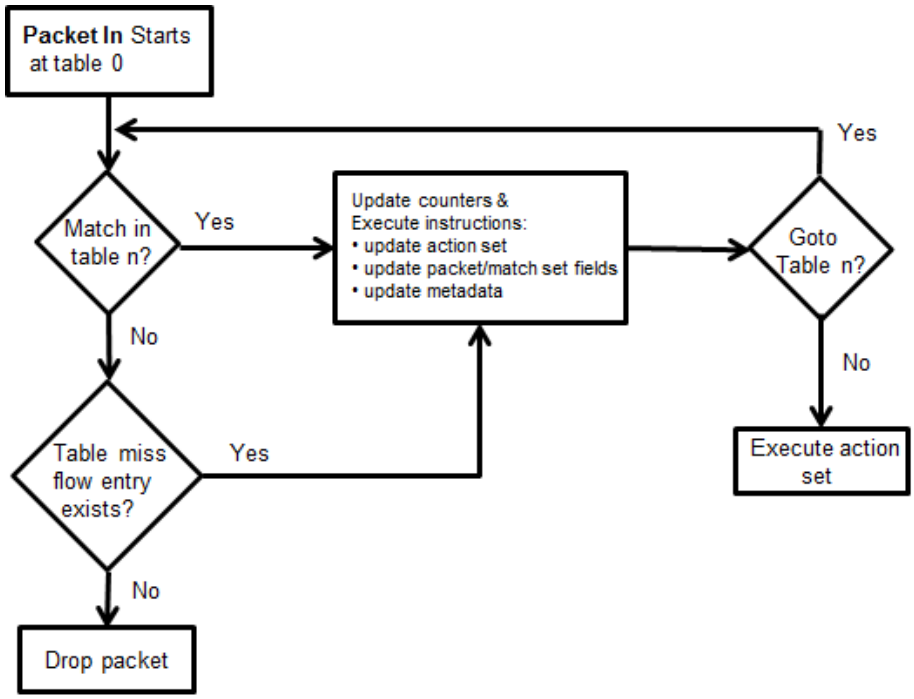


Figura 2.5: Diagrama de Flujo de un Switch OpenFlow
Fuente : Open Networking Foundation

En los dispositivos compatibles y sistemas hibridos en OpenFlow, la tabla de flujo reutilizará el hardware existente, como las TCAM que se utilizan en los Switches y Routers modernos.

El usuario puede definir las entradas de flujo a través del controlador que son insertados en el TCAM. El paquete encapsula los protocolos Ethernet, IP y TCP, etc.

TCAM (ternary content-addressable memory) es un tipo de memoria especializado, de alta velocidad que busca todo su contenido en un solo ciclo de reloj.

El término “ternario” se refiere a la capacidad de la memoria para almacenar y consultar datos utilizando tres entradas diferentes: 0, 1 y X. La entrada “X”, que a menudo se denomina estado de “no importa” o “comodín”, permite a TCAM realizar búsquedas más amplias basadas en la coincidencia de patrones, a diferencia de la CAM binaria, que realiza búsquedas de coincidencia exacta utilizando solo 0s y 1s.

Los Switch OpenFlow pueden contar el tráfico según las reglas de X(comodín) que coinciden con los bits en el encabezado del paquete, incluidas las direcciones IP y los números de puerto TCP / UDP. Las reglas de X(comodín) encajan con la memoria direccionable de contenido ternario (TCAM) disponible en muchos conmutadores.

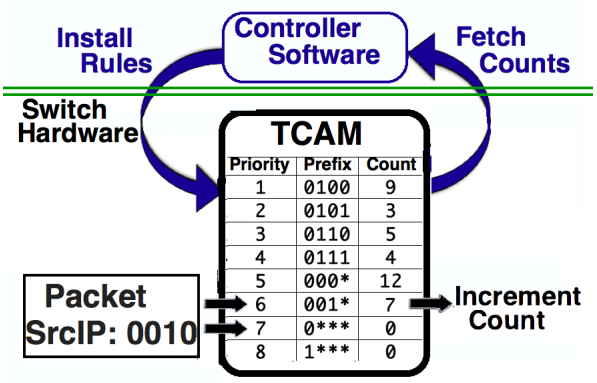


Figura 2.6: TCAM
Fuente : Online Measurement of Large Traffic Aggregates on Commodity Switches

Al procesar un paquete, el conmutador identifica las reglas coincidentes, selecciona la regla con la prioridad más alta, actualiza el contador asociado y realiza alguna acción (por ejemplo, descartar o reenviar el paquete).

2.2.3. Tipos de mensajes OpenFlow

La especificación OpenFlow 1.3 básicamente define tres tipos de mensajes: simétrico, asíncrono y controlador a switch.

Los **mensajes simétricos** son utilizados con el switch tanto como el controlador para saber si existe conexión son 3 y se muestran en la siguiente tabla:

Nro	Mensaje	Informacion
1	`Hello`	Intercambiado al inicio de la conexión.
2	`Echo`	Determina la latencia y el ancho de banda del controlador y la conexión del interruptor.
3	`Experimenter`	Proporciona una forma estándar de ofrecer funcionalidades adicionales o como base para las futuras revisiones.

Tabla 2.4: Mensajes Simetricos.
Fuente : Propia

Los **mensajes asíncronos** se utilizan para informar al controlar de cambios en el estado del switch y para informar sobre eventos en la red, estos mensajes son enviados del switch al controlador son 4 y se muestran en la siguiente tabla:

Nro	Mensaje	Informacion
1	`Packet-in`	Mensaje es usado cuando un paquete que ha llegado no coincide con ninguna entrada definida en la tabla de flujo y si el switch admite el almacenamiento en búffer.
2	`Flow-removed`	Es utilizado por los controladores para eliminar los flujos de ruta de los switches.
3	`Port-status`	Es utilizado durante la configuración del puerto.
4	`Error`	El controlador es notificado por los interruptores de error.

Tabla 2.5: Mensajes Asíncronos.
Fuente : Propia

Los **mensajes controlle**r to switch pueden enviar directamente mensajes al switch para administrar su estado, estos se utilizan para gestionar y programar el switch, son 3 y se muestran en la siguiente tabla:

Nro	Mensaje	Informacion
1	`Features`	El controlador entiende las capacidades del switch y el switch responde con una respuesta que la caracteriza.
2	`Configuration`	Este tipo es utilizado por el controlador para cambiar los parámetros de configuración en el switch.
3	`Packet-out`	Se utiliza para enviar paquetes desde un puerto en el switch.

Tabla 2.6: Mensajes Controller to Switch.
Fuente : Propia

2.2.4. Software Switch

En el proyecto se considera dos tipos de Switch, cada uno de los Switch son compatibles con OpenFlow 1.3, y ambos son emulables con Mininet para producir una estimación del comportamiento de la red durante el diseño.

CPqD Software Switch

Actualmente hay un buen número de Switch de hardware para probar OpenFlow, pero la mayoría de ellos todavía implementa solo la versión 1.0 del protocolo, que carece de las nuevas características de las versiones más recientes. Por lo tanto, para no tener innovación dependiente de hardware, los conmutadores de software se están implementando desde las versiones más primitivas de OpenFlow.

El Switch de software OpenFlow 1.3 se basa en el conmutador de referencia Stanford OpenFlow 1.0 y en el conmutador OpenFlow 1.1 Traffic Lab de Ericsson y está diseñado para propósitos de experimentación rápida.

Admite la mayoría de las funciones de OpenFlow 1.3 y, lo que es más importante, las características del medidor. Desafortunadamente, hay algunos errores existentes en el conmutador de software CPqD actual, como no reconocer las máscaras de bits IP.

Open vSwitch

"Según OVS.org, Open vSwitch (OVS) es un conmutador de software de código abierto diseñado para ser utilizado como un 'virtual switch' en entornos de servidor virtualizados. El objetivo de OVS es implementar una plataforma de conmutación que permita una gestión estándar, independiente del proveedor Interfaces y abre las funciones de reenvío de conmutadores a extensión programática y control." (Open vSwitch FAQs, 2019). Es compatible con todas las versiones del protocolo OpenFlow. Usando la herramienta ovs-ofctl, cualquier versión deseada de OpenFlow puede implementarse en un interruptor físico.

"Simplemente, Open vSwitch es un 'Software de switch' que implementa el protocolo OpenFlow para la conmutación de hardware."(Open vSwitch FAQs., 2019) Administra las tablas de flujo para los Datapaths que se utilizan para reenviar el tráfico entrante de acuerdo con las entradas coincidentes. La estructura de las entradas de la tabla de flujo se explica en la tabla 2.7.

Match Fields	Priority Counters	Instructions	Timeout	Cookies

Tabla 2.7: Tabla de flujo de entrada
Fuente : Open vSwitch

2.3. Controladores OpenFlow

Un controlador gestiona los recursos de la red, se comunica con los dispositivos para obtener la información global del estado de la red y las configuraciones específicas para controlar el comportamiento el envió y enrutamiento.

Como se ve en la figura 2.8 una visión general de la arquitectura SDN, se observa que está compuesta por tres capas: Aplicación, Control y Datos.

El cerebro de una red SDN se encuentra centralizada en la capa de control, por lo cual se han desarrollado varios controladores que son básicamente software que comanda y controla el switch habilitado para OpenFlow, maneja y manda el flujo de información se utilizan controladores los cuales tienen las siguientes características:

	Beacon	Floodlight	NOX	POX	Trema	Ryu	ODL
Soporte OpenFlow	OF v1.0	OF v1.0	OF v1.0	OF v1.0	OF v1.3	OF v1.0, v1.2, v1.3 y extensiones Nicira	OF v1.0
Virtualización	Mininet y Open vSwitch	Mininet y Open vSwitch	Mininet y Open vSwitch	Mininet y Open vSwitch	Construcción de una herramienta virtual de simulación	Mininet y Open vSwitch	Mininet y Open vSwitch
Lenguaje de desarrollo	Java	Java	C++	Python	Rudy/C	Python	Java
Provee REST API	No	Si	No	No	Si (Básica)	Si (Básica)	Si
Interfaz Gráfica	Web	Web	Python+, QT4	Python+, QT4, Web	No	Web	Web
Soporte de plataformas	Linux, Mac OS, Windows y Android para moviles	Linux, Mac OS, Windows	Linux	Linux, Mac OS, Windows	Linux	Linux	Linux, Mac OS, Windows
Soporte de OpenStack	No	Si	No	No	Si	Si	Si
Multiprocesos	Si	Si	Si	No	Si	No	Si
Código Abierto	Si	Si	Si	Si	Si	Si	Si
Tiempo en el mercado	4 años	2 años	6 años	1 años	2 años	1 años	5 meses
Documentación	Buena	Buena	Media	Pobre	Media	Media	Media

Tabla 2.8: Características de los controladores
Fuente : OpenStack

Estos controladores son algunos más populares para este tipo de red, y todos ellos vienen con una GUI y un REST API para la conexión con el usuario, también proporcionan comandos para habilitar y configurar las rutas a través del protocolo OpenFlow.

En la Tabla 2.8 se expone las diferentes características de los controladores encontrados, se muestra la comparación de los controladores y sus principales características también destacando que todos son OpenSource y que permiten la virtualización mediante Mininet.

3. Ingenieria del Proyecto

SDN brinda la virtualización de redes, economiza la escala una red donde una solución basada en software resulta más economica que una basada en hardware, cambiando las caracteristicas del controlador se pueden afrontar los diversos problemas.

Considerando los datos técnicos explicados en la tabla 2.8, se procede a implementar los diferentes casos explicados en la siguiente seccion, por consiguiente se implementaran en el emulador Mininet los cuales se aplicaran en el controlador RYU.

3.1. Productos a ser desarrollados

Para el uso de la tecnología SDN se debe contar con:

- Conocimiento previo del modelo OSI.
- Conocimiento básico de lenguaje python.
- Manejo de aplicaciones en Ubuntu.

En el proyecto se tomó como ejemplo la red del Instituto de Electrónica Aplicada (IEA) para dicha simulacion se require:

a) Manejar el emulador Mininet.

b) Aprender a utilizar el controlador RYU.

c) Saber la topología de la red IEA.

A cada inciso se le dedicara un caso respectivamente, expuestos en el siguiente capitulo

A continuación se explicara los diferentes casos para el uso del emulador en Mininet:

- En el primer caso se propone una red simple como de guía que usa Mininet para el buen funcionamiento del emulador(Capítulo 4, Sección 4.1)
- En el segundo caso se tendrá la misma red propuesta guía que usa Mininet para el buen funcionamiento del emulador, con la diferencia de que se empleara el controlador externo RYU, para observar las características de la Capa de enlace de datos (Capa 2 del modelo OSI). (Capítulo 4, Sección 4.2)

- En el tercer caso se verá el uso de la Capa de red (Capa 3 del modelo OSI), enrutamiento de paquetes mediante el direccionamiento lógico y el control de subredes; para este fin utilizaremos la topología de la red de IEA. (Capítulo 4, Sección 4.3)

- En el cuarto caso se tendra la aplicacion de QoS para la red, posibilitando el Balance de Carga y el uso de Servicios diferenciados. (Capítulo 4, Sección 4.4)

3.2. Resultados Esperados

- Proporcionar fundamentos teóricos aplicables en las redes tradicionales.
- Demostrar que SDN es una buena alternativa y/o solución al posiblilitar significativamente los gastos de operación (OPEX) y adquisición de infraestructura (CAPEX).
- Exponer el uso de SDN para distintos servicios que requieren equipos dedicados a un solo tipo de servicio.
- Generar 4 modelos para la comprension de las redes SDN.
- Desarrollar una estructura sencilla, que permita la comprensión OpenFlow y que utilice RYU como controlador.
- Difundir el proyecto para dar paso a las nuevas formas de creación de redes.
- Viabilizar el uso de la nueva tecnología.
- Fortalecer las actividades de investigación en SDN.

3.3. Equipo de trabajo

Se implementa en una máquina virtual ejecutada desde VirtualBox. Esta máquina funciona un sistema operativo Linux en una de sus distribuciones Ubuntu 16.04.5.

La PC debe tener las siguientes caracteristicas:

- Procesador i5 o superior.
- Memoria disponible de 200Gb.
- Memoria RAM de 8Gb.
- Sistema Operativo compatible con Virtual Box.

Una vez instalada la maquina virtual con Ubuntu, debemos proceder con las instalaciones de Mininet y RYU.

3.4. Entorno de simulación

Explicamos cada uno los elementos y software que se utilizara en la simulación, se propone dar una visión general; como encajan para el diseño de la red ya propuesta en el Instituto de Electrónica Aplicada.

En la figura 3.1 se muestra el diagrama de bloques del esquema físico del proyecto, en el que figuran los elementos principales. Comenzando desde fuera, se encuentra el PC, desde donde se realiza el trabajo.

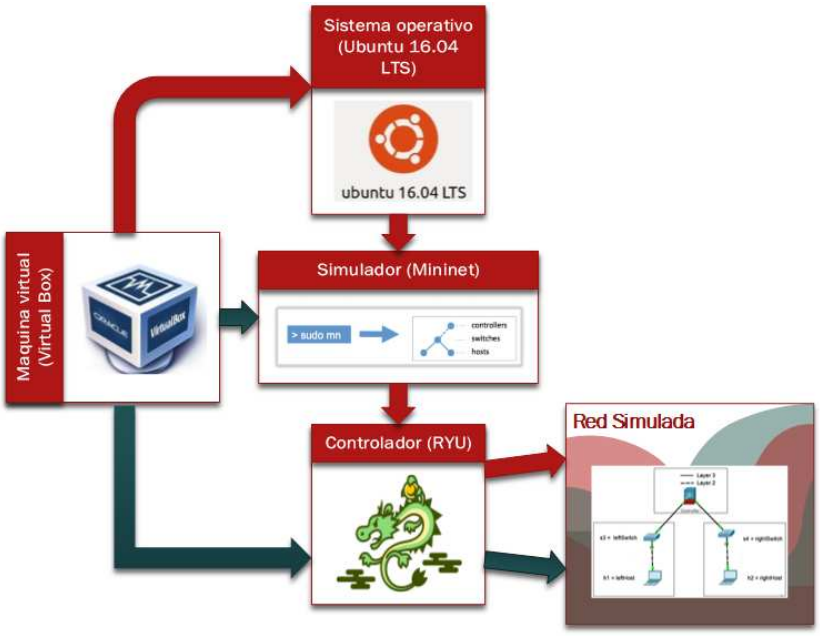


Figura 3.1: Diagrama del proyecto
Fuente : Propia

En la figura 3.1 se propone dos alternativas para la virtualización de una red:

- La primera es instalar Ubuntu dentro de Virtual Box y una vez en Ubuntu instalar internamente los paquetes de Mininet y Ryu para la virtualización de la red.
- La segunda es instalar los archivos ISO de Mininet y Ryu en Virtual Box y proceder a la virtualización de la red.

En el proyecto se muestra la primera opción debido a la variedad de pruebas realizadas, también por ser más dinámico y practico.

3.5. Herramientas

En esta sección se describen brevemente las herramientas utilizadas para simular la red virtual y asi también las pruebas para el diseño propuesto.

3.5.1. Mininet

Mininet es un emulador de red escrito en Python y C., es compatible con múltiples switches OpenFlow, permite crear redes virtuales de gran escala. Es una plataforma muy utilizada para el estudio SDN y puede implementar el protocolo OpenFlow en sus distintas versiones como con diversos controladores.

Mininet tiene las siguientes características:

- **Flexibilidad:** Topologias y caracteristicas nuevas que se pueden configurar por software usando lenguajes de programación.
- **Aplicabilidad:** Se puede crear cualquier topología y además mediante lenguaje se pueden implementar en redes basadas en Hardware sin cambiar su código fuente.
- **Interactiva:** Administración y simulación ocurren en tiempo real.
- **Escalabilidad:** Ambiente de en un prototipo es escalable a redes grandes con un solo computador.
- **Entorno Real:** Comportamiento del prototipo representa un uno real con alto grado de confianza.

Para el uso de Mininet se explicarán 4 tipos de topologias descritas a continuación:

Nro	Topología	Descripción	Codigo
1	`Single`	Un switch conectado a N hosts.	`$sudo mn --topo single,N`
2	`Linear`	Cada switch se conecta con otro de forma linear, y cada switch tiene un host conectado.	`$sudo mn --topo linear,N`
3	`Tree`	Crea una topologia de arbol con profundidad N y anchura M.	`$sudo mn --topo tree, depth=n,fanout=m`
4	`Custom`	Una topología customizada es una creada por el usuario, para este fin es necesario crear un archivo en python con la topología deseada.	`$sudo mn --custom mytopo.py --topo mytopo`

Tabla 3.1: Topologias Mininet
Fuente : Propia

Mininet utiliza la virtualización basada en procesos para ejecutar switches y hosts dentro del sistema operativo, se convierte en una herramienta práctica y util a la hora de hacer pruebas con SDN, puede ser instalado en un sistema Linux existente o se puede descargar en una máquina virtual, tomando en cuenta que Mininet por defecto

usa NOX como controlador para su red inicial, debido a eso es necesario incluir en el codigo el uso de un controlador externo.

Instalacion de Mininet

El proceso de la instalación de Mininet es el siguiente: (Mininet, 2019)

1. Ingresar al terminal de Ubuntu (por defecto Ubuntu tiene instalado python en su sistema operativo).
2. Ingresar el comando siguiente para clonar el repositorio e instalando manualmente:

Comando 3.1 : Instalacion de Mininet

```
1 git clone git://github.com/mininet/mininet
```

3. Debemos hacer el "checkout" explicando la versión en este caso será la versión 2.2.2:

Comando 3.2 : Versión de Mininet

```
1 cd mininet
2 git tag
3 git checkout -b 2.2.2
```

4. Una vez completado con los 2 pasos se debe usar el script install.sh para instalar los diferentes paquetes de mininet:

Comando 3.3 : Instalacion de paquetes Mininet

```
1 mininet/util/install.sh
```

Comprobación de funcionamiento.

Para comprobar la instalación se ingresa:

Comando 3.4 : Instalacion de paquetes Mininet

```
1 sudo mn   test   pingall
```

```
elizabeth@VirtualBox:~$ sudo mn --test pingall
*** 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 ...
*** Waiting for switches to connect
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 5.745 seconds
```

Figura 3.2: Prueba de Mininet.
Fuente : Propia

3.5.2. RYU

"RYU es un marco de red definido por software basado en componentes, proporciona componentes de software con una API bien definida que facilita a los desarrolladores la creación de nuevas aplicaciones de control y gestión de red, es compatible con varios protocolos para administrar dispositivos de red, como OpenFlow, Netconf, OF-config, etc. Acerca de OpenFlow, es compatible con versiones de OpenFlow 1.0, 1.2, 1.3, 1.4, 1.5 y Nicira Extensions. Todo el código está disponible gratuitamente bajo la licencia Apache 2.0" (Ryu SDN Framework, 2019).

RYU significa 'flujo' en japonés. Ryu se pronuncia 'ree-yooh'ree-yooh (Ryu SDN Framework, 2019).

El controlador está completamente escrito en Python y es uno de los más populares controladores debido a que es OpenSource, compatible con muchas bibliotecas de terceros que incluyen enlaces de Python, ellos deben ser específicos para la gestión del tráfico de red. Usa una biblioteca de codificadores y decodificadores para paquetes OpenFlow y contiene un función principal ejecutable llamado ryu-manager donde escucha una IP específica la dirección y en el puerto 6633, el puerto estándar de OpenFlow. Los componentes principales en el RYU.

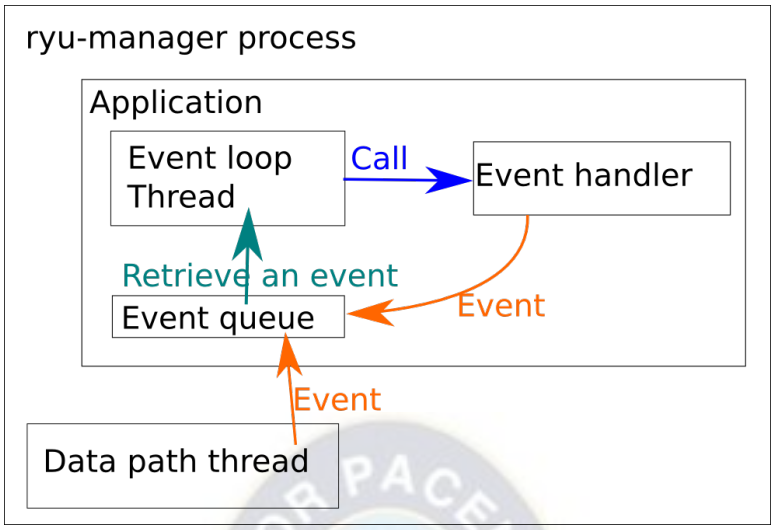


Figura 3.3: Manejo de Eventos RYU.
Fuente : RyuBook

El ryu-manager se utiliza para gestionar los mensajes RYU se compone en grupos de sub-procesos para encargarse de los paquetes entrantes, procesos, visores de topología y generadores de paquetes, viene con varias aplicaciones ya escritas para implementar la red SDN, como Spanning Tree manager, la aplicación de conmutación básica para crear reglas de reenvío, firewall y aplicación de enrutamiento. La arquitectura de Ryu se muestra en la Figura 3.4.

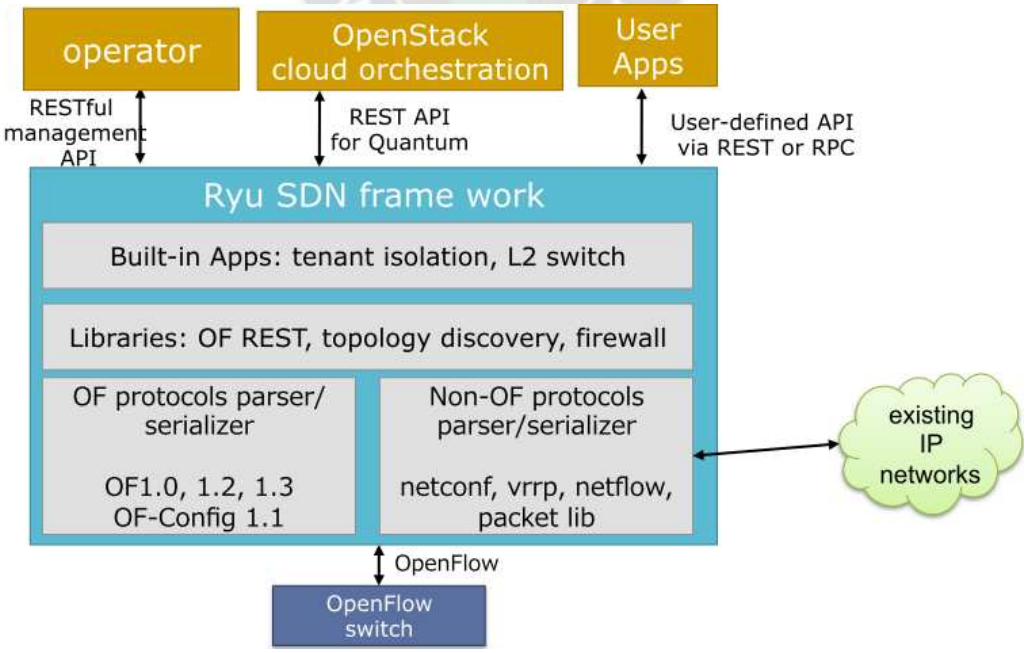


Figura 3.4: Arquitectura RYU.
Fuente : Ryu SDN Framework

Este controlador gestiona los diferentes mensajes OpenFlow, y para recibir los mensajes manda a una cola de recepción para estos eventos llamada FIFO que se utiliza para mantener el orden de los eventos entrantes, conservando el orden de llegada de los diferentes mensajes.

Instalación RYU

El proceso de ryu instalación es el siguiente: (Ryu SDN Framework, 2019).

Ingresar al terminal de Ubuntu se debe ingresar el siguiente comando para clonar el repositorio e instalando manualmente:

Comando 3.5 : Instalacion de RYU

```
1 git clone git://github.com/osrg/ryu.git
2 cd ryu; python ./setup.py install
```

Para poder designar el controlador utilizaremos `ryu-manager` con el `yourapp.py` o con la topología insertada en mininet.

3.5.3. Wireshark

Wireshark es uno más de los software OpenSource que esta programado en C y es multiplataforma, Wireshark en un inicio es conocido como Ethereal B, es un analizador de protocolos recibe una copia de los mensajes que están siendo recibidos o enviados en el terminal donde está ejecutándose.

Está compuesto principalmente de dos elementos: una librería de captura de paquetes, recibe una copia de cada trama de enlace de datos que se envía o recibe, un analizador de paquetes, muestra los campos correspondientes a cada uno de los paquetes capturados. Para realizar esto, el analizador de paquetes ha de conocer los protocolos que está analizando de manera que la información mostrada sea coherente, es decir, si capturamos un mensaje HTTP, el analizador de paquetes ha de saber que este mensaje se encuentra encapsulado en un segmento TCP, que a su vez se encuentra encapsulado en un paquete IP y este a su vez en una trama de Ethernet.

Para instalar Wireshark desde la fuente el comando es el siguiente:

Comando 3.6 : Instalacion de Wireshark.

```
1 sudo apt-get -y install wireshark
```

Cabe resaltar que esta instalación no es necesaria ya que viene instalada con los paquetes de Mininet.

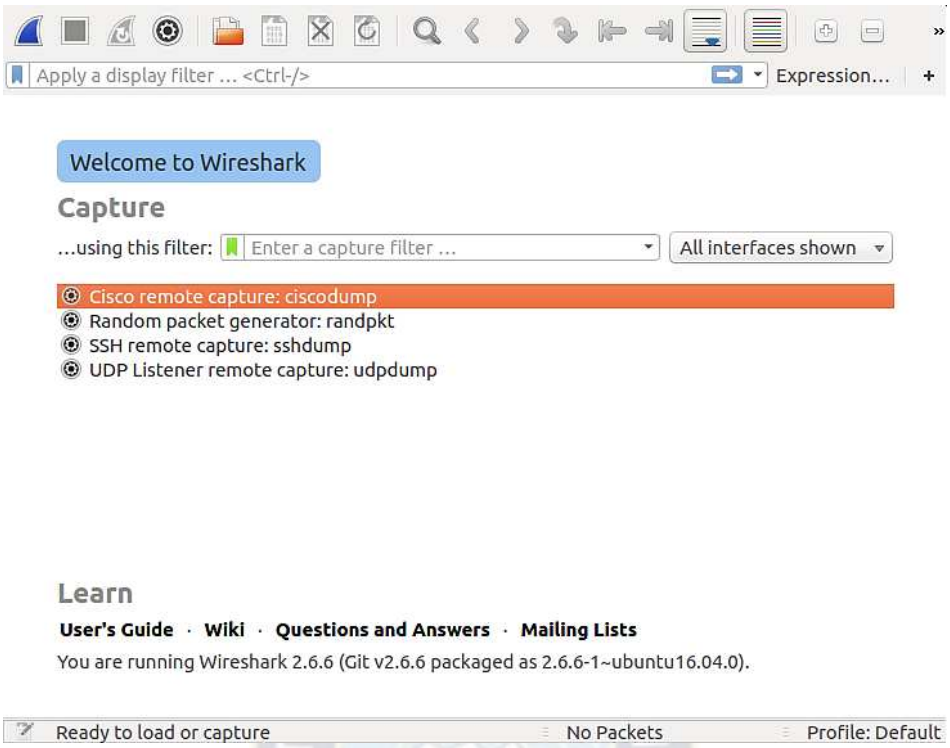


Figura 3.5: Inicio de Wireshark.
Fuente : Propia

3.5.4. VirtualBox

VirtualBox es una aplicación de virtualización multiplataforma. Esto significa que se instala en computadores existentes basados en Intel o AMD, y está disponible para los sistemas Windows, Mac, Linux y Solaris. Amplía las capacidades del computador existente para que puedan ejecutar múltiples sistemas operativos (dentro de múltiples máquinas virtuales) al mismo tiempo.

Este software se puede decir que es simple pero también muy poderoso, puede ejecutarse desde cualquier lugar, de pequeños sistemas integrados o máquinas de escritorio a implementaciones de centros de datos e incluso entornos en la nube.

En el proyecto, emplea VirtualBox para crear una máquina virtual con el sistema operativo Linux en una distribución de Debian, específimente Ubuntu en su versión 16.04.2. con el fin de que una vez se haya completado el trabajo, la máquina virtual pueda ser clonada y empleada en diferentes entornos de prácticas con todo el software necesario ya instalado y listo para el aprendizaje.

3.6. Costo del proyecto

El proyecto se realizó la implementación de SDN, debido a que esta responde a los requierimientos de la red. Actualmente en Bolivia existen pocos dispositivos compatibles

y esta tecnología aún está en aprendizaje para múltiples usuarios(Cisco, HP, Huawei, etc), ante esas dificultades se opto por hacer las redes SDN de manera emulada con ayuda de un software libre, evitando gastos de compra en productos de software como de hardware.

Se recurrió a Opex para considerar los costos, pues incluyen gastos de investigación, desarrollo y tiempo en la siguiente tabla 3.2 se desgloza el costo del proyecto.

Herramientas	**Tiempo estimado**	**Costo (Bs)**
Computador personal (Laptop)	6 meses	1600
Curso de Python	1 mes	250
Introduccion SDN	1 mes	0
Emulador Mininet	2 meses	0
Controlador Ryu	3 meses	0
Distintas librerias y programas	3 meses	0
Electricidad (70Bs x mes)	6 meses	420
Conexion a Internet (180Bs x mes)	6 meses	1080
Implementacion y pruebas (1600Bs x mes)	6 meses	9600
	Total	**12950**

Tabla 3.2: Costos
Fuente : Propia

La implementacion de la tecnologia SDN posibilita reducir significativamente los gastos de operación (OPEX) y adquisición de infraestructura (CAPEX), permitiendo ofrecer innovación en ámbitos aplicativos.

Por ejemplo se reemplazan los routers o firewalls, por un switch generando una red más simple para manejar los paquetes.

4. Desarrollo del proyecto

En este capitulo se tomó los diferentes casos descritos en el Capitulo 3, Seccion 3.1, donde se apartó su distinta funcionalidad; desde el uso más básico del emulador, hasta la implementación del controlador Ryu y su aplicación.

4.1. *Caso 1:* Configuracion de una red simple en Mininet

En este caso vemos una red simple, mostrando la introduccion al emulador Mininet, donde se tomó la topologia tipo custom descrito en el capitulo 3, Seccion 3.5.1, donde se expone la siguiente topología:

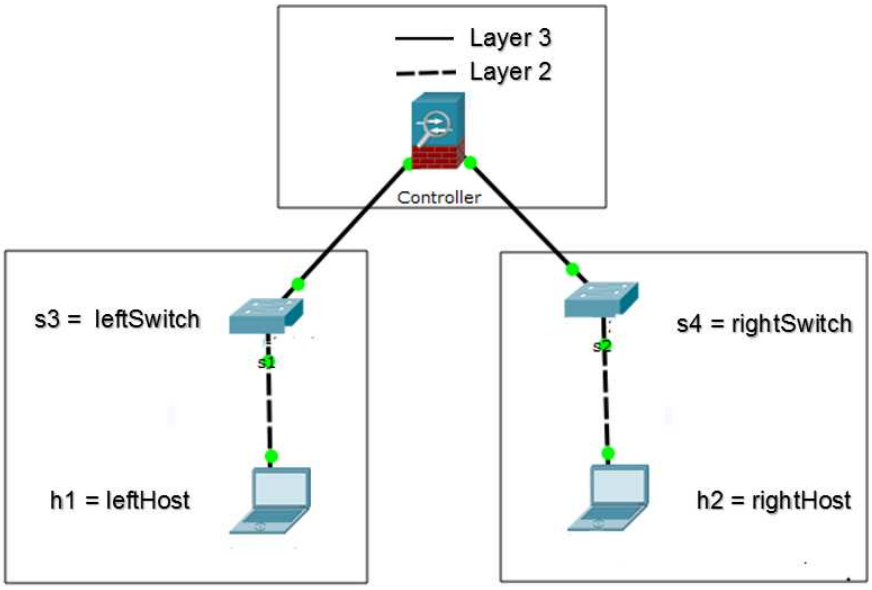


Figura 4.1: Topologia custom de Mininet.
Fuente : Propia

Donde se utiliza un codigo ya creado que esta en la siguiente figura:

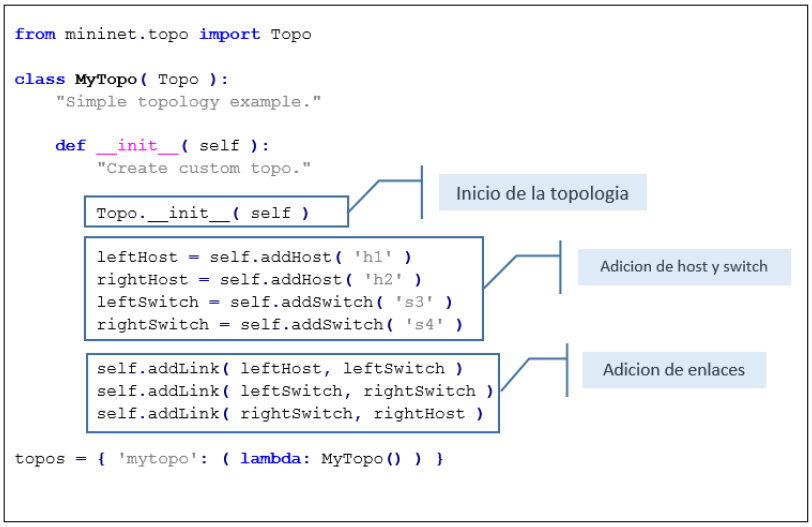


Figura 4.2: Codigo de topologia simple.
Fuente : Propia

Una vez conocida la topologia y el codigo de la red a implementar, se procedio a ejecutar la red donde se coloca en la terminal de Ubuntu el siguiente comando de Mininet:

Comando 4.1 : Topología Tipo Custom.

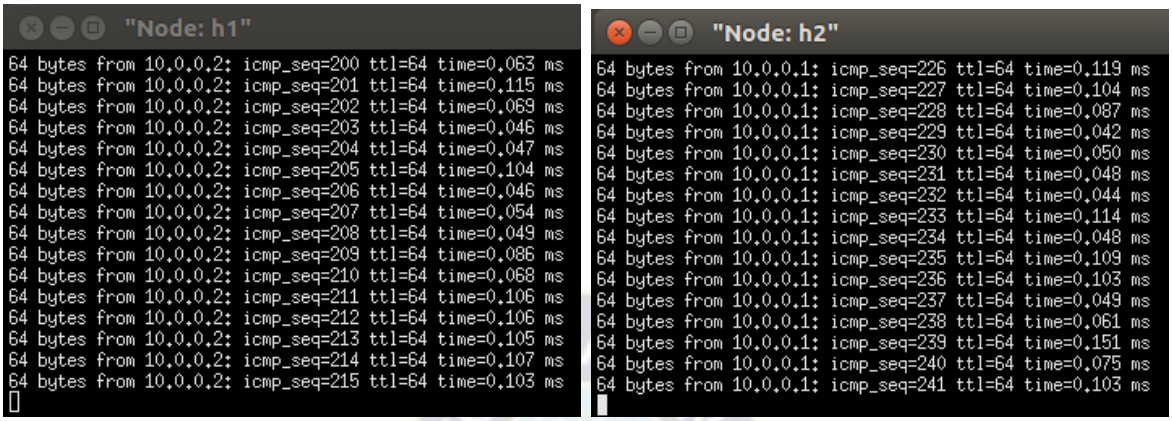
```
sudo mn --custom ~/mininet/custom/topo-2sw-2host.py --topo mytopo
```

Como podemos observar en la figura 4.3 la red puede funcionar correctamente ya desde un inicio eso se debe a la buena instalacion del emulador Mininet y utiliza el controlador NOX que es uno de los controladores más antiguos.

Figura 4.3: Compilacion de topologia custom en Mininet.
Fuente : Propia

Para verificar las conectividades se realiza en la terminal de `xterm` de cada host un

`ping 10.0.0.2` que es de la h2 a h1 y tambien se tiene en viceversa `ping 10.0.0.1` y se muestra en la figura 4.4.



(a) Ping h1 a h2 (b) Ping h2 a h1

Figura 4.4: Ping de topología tipo Custom.
Fuente : Propia

En la figura 4.4 se verificó la correcta conectividad entre los 2 host.

Para tener más conocimiento de Mininet se vera un Manual con comandos de Mininet en el Anexo A.

4.1.1. Prueba con Wireshark para topologia custom sin el controlador

Para realizar las distintas pruebas se abrió un Xterm del controlador en la terminal de Mininet llamando a wireshark:

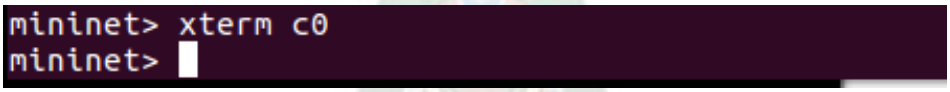


Figura 4.5: Terminal Mininet.
Fuente : Propia

Figura 4.6: Llamado a wireshark
Fuente : Propia

Y colocando el trafico en Any el cual nos muestra todas las conexiones.

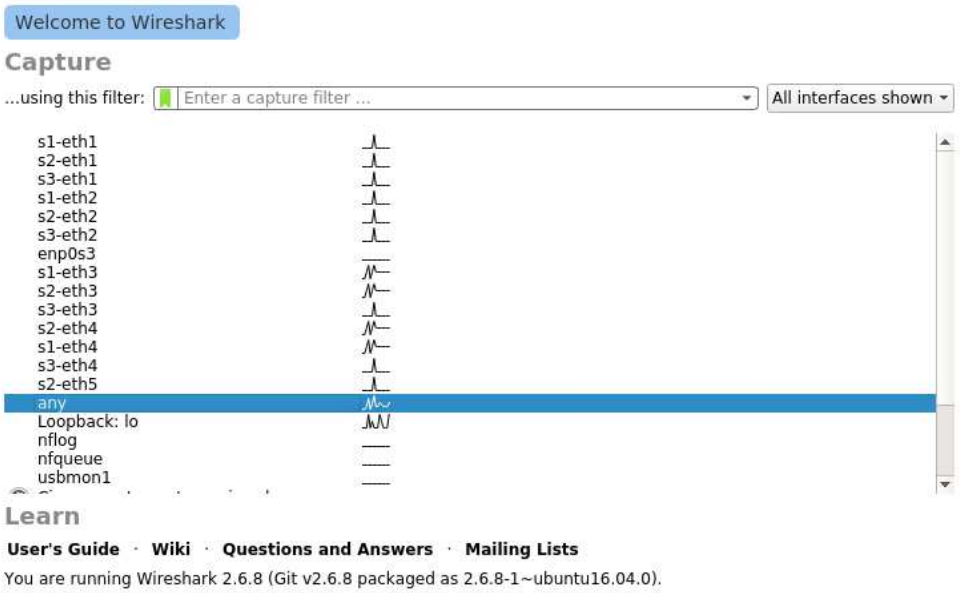


Figura 4.7: Inicio Wireshark.
Fuente : Propia

Wireshark nos ayudó a verificar las capacidades de trafico tanto TCP y UDP, se hizó varias pruebas y se muestra en la figura 4.8 el trafico y los mensajes OpenFlow.

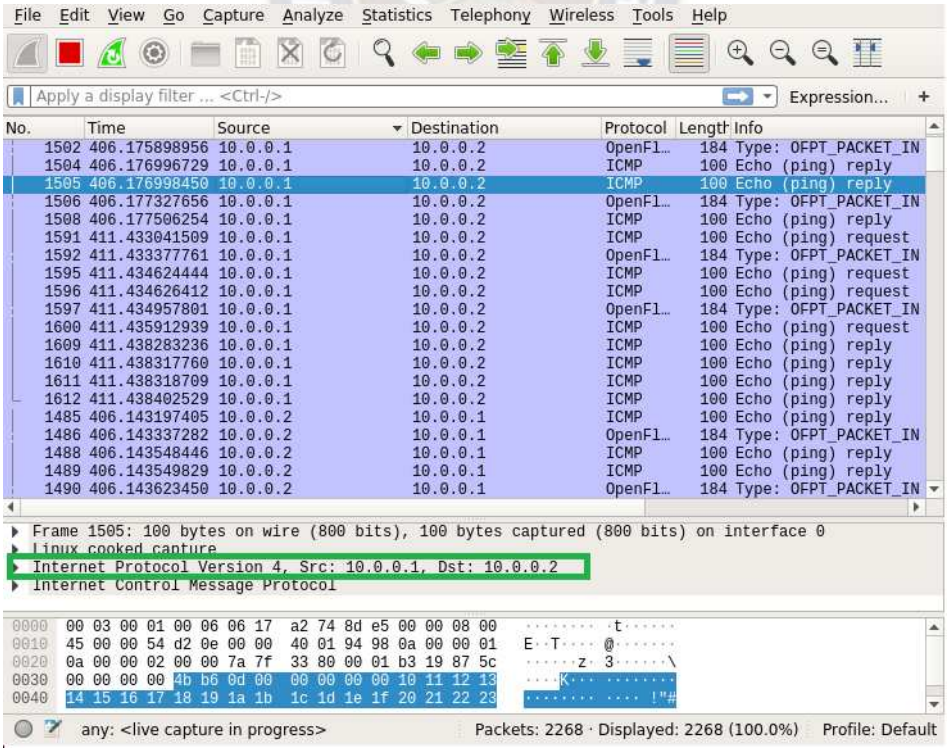


Figura 4.8: Trafico de topología tipo Custom.
Fuente : Propia

En la figura 4.8 se muestra los diferentes mensajes OpenFlow que se logra ver con la ayuda del controlador por defecto de Mininet.

4.1.2. Analisis de Resultados *Caso 1*

En este caso se vió la introducción a una red SDN se expuso una red por defecto del emulador de red SDN mininet, se comprobó la facilidad de su uso y su correcto funcionamiento del emulador.

4.2. *Caso 2:* Configuracion de una red simple con el controlador RYU

Dando referencia a una aplicación básica de un elemento de red se toma en consideración el switch, debidó a que es uno de elementos más utilizados y conocidos en la industria.

Para poder determinar una equidad entre la forma de Switching en RYU y la forma de un switch tradicional se tomó en cuenta la variedad de funciones del Switching:

- Aprende la dirección MAC y conserva en una tabla de direcciones MAC.
- Cuando recibe paquetes dirigidos a un host ya aprendido, transfiere al puerto conectado al host.
- Al recibir paquetes dirigidos a un host desconocido, realiza una inundación.

A continuación implementaremos RYU para realizar un Switching y para probar las aplicaciones del controlador RYU utilizaremos la topología de Mininet ya descrita en la anterior sección pero esta vez con un controlador externo que en este caso es RYU y tendrá que colocar en la terminal el comando:

Comando 4.2 : Topología Tipo Custom controlador RYU.

```
1 sudo mn --custom ~/mininet/custom/topo-2sw-2host.py --topo mytopo --
    mac --switch ovsk --controller remote -x
```

Dando enfasis en `-controller remote -x` el cual nos permite usar un controlador externo, en este caso Ryu.

```
elizabeth@VirtualBox:~$ sudo mn --custom ~/mininet/custom/topo-2sw-2host.py --t
opo mytopo --mac --switch ovsk --controller remote -x
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2
*** Adding switches:
s3 s4
*** Adding links:
(h1, s3) (s3, s4) (s4, h2)
*** Configuring hosts
h1 h2
*** Running terms on :0
*** Starting controller
c0
*** Starting 2 switches
s3 s4 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet>
```

Figura 4.9: Topología Tipo Custom controlador RYU.
Fuente : Propia

Al ejecutar el comando nos mostrara en la figura 4.9 las características de la red creada en este caso vemos que el controlador es remoto.

Para habilitar el controlador remoto RYU utilizaremos el `ryu-manager` y la aplicación `simple_switch_13.py`, escritos en el siguiente comando:

Comando 4.3 : ryu-manager para Switching simple.

```
1 ryu-manager --verbose ryu.app.example_switch_13
```

Una vez implementado el comando en la terminal, el controlador procede a enviar y recibir mensajes OpenFlow., el cual se muestra en la siguiente figura:

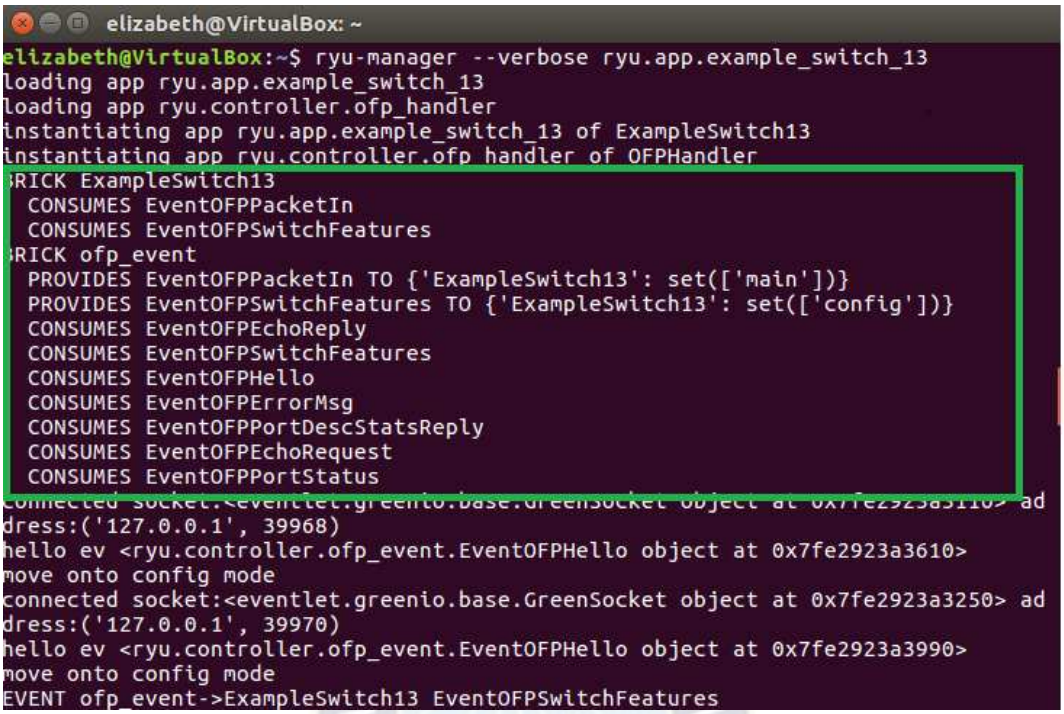


Figura 4.10: Implementacion de controlador RYU en la red.
Fuente : Propia

Como se ve en la figura 4.10 RYU utiliza eventos y son descritos en el Capitulo 2.

La implementación de `simple_switch_13.py` ejemplíca los procedimientos básicos para implementación de una aplicación RYU y un método sencillo de controlar el conmutador OpenFlow, donde el codigo esta descrito en el Anexo D.

4.2.1. Prueba con Wireshark para topologia custom con el controlador

Procedemos con el uso de wireshark, donde describiremos cada mensaje OpenFlow.

En la figura 4.10 vemos el establecimiento de la conexión en el controlador, pero estos mensajes se pueden ver más claramente haciendo una captura con la herramienta Wireshark y se visualiza en la figura 4.10:

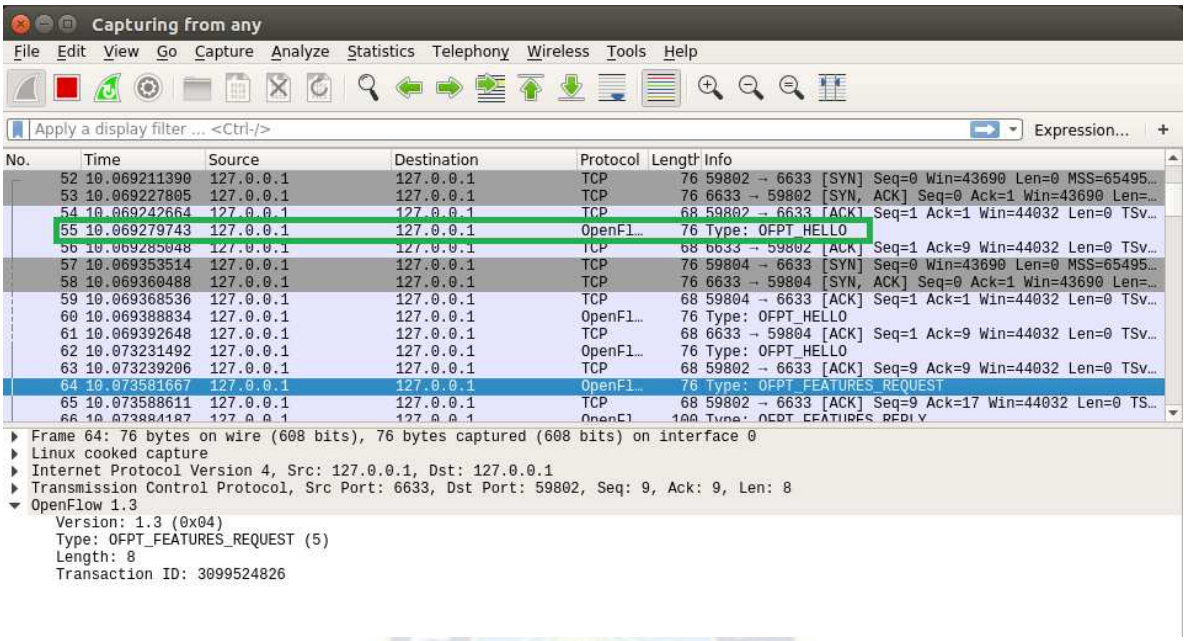


Figura 4.11: Proceso de conexion entre Mininet y Ryu con Wireshark.
Fuente : Propia

Al analizar la Figura 4.11, se vió como la conexión entre el controlador y el switch se lleva a cabo mediante el intercambio de mensajes `HELLO`, si analizamos el contenido del paquete `OFPT_HELLO` mostrado en la 4.12, vemos que en él podemos encontrar campos que establecen la versión del protocolo OpenFlow utilizado, el tipo de mensaje OpenFlow y su longitud.

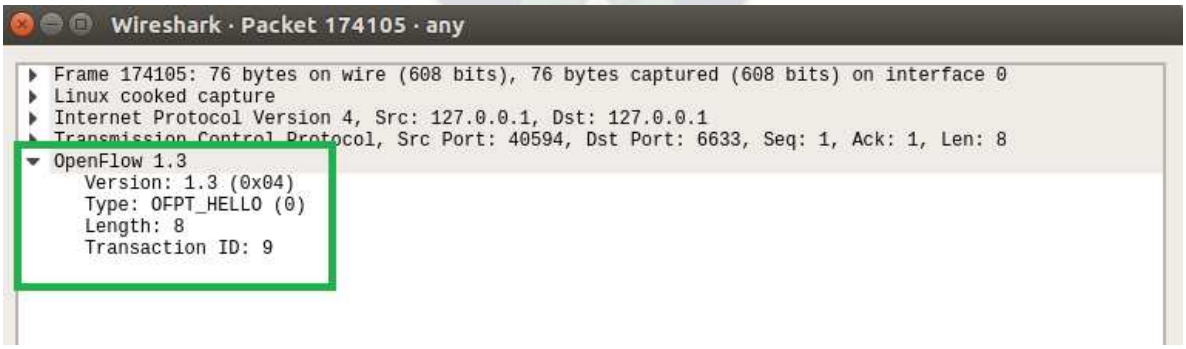


Figura 4.12: Mensaje OFPT_HELLO.
Fuente : Propia

Despues de los mensajes `OFPT_HELLO` aparecen dos mensajes de tipo `OFPT_Request`. Uno de ellos es de petición y otro de respuesta. Con estos mensajes, el controlador solicita que el switch se identifique y este responde a esta petición.

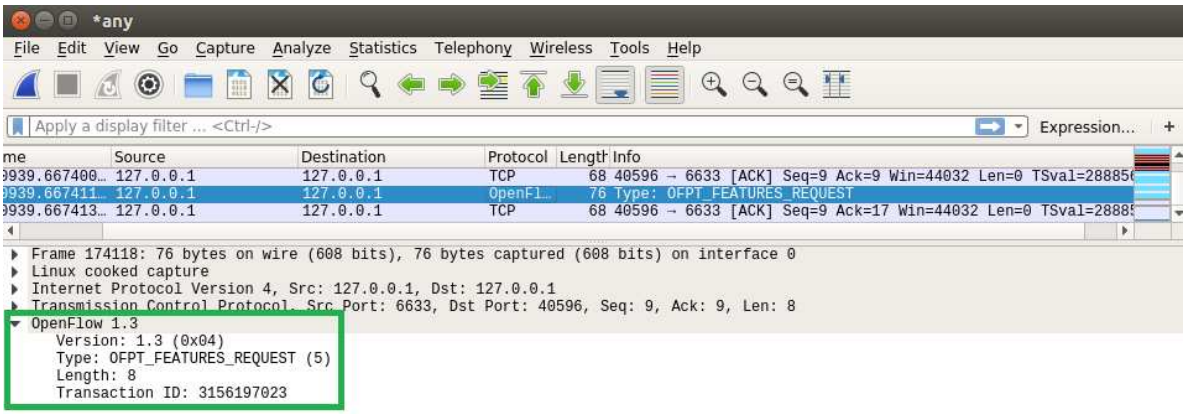


Figura 4.13: Mensajes OFPT_features_request.
Fuente : Propia

Con el mensaje mostrado en la Figura 4.13, se estableció la petición por parte del controlador sobre la identificación y las capacidades del switch. En la Figura 4.14 mostrada a continuación, el Switch manda al controlador su identificación y sus características.

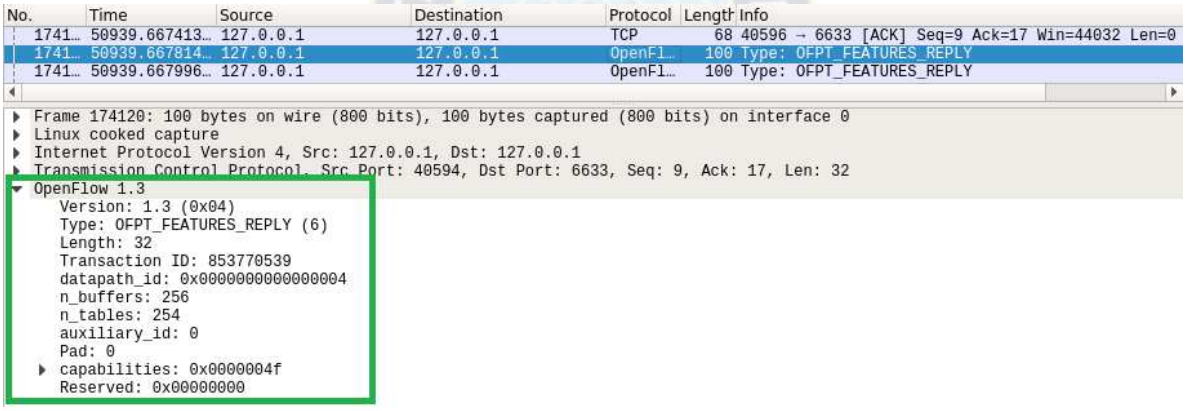


Figura 4.14: Mensajes OFPT_features_reply.
Fuente : Propia

A continuación, se establece el flujo de paquetes del tipo `PACKET_IN` y `PACKET_OUT`. En la figura 4.14 mostrada a continuación se vió el contenido de los paquetes del tipo `PACKET_IN`.

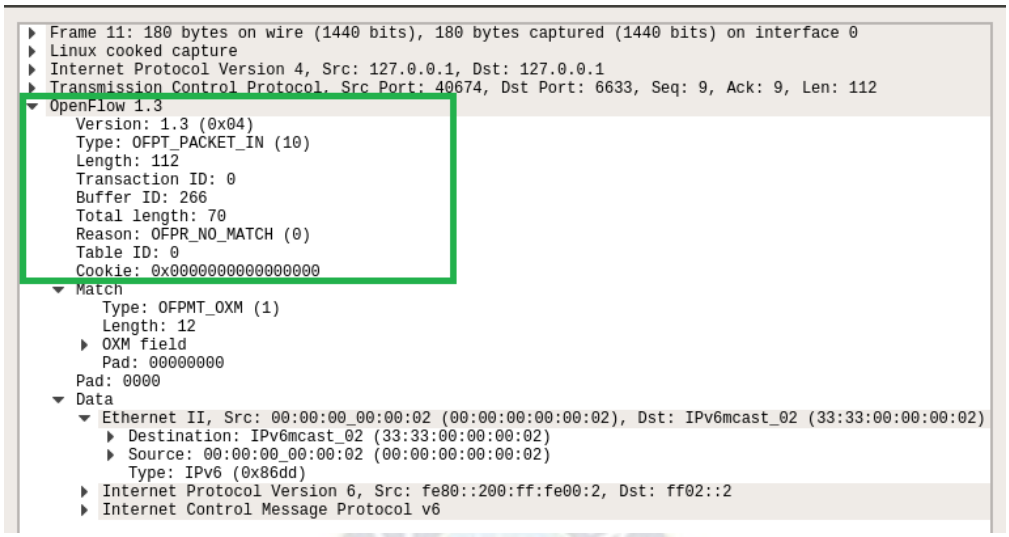


Figura 4.15: Mensaje en el controlador PACKET_IN.
Fuente : Propia

Como se vió en la figura, donde aparece la versión del protocolo OpenFlow utilizada, además de un paquete de tipo Ethernet con los campos de origen y destino del paquete.

En la figura 4.16, se muestra el mensaje OpenFlow del tipo `PACKET_OUT`; la información que se muestra en la captura es la acción asociada a este paquete. En este caso la acción asociada es la de descartar un paquete almacenado en el buffer.

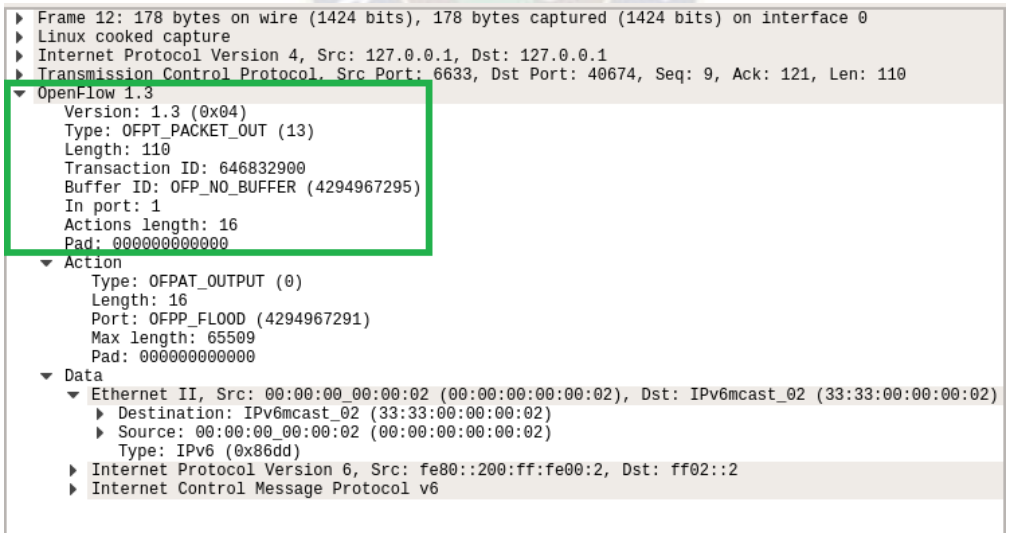
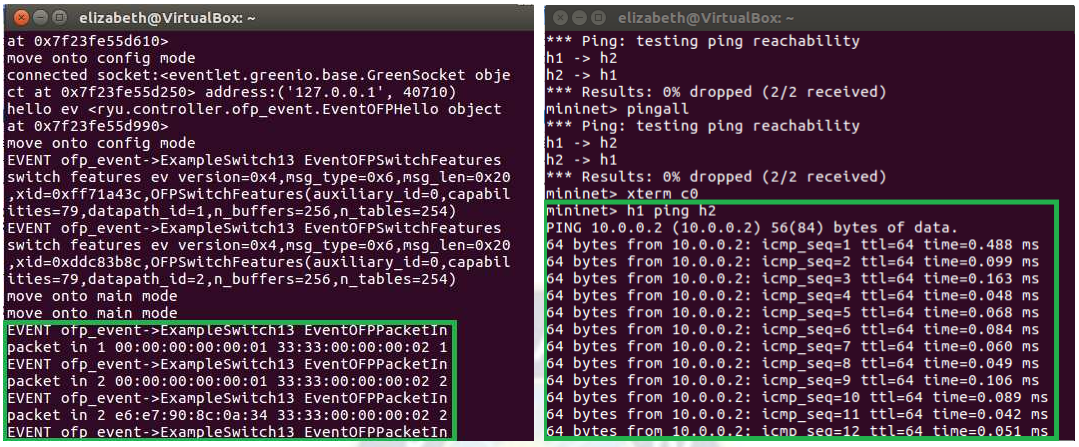


Figura 4.16: Mensaje en el controlador PACKET_OUT.
Fuente : Propia

Una vez que tanto Mininet como RYU están iniciados pasaremos a realizar las pruebas. Comenzaremos por realizar una prueba de conectividad entre los dos hosts de la

topología, se hará mediante el uso del comando ping, cuyo resultado se muestra en la figura 4.17.



(a) Intercambio de paquetes en el controlador (b) Ping h1 a h2 en el Shell Mininet

Figura 4.17: Test de conexión entre h1 y h2.
Fuente : Propia

Al ver que existe conectividad entre los hosts de la topología. Se comprueba que es posible enviar paquetes ICMP entre los hosts de la red emulada. Además es posible ver en el controlador una serie de paquetes de tipo OpenFlow.

Al comprobar la conectividad, vemos cuales son los paquetes que circulan por la red, a la vez que se lleva a cabo el comando ping, utilizaremos Wireshark para escuchar el tráfico de la interfaz lo, como se muestra en la figura 4.18. En el controlador aparecerán paquetes de tipo `ECHO`.

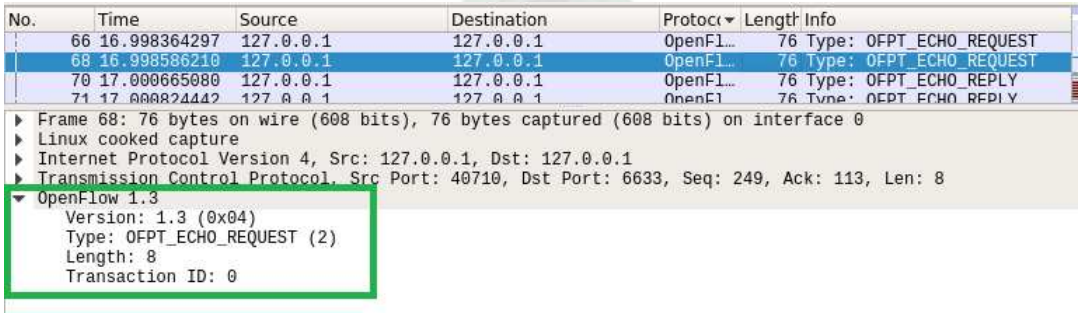


Figura 4.18: OFPT_ECHO_REQUEST.
Fuente : Propia

Los mensajes ECHO son mensajes de tipo petición/respuesta que pueden ser enviados bien desde el controlador o bien desde el switch, se enviará primero un paquete del tipo `of_echo_request` (petición) y este debe ser respondido con un mensaje `of_echo_reply`.

Además de en la interfaz anterior, también se ha capturado el tráfico existente entre los dos hosts (h1 y h2) de la topología, se muestra en la figura 4.19 .

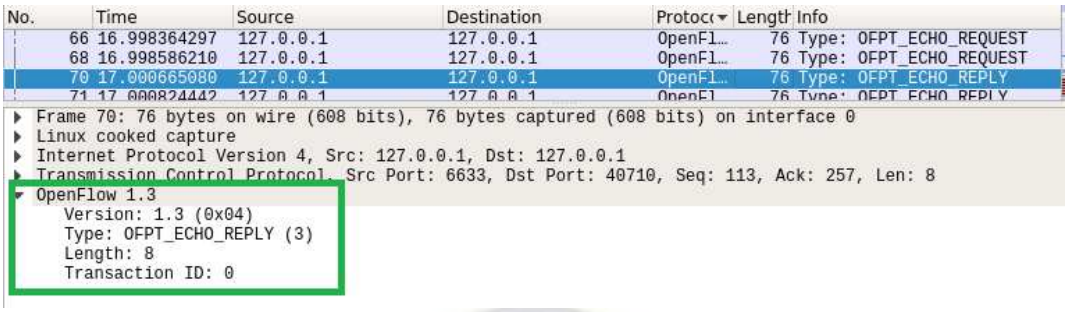


Figura 4.19: OFPT_ECHO_REPLY.
Fuente : Propia

Analizando la figura 4.19 mostrada anteriormente, se observa que además de los mensajes de petición respuesta, aparecen mensajes de tipo `PACKET_IN`. En estos mensajes se proporciona la información de las direcciones de origen y destino del ping encapsulada dentro del mensaje OpenFlow. La dirección IP de origen es la 10.0.0.1 y la de destino es la 10.0.0.2, correspondientes a los hosts h1 y h2 respectivamente.

4.2.2. Analisis de Resultados *Caso 2*

En este caso se vió la correcta interacción entre el controlador RYU y el protocolo OpenFlow, cabe resaltar que cada controlador pueden que tengan diferentes maneras de proceder con los mensajes Openflow.

Se mostró a detalle y paso a paso desde el inicio de la secuencia de los mensajes de OpenFlow a el controlador hasta las mostrar la conectividad del host, demostrando que existe coenctividad y un correcto funcionamiento.

4.3. *Caso 3:* Virtualizacion de la red IEA con el controlador RYU

Partimos de la red tradicional diseñada para la IEA (Instituto de Electrónica Aplicada). En la figura 4.20 vemos la topologia tradicional de la red IEA.

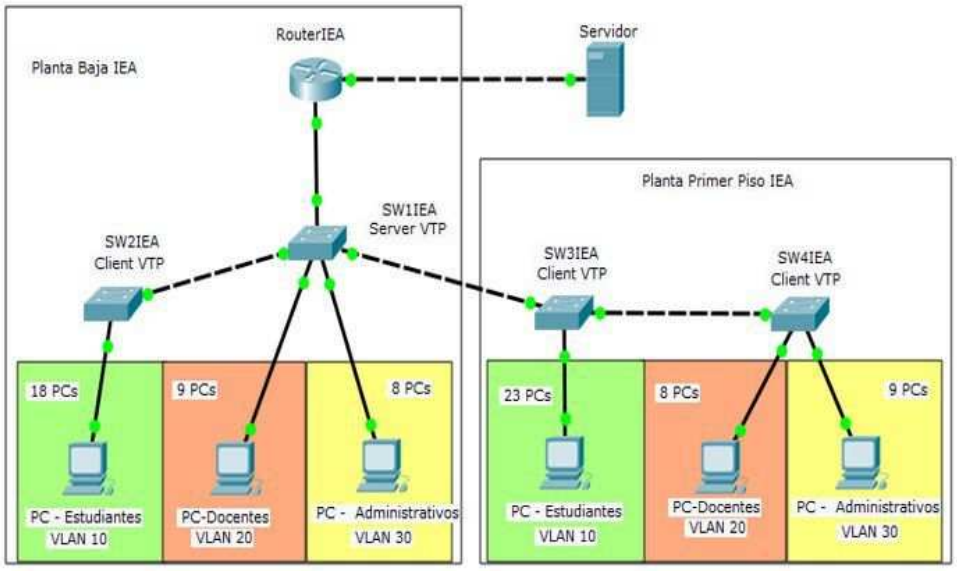


Figura 4.20: Topologia de la red IEA.
Fuente : Alfonso Jorge Gallegos Garcia

Tomando como referencia la red funcional para poder saber cuantos Switch Openflow colocar de manera que se asemeje a la red principal para ver la eficacia para la virtualizacion de red.

En general se formuló un codigó escrito en python que globaliza las diferentes configuraciones que se mostraran a continuación. Creamos una red en Mininet con el comando:

Comando 4.4 : Configuracion mininet.

```
1 sudo mn --topo linear,3,3 --mac --switch ovsk --controller remote -x
```

Una vez configurada la red en mininet con 9 host y 3 switches, se procede a configurar cada host, para una determinada IP, donde:

- Abriremos Xterm para cada host

```
mininet> xterm h1
mininet>
```

Figura 4.21: Xterm h1
Fuente : Propia

- Borrado la IP que tiene por defecto

Comando 4.5 : Borrado de la IP por defecto.

```
1 ip addr del 10.0.0.1/8 dev h1-eth0
```

- Añadiendo la ip designada para la creacion de Vlans

Comando 4.6 : Adición de la IP designada.

```
1 ip addr add 192.168.10.10/24 dev h1-eth0
```

Se debe configurar cada host individualmente como se vio anteriormente.

Se observó en la topología de la Red IEA la división de la red por VLAN, del inglés Virtual LAN (Red de área local y virtual), es un método que permite crear redes que lógicamente son independientes, aunque estas se encuentren dentro de una misma red física. De esta forma, un usuario podría disponer de varias VLANs dentro de un mismo router o switch.

Las VLANs comparte la red entre varios switches, routers y servidores. El protocolo en cargado de interconectar y compartir múltiples redes por la misma red física de las VLANs es 802.q de la IEEE proporcionando a su vez un mayor nivel de seguridad entre los segmentos de redes internas.

Con las VLANs se crean redes virtuales lo cual facilita la división de la red sin tener la necesidad de adquirir más equipos, para ese fin crearemos 3 VLANs. La configuración de la VLAN está codificada dentro de la aplicación antes de ejecutarse.

Una vez configurada la red en Mininet se procede de acuerdo al siguiente Diagrama de flujo donde mostramos los pasos para configuración del controlador.

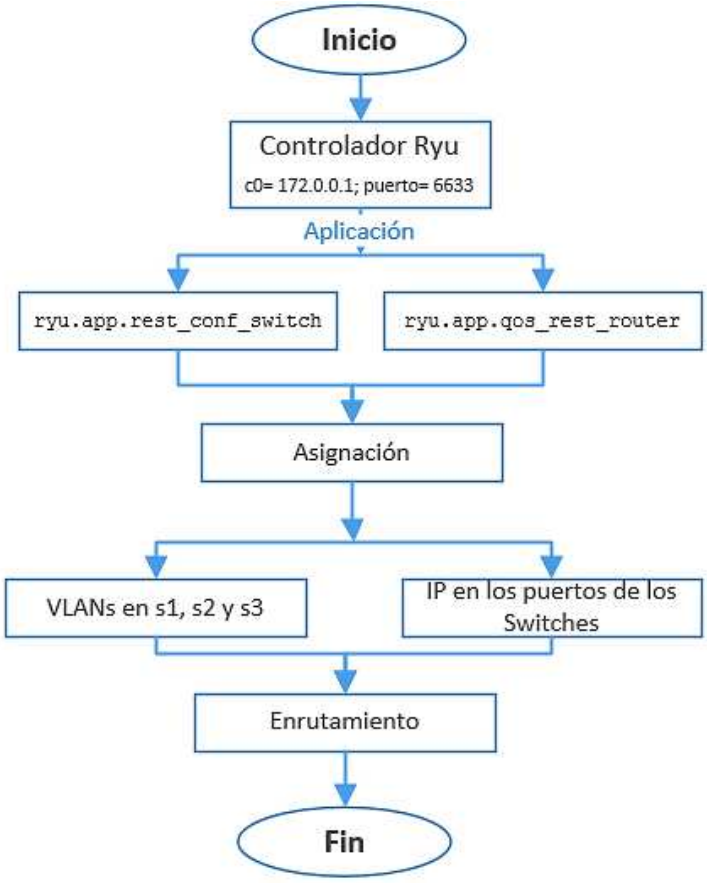


Figura 4.22: Diagrama de Configuración del Controlador Ryu.
Fuente : Propia

Ya conocido el proceso para la aplicación en el controlador procedemos con el siguiente comando con la que generaremos la red ya configurada:

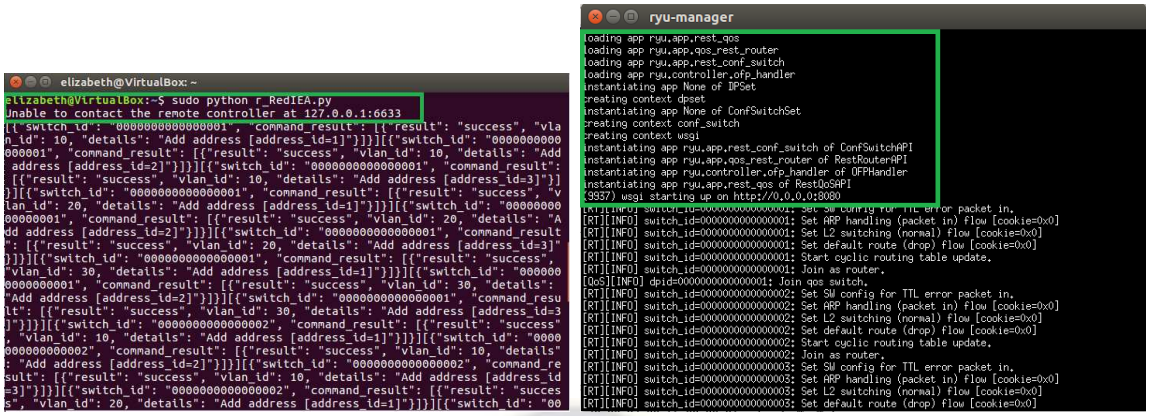
Comando 4.7 : Código de la Red IEA Virtualizada.

```
1 sudo python Route_RedIEA.py
```

El código del programa estará en el Anexo E.

Una vez ejecutado en la terminal el comando; se abrirá una ventana de xterm en donde se ejecuta Ryu-manager, a la vez Shell de Mininet, empezara el inicio del enrutamiento y paralelamente en el xterm del controlador se podrá ver las configuraciones del enrutamiento y designaciones de las Vlans en cada Switch.

A continuación procede al enrutamiento de cada uno de los switch.



(a) Ingreso del Comando e inicio de configuraciones

(b) Xterm del controlador y configuraciones del enrutamiento

Figura 4.23: Inicio de topología de la red IEA en Mininet.
Fuente : Propia

Donde exponemos el proceso para las aplicaciones que se implementaron en el codigo para la ejecución controlador Ryu:

- La primera aplicación es `rest_conf_switch.py` responsable de configurar todos los conmutadores para que actúen como conmutadores de capa 2, de manera similar a ela aplicación del 2do caso.

- La segunda aplicación es `qos_rest_router.py` es responsable del enrutamiento de todos los switches y guardar la configuración en formato JSON.

De acuerdo a la tabla de designación IP la topología de red será la siguiente:

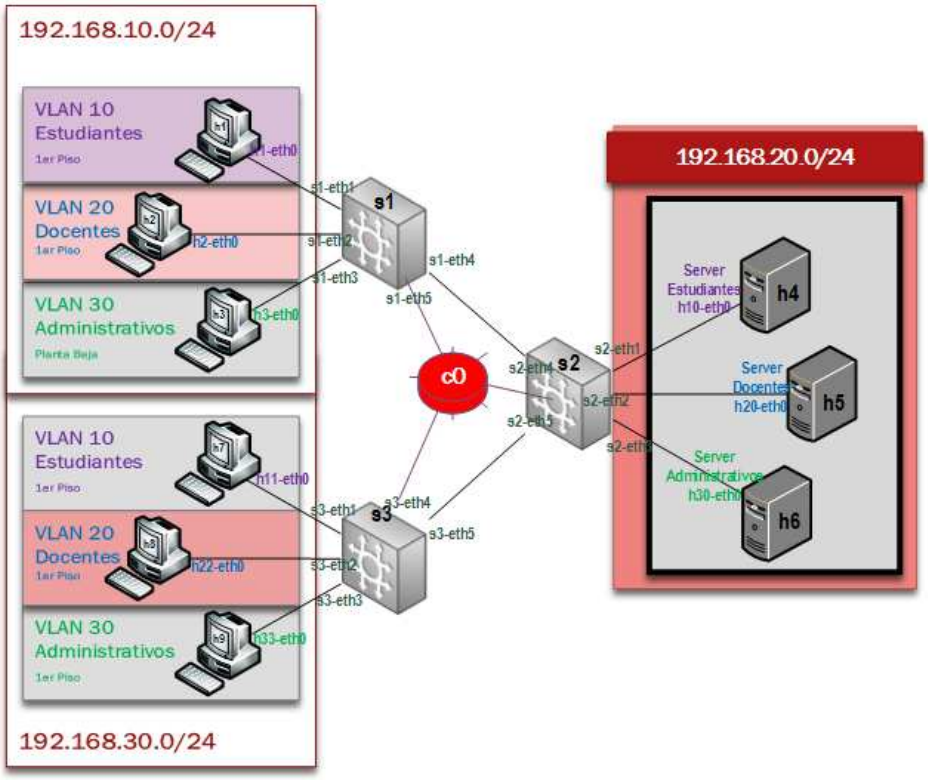


Figura 4.24: Topología de la red IEA Virtualizada.
Fuente : Propia

Para su configuración en Ryu se procedió a agregar el siguiente comando:

Comando 4.8 : Addicion de la red al controlador.

```
curl -X POST -d '{"address": "192.168.10.1/24"}' http://localhost
   :8080/router/0000000000000001/10
```

Al insertar el comando se creara un código JSON el cual muestra la siguiente información:

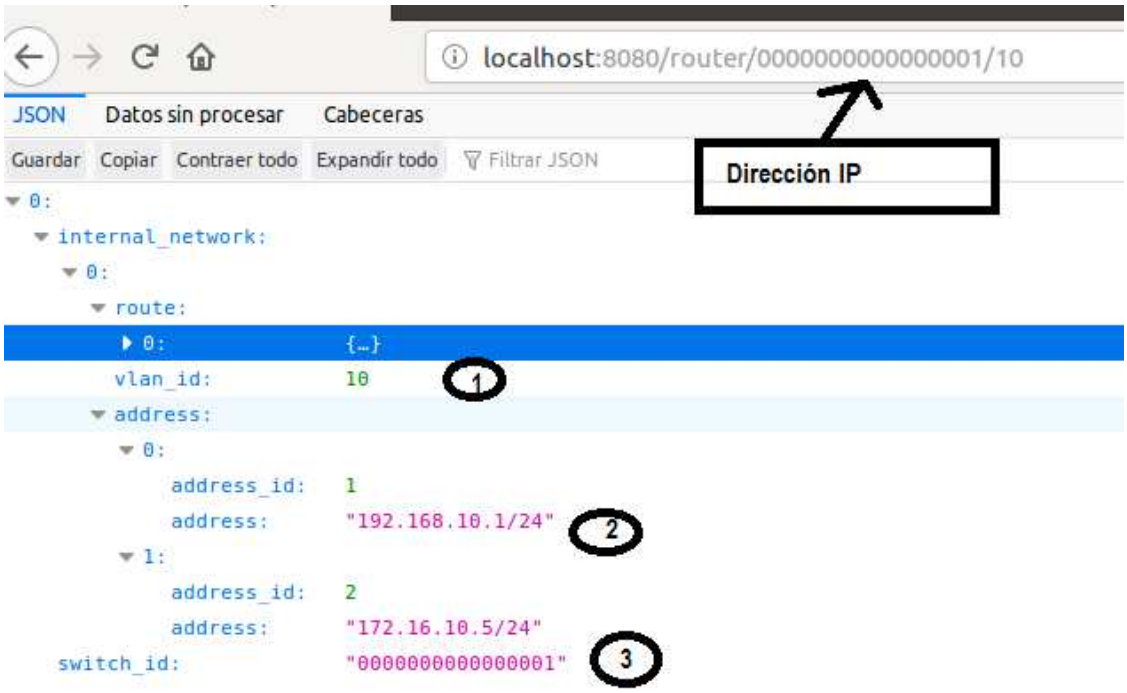


Figura 4.25: Configuracion de s1 en el controlador .
Fuente : Propia

La red se define con las diferentes VLANs escritas en la tabla a continuación:

	VLAN	RED	Host	Host Server
Estudiantes	10	192.168.10.10 192.168.30.70 192.168.20.40	h1 h7	 h4
Docentes	20	192.168.10.20 192.168.30.80 192.168.20.50	h2 h8	 h5
Administradores	30	192.168.10.30 192.168.20.60 192.168.30.90	h3 h9	 h6

Tabla 4.1: Tabla de designación de IP
Fuente : Propia

En primer lugar debemos hacer una tabla de enrutamiento:

	VLAN	IP Address	Defaulf Route	Static Route
s1	10, 20 y 30	192.168.10.1/24 172.16.10.5/24	172.16.10.10	
s2	10, 20 y 30	192.168.20.1/24 172.16.10.10/24 172.16.20.15/24	172.16.10.5	Destination: 192.168.30.0 Gateway: 172.16.20.20
s3	10, 20 y 30	192.168.30.1/24 172.16.20.20/24	172.16.20.15	

Tabla 4.2: Tabla de Enrutamiento Red IEA.
Fuente : Propia

Una vez establecidas las direcciones IP y la tabla de enrutamiento procedemos a la configuración de acuerdo a la figura 4.2, en Mininet y el controlador RYU, se propusó un programa escrito en Python el cual desplazo tanto la topologia de red como las aplicaciones del controlador RYU.

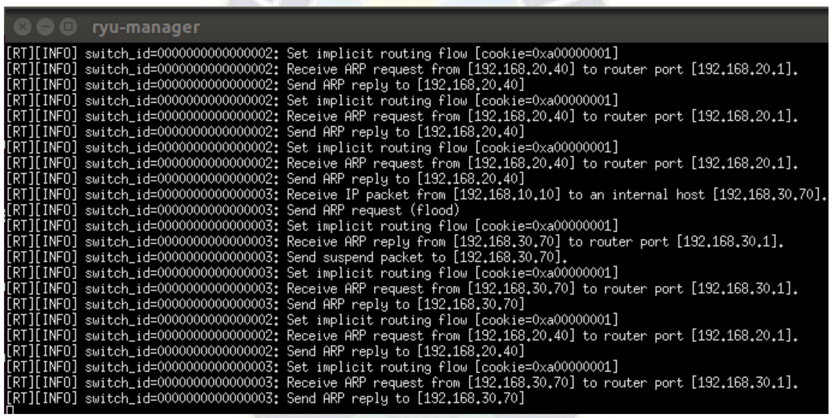
Para su configuración en RYU se procedió a agregar el siguiente comando:

Comando 4.9 : Proceso de enrutamiento.

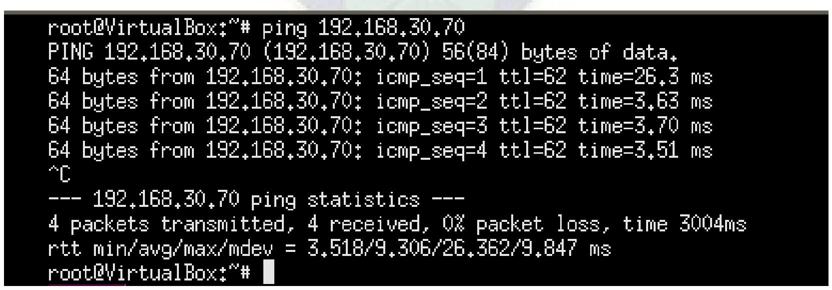
```
curl -X POST -d '{"destination": "192.168.30.0/24", "gateway":
   "172.16.20.20"}' http://localhost:8080/router/0000000000000002/2
```

Pruebas

Para hacer las pruebas se toma encuenta la conectividad haciendo ping, y támbien para ver las conexiones ARP de los switch que se vera en las figura 4.28.



(a) Proceso ARP en el controlador



(b) Ping entre h1 y h7

Figura 4.26: Llenado de tablas ARP para h1 y h7.
Fuente : Propia

Como existe comunicación ARP, la red virtual están configurado correctamente. De la figura 4.28 podemos concluir que no existe pérdida de paquetes ICMP por lo que la conectividad de la red está asegurada. Para ver la latencia, deberemos fijarnos en el parámetro rtt (Round-trip delay). Este parámetro proporciona el tiempo que tarda un

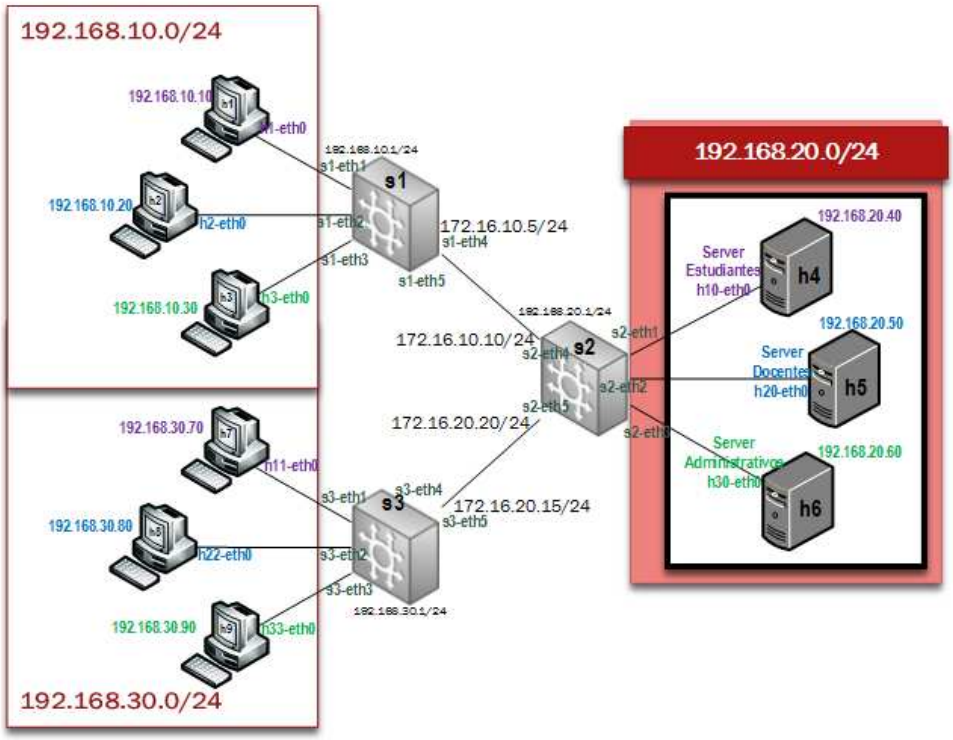


Figura 4.27: Configuracion de la red IEA Virtualizada.
Fuente : Propia

paquete de datos enviado desde el host h10 en volver a este host habiendo pasado por el host h1. En este caso vemos que el valor medio de rtt es 9.844 ms.

Para hacer las pruebas se tuvo transmisiones con el protocolo TCP durante un intervalo de 15 segundos. Estas transmisiones se utilizarán para medir el throughputh entre las diferentes VLAN:

- VLAN 10: h7 se ejecuta como servidor y h1 como cliente.
- VLAN 20: h8 se ejecuta como servidor y h5 como cliente.
- VLAN 30: h9 se ejecuta como servidor y h3 como cliente.

Los resultados obtenidos se muestran en la figura anterior.

Los resultados obtenidos se muestran en la figura 4.28.

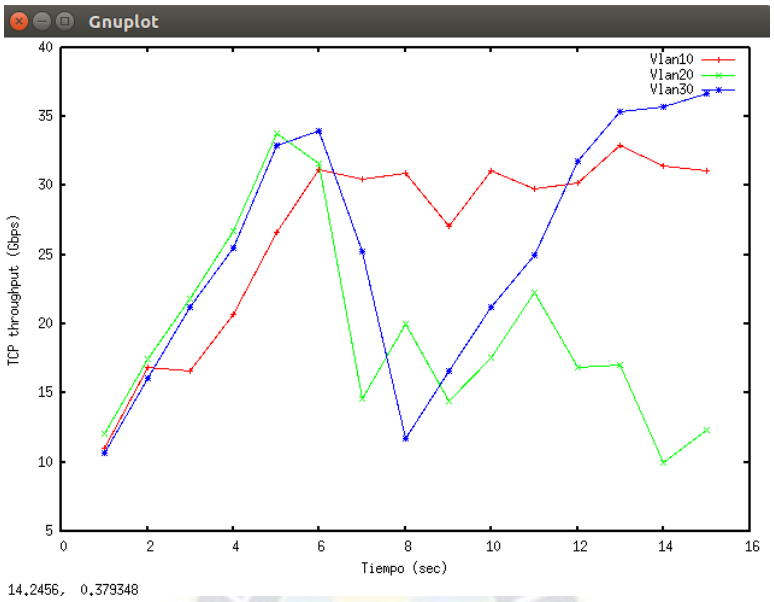


Figura 4.28: Grafico de TCP Trougthput VLAN 10, 20 y 30
Fuente : Propia

El grafico de TCP Trougthput de la figura 4.28 nos reflejó un correcto envió de datos y un buen funcionamiento de la transición de paquetes, el valor promedio de la red es 23.6 Gbps.

Realizamos a continuación una transmisión UDP, que servirá para medir la pérdida de paquetes. La duración de la transmisión es de 15 segundos. Los resultados se muestran en la Figura 4.29. Las transmisiones UDP son útiles para ver si existen pérdidas de paquetes en la transmisión. Para ello mandamos 10Mbps y nos fijamos en las estadísticas obtenidas en la trasmisión, que se muestran a continuación:

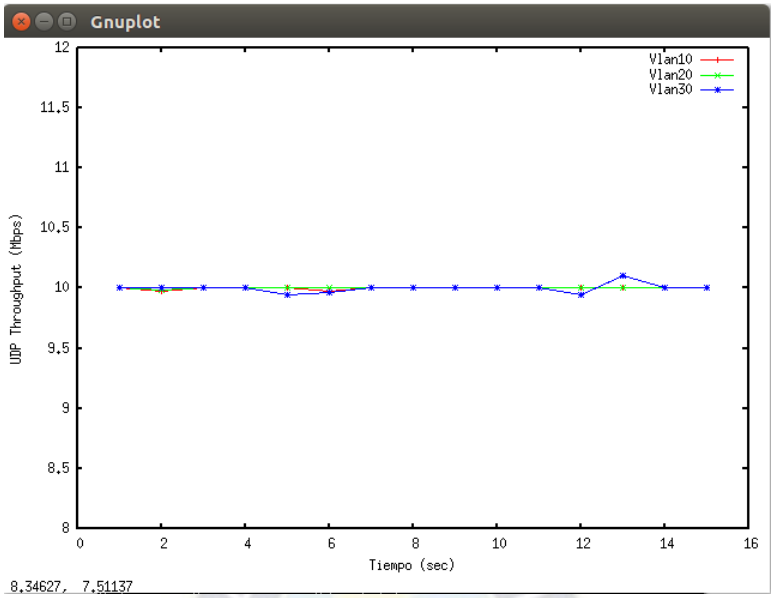


Figura 4.29: Grafico de UDP Trougthput VLAN 10, 20 y 30
Fuente : Propia

VLAN	Datagrama
10	Client connecting to 192.168.30.70, UDP port 5566 Sending 1470 byte datagrams UDP buffer size: 208 KByte (default) [32] local 192.168.10.10 port 39149 connected with 192.168.30.70 port 5566 [ID] Interval Transfer Bandwidth [32] 0.0-15.0 sec 17.9 MBytes 10.0 Mbits/sec [32] Sent 12756 datagrams [32] Server Report: [32] 0.0-15.0 sec 17.9 MBytes 10.0 Mbits/sec 0.043 ms 0/12755 (0%) [32] 0.0-15.0 sec 1 datagrams received out-of-order root@VirtualBox:~#
20	Client connecting to 192.168.30.80, UDP port 5566 Sending 1470 byte datagrams UDP buffer size: 208 KByte (default) [32] local 192.168.20.50 port 51006 connected with 192.168.30.80 port 5566 [ID] Interval Transfer Bandwidth [32] 0.0-15.0 sec 17.9 MBytes 10.0 Mbits/sec [32] Sent 12756 datagrams [32] Server Report: [32] 0.0-15.0 sec 17.9 MBytes 10.0 Mbits/sec 0.030 ms 0/12755 (0%) [32] 0.0-15.0 sec 1 datagrams received out-of-order root@VirtualBox:~#
30	Client connecting to 192.168.30.90, UDP port 5566 Sending 1470 byte datagrams UDP buffer size: 208 KByte (default) [32] local 192.168.10.30 port 39104 connected with 192.168.30.90 port 5566 [ID] Interval Transfer Bandwidth [32] 0.0-15.0 sec 17.9 MBytes 9.99 Mbits/sec [32] Sent 12748 datagrams [32] Server Report: [32] 0.0-15.0 sec 17.9 MBytes 9.99 Mbits/sec 0.032 ms 0/12747 (0%) [32] 0.0-15.0 sec 1 datagrams received out-of-order root@VirtualBox:~#

Tabla 4.3: Datagramas de VLANs
Fuente : Propia

En la tabla 4.3, se puede observar que las pérdidas de datagramas son los siguientes:

- **VLAN 10**= 0 %.
- **VLAN 20**= 0 %.

– **VLAN 30=** 0 %.

Teóricamente, no se debería tener una pérdida de paquetes mayor del 1 %. Como en este caso no existe perdida, por lo tanto se tiene una transmisión fiable.

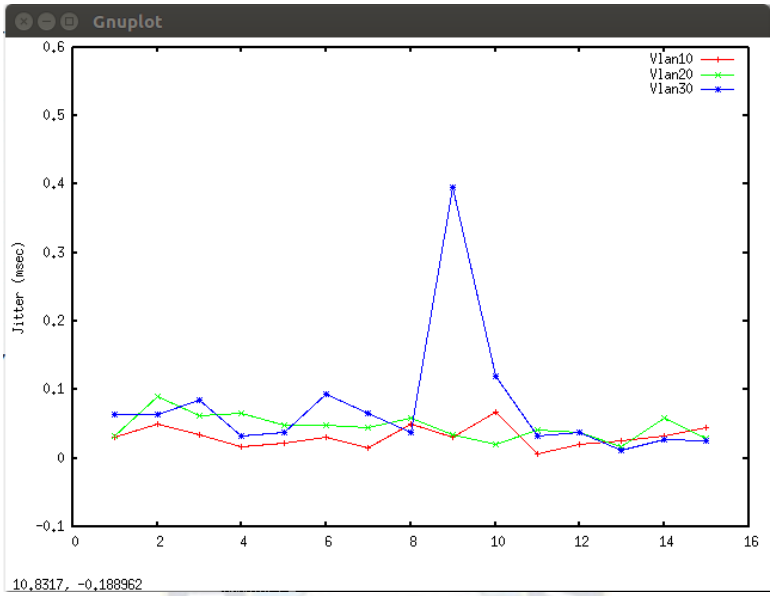


Figura 4.30: Grafico de Jitter VLAN 10, 20 y 30
Fuente : Propia

En la figura 4.30 se puede observar el jitter obtenido durante la transmisión. Este parámetro nos permite saber la calidad del enlace. Un jitter alto en el enlace significaría que el enlace es de mala calidad, puesto que existiría una variación en latencia importante, lo que provocaría retrasos en las transmisiones Un valor de jitter superior a 100 ms sería considerado alto. Si el valor es menor a esa cifra, podría compensarse de forma apropiado. En este caso el valor medio del jitter es de 0,035 ms, por lo que la calidad del enlace es buena.

4.3.1. Analisis de Resultados *Caso 3*

Pudimos implementar el uso del controlador enrutando una red ya creada que es la IEA, considerando que es notable el aumento de agilidad que se obtiene al separar el plano de control y el de datos. Anulando la tediosa tarea de generar configuraciones que son introducidas en cada switch para que la red funcione, cimplificando en gran medida a través del controlador que gestiona todo su entorno de forma global.

Guardando sus configuraciones en el código JSON para poder usarlos en otras redes, para poder facilitar su uso en diferentes controladores en la red, teniendo una red más versátil y flexible.

4.4. *Caso 4:* Balanceo de carga con QoS

Actualmente las redes en su mayoría hacen uso de la calidad de servicio o QoS, en consecuencia si las redes SDN abordaron el tema, para poder triunfar en la industria tienen que ofrecer al menos los mismos servicios y aplicaciones que las redes tradicionales.

Se muestran 2 incisos para desglosar las posibles aplicaciones de QoS en SDN con diferentes características:

a) Balance de carga a las diferentes Vlans

b) Qos con DiffServ

Se incorpora QoS en una red SDN con la finalidad de mostrar la facilidad de su implementación en las redes definidas por software, donde se pueden transferir los datos de una red en función de la prioridad basada en los tipos de datos y reservar ancho de banda de red para una comunicación particular con el fin de comunicarse con una constante ancho de banda de comunicación en la red.

4.4.1. a) Balance de carga a las diferentes Vlans

En esta sección daremos diferentes anchos de bandas a los diferentes usuarios, añadiendo configuraciones y reglas a las Queue para reservar el ancho de banda.

Definición de escenario

Utilizando la red ya diseñada de la IEA designaremos un diferente ancho de banda a los usuarios:

Usuario	IP	Puerto de destino	Protocolo	Queue ID	QoS ID	Ancho de Banda (kbps)
Estudiantes	192.168.20.40	5002	UDP	1	1	300
Docentes	192.168.20.50	5002	UDP	1	1	500
Administrativos	192.168.20.60	5002	UDP	1	1	700

Tabla 4.4: Designación de Ancho de Banda
Fuente : Propia

Configuracion de Queue

Lo primero debemos habilitar el puerto 6632 en s2 para acceder a OVSDB (Open vSwitch Database Management Protocol) en la terminal que ya está abierta para los comandos ovs:

Comando 4.10 : Configuración de puerto 6632.

```
sudo ovs-vsctl set Bridge s2 protocols=OpenFlow13
sudo ovs-vsctl set-manager ptcp:6632
```

Despues de iniciar las configuraciones para acceder a OVSDB con el comando 4.11:

Comando 4.11 : Localización de OVSDB.

```
curl -X PUT -d '"tcp:127.0.0.1:6632"' http://localhost:8080/v1.0/conf/
    switches/0000000000000001/ovsdb_addr
```

Permitirá después crear las colas, con comando 4.12 se define el ancho de banda de las diferentes colas con los campos `min_rate` y `max_rate` que permiten asegurar un mínimo ancho de banda y un máximo, respectivamente.

Comando 4.12 : Ancho de Banda.

```
curl -X POST -d '{"port_name": "s2-eth1", "type": "linux-htb", "
    max_rate": "300000", "queues": [{"max_rate": "350000"}, {"min_rate
    ": "200000"}]}' http://localhost:8080/qos/queue/0000000000000002
```

Con el comando POST anterior se definen las diferentes colas y sus anchos de banda. En el comando 4.12 hace referencia a un switch con un dpid de valor 0000000000000002, siendo este su identidad o id.

En el comando POST y su output se hace referencia a ese mismo id:

Comando 4.13 : Designación de IP y Queue para el Ancho de Banda.

```
curl -X POST -d '{"match": {"nw_dst": "192.168.20.40", "nw_proto": "
    UDP", "tp_dst": "5002"}, "actions":{"queue": "1"}}' http://
    localhost:8080/qos/rules/0000000000000002/10
```

Donde hace referencia tambien al IP y a Vlan que se designa, con el comando se generan las colas que se han creado en el switch con ese ID y Vlan. En este caso no hay problema porque solo existe un switch.

Para la red IEA tenemos diferentes Queue para cada switch:

VLAN 10		
Queue ID	Max rate (kbps)	Min rate (kbps)
0	300	
1	350	200

Tabla 4.5: Configuracion de Queue s2 VLAN 10.
Fuente : Propia

VLAN 20		
Queue ID	**Max rate (kbps)**	**Min rate (kbps)**
0	500	
1	550	400

Tabla 4.6: Configuracion de Queue s2 VLAN 20.
Fuente : Propia

VLAN 30		
Queue ID	**Max rate (kbps)**	**Min rate (kbps)**
0	700	
1	750	600

Tabla 4.7: Configuracion de Queue s2 VLAN 30.
Fuente : Propia

Pruebas

Para las pruebas utilizaremos el comando Iperf como:

- h1 y h7 como host, y h4 como servidor.
- h2 y h8 como host, y h5 como servidor.
- h3 y h9 como host, y h6 como servidor.

Utilizaremos el comando 4.14 para el servidor:

Comando 4.14 : Iperf para sevidor.

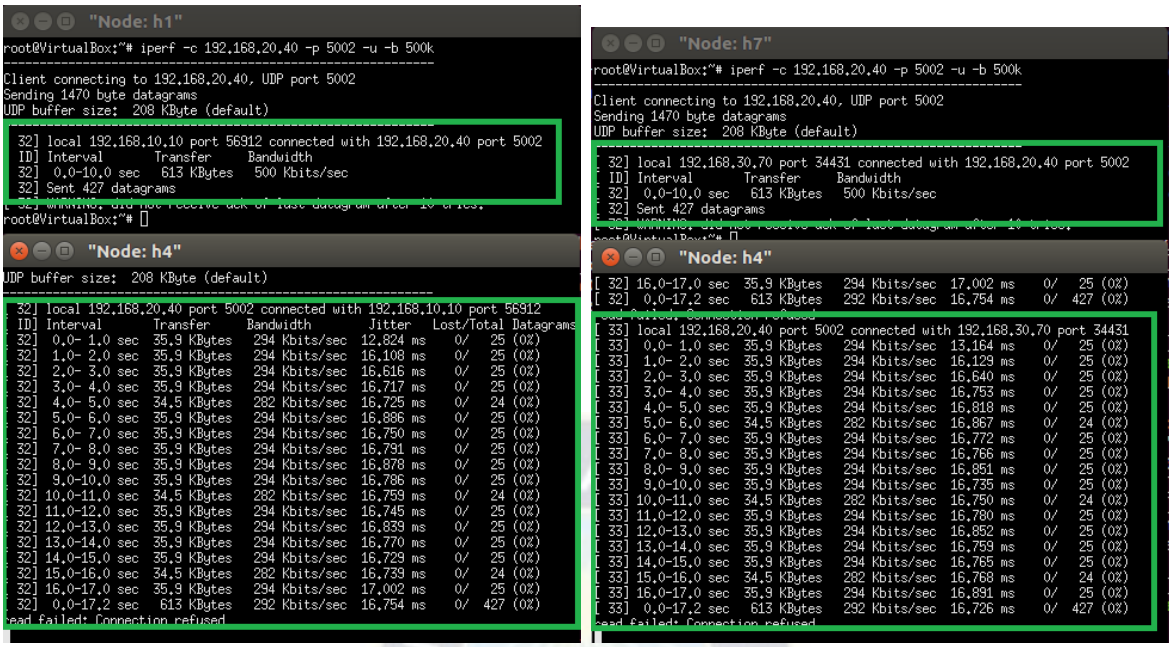
```
1 iperf -s -u -i 1 -p 5002
```

Utilizaremos el comando 4.15 para los host:

Comando 4.15 : Iperf para los host.

```
1 iperf -c 192.168.20.40 -p 5002 -u -b 500k
```

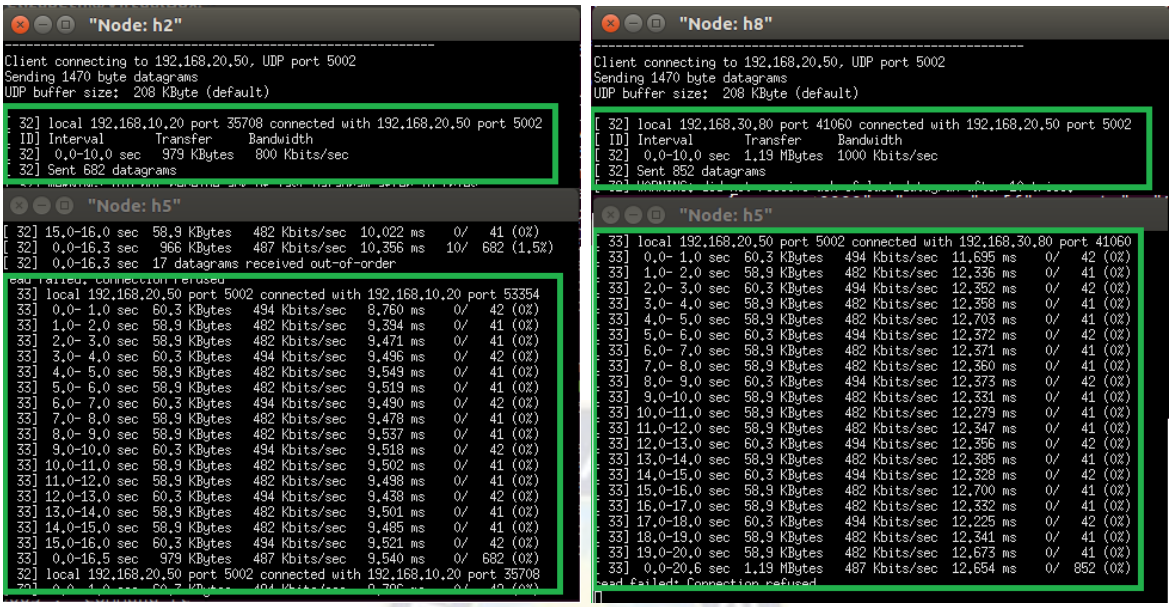
Creando un trafico UDP en cada servidor, se generan los siguientes resultados:



(a) Prueba UDP de hots h1 a servidor h4 (b) Prueba UDP de hots h7 a servidor h4

Figura 4.31: Ancho de banda Vlan 10.
Fuente : Propia

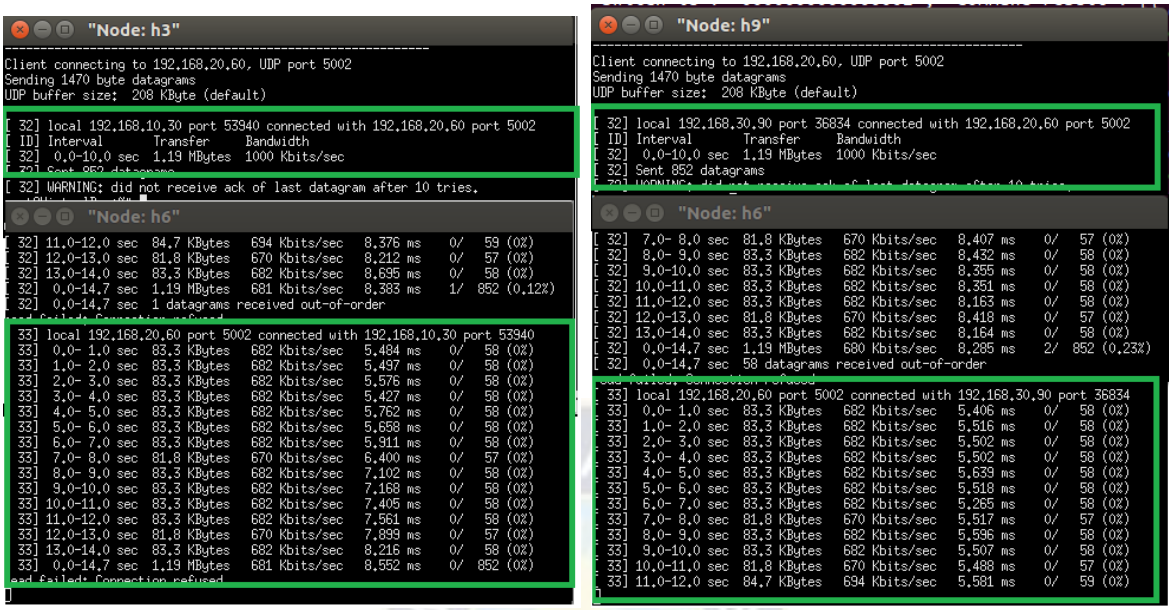
Se han transferido en total en esta transmisión 613kbps, como se observa en la figura 4.31, con un ancho de banda de aproximadamente 294kbps. Un resultado bastante preciso, teniendo en cuenta que al crear las colas se impuso en esta primera fase que la tasa máxima fuera `max_rate` 300kbps y sus bordes `min_rate` 200 kbps y `max_rate` 350 kbps, donde obserbamos que esta dentro de los margenes.



(a) Prueba UDP de hots h2 a servidor h5 (b) Prueba UDP de hots h8 a servidor h5

Figura 4.32: Ancho de banda Vlan 20.
Fuente : Propia

Se han transferido en total en esta transmisión 979kbps, como se observa en la figura 4.32, con un ancho de banda de aproximadamente 494kbps. Un resultado bastante preciso, teniendo en cuenta que al crear las colas se impuso en esta primera fase que la tasa máxima fuera `max_rate` 500kbps y sus bordes `min_rate` 400 kbps y `max_rate` 550 kbps, donde obserbamos que esta dentro de los margenes.



(a) Prueba UDP de hots h3 a servidor h6 (b) Prueba UDP de hots h9 a servidor h6

Figura 4.33: Ancho de banda Vlan 30.
Fuente : Propia

Se han transferido en total en esta transmisión 1.19Mbps, como se observa en la figura 4.33, con un ancho de banda de aproximadamente 682kbps. Un resultado bastante preciso, teniendo en cuenta que al crear las colas se impuso en esta primera fase que la tasa máxima fuera `max_rate` 700kbps y sus bordes `min_rate` 600 kbps y `max_rate` 750 kbps, donde obserbamos que esta dentro de los margenes.

4.4.2. b) Qos con DiffServ

Con el inciso a) se muestra QoS con un ancho de banda, donde se puede controlar el flujo finamente, pero a medida que aumentan los flujos de comunicación, también aumentan las entradas de flujo que se configuran para cada interruptor para controlar el ancho de banda, donde la QoS por flujo no es escalable.

Por lo tanto, el siguiente inciso divide los flujos en varias clases de QoS en el swicth de entrada del dominio DiffServ y aplica DiffServ para controlar los flujos para cada clase.

DiffServ reenvía los paquetes de acuerdo con el PHB (per-hop behavior) definido por el valor DSCP, que es el primer campo ToS de 6 bits en el encabezado IP, y realiza QoS.

A continuación, se muestra un ejemplo con la Vlan 30 de la red IEA virtualizada los cuales tendran configuración de cola y ancho de banda basada en la clase de QoS en Switch s2, y las reglas de instalación para marcar el valor DSCP de acuerdo con el flujo.

Definición de escenario

Utilizando la red ya diseñada de la IEA manejamos solamente la Vlan de Administracion dando atributos para garantizar su trafico:

IP	Puerto de destino	Protocolo	DSCP	Queue ID	QoS ID
192.168.10.30	5002	UDP	26(AF31)	1	1
192.168.10.30	5003	UDP	34(AF41)	1	2

Tabla 4.8: Entrada de flujo de acuerdo con el valor DSCP en el s2
Fuente : Propia

Configuracion de Queue

Utilizamos los comandos 4.10 y 4.11 que en el primer insciso modificando los parametros de los Queues:

Comando 4.16 : Ancho de Banda para QoS.

```
curl -X POST -d '{"port_name": "s2-eth3", "type": "linux-htb", "
    max_rate": "1000000", "queues":[{"max_rate": "1000000"}, {"min_rate
    ": "200000"}, {"min_rate": "500000"}]}' http://localhost:8080/qos/
    queue/0000000000000002/30
```

Configuracion de QoS

La configuracion se basara en la tabla 4.8 donde designamos valores DSCP en s2.

Comando 4.17 : Designación de DSCP 26(AF31) en Queue 1.

```
curl -X POST -d '{"match": {"ip_dscp": "26"}, "actions":{"queue":
    "1"}}' http://localhost:8080/qos/rules/000000000000002/30
```

Comando 4.18 : Designación de DSCP 34(AF41) en Queue 2.

```
curl -X POST -d '{"match": {"ip_dscp": "34"}, "actions":{"queue":
    "2"}}' http://localhost:8080/qos/rules/000000000000002/30
```

Asignaremos las QoS en los host conectados:

Comando 4.19 : Designación de DSCP 26(AF31)para h3.

```
curl -X POST -d '{"match": {"nw_dst": "192.168.30.10", "nw_proto": "
    UDP", "tp_dst": "5002"},"actions":{"mark": "26"}}' http://localhost
    :8080/qos/rules/0000000000000001
```

Comando 4.20 : Designación de DSCP 34(AF41) para h3.

```
curl -X POST -d '{"match": {"nw_dst": "192.168.30.10", "nw_proto": "
    UDP", "tp_dst": "5003"},"actions":{"mark": "34"}}' http://localhost
    :8080/qos/rules/0000000000000001
```

Donde hace referencia tambien al IP y a Vlan que se designa, con el comando se generan las colas que se han creado en el switch con ese ID y Vlan, añadiendo el QoS en cada Queue.

Para la Vlan 30 Administracion tenemos diferentes Queue para cada switch:

VLAN 30			
Queue ID	Max rate (kbps)	Min rate (kbps)	Clase
0	1Mbps		Defecto
1	1Mbps	200kbps	AF3
2	1Mbps	500kbps	AF4

Tabla 4.9: Configuracion de Queue s2 VLAN 30.
Fuente : Propia

Pruebas

Para las pruebas utilizaremos el comando Iperf como:

- h3 y h9 como host, y h6 como servidor.

Utilizaremos el comando 4.21 para el servidor, para el puerto 5002:

Comando 4.21 : Iperf para sevidor.

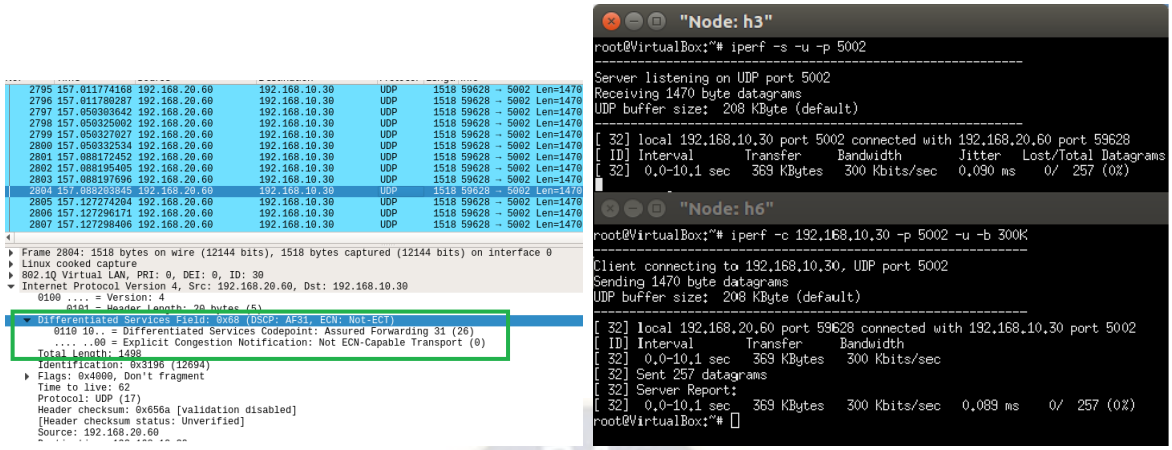
```
1 iperf -s -u -i 1 -p 5002
```

Utilizaremos el comando 4.22 para los host, para el puerto 5002:

Comando 4.22 : Iperf para los host.

```
1 iperf -c 192.168.10.30 -p 5002 -u -b 300K
```

Creando un trafico UDP en cada servidor, se generan los siguientes resultados:



(a) Captura de UDP en wireshark (b) Prueba UDP de hots h3 a servidor h6

Figura 4.34: Diffserv en s2 puerto 5002.
Fuente : Propia

En la figura 4.34 vemos la captura en wireshark donde captura trafico UDP que demuestra la clasificacion y marcado de paquetes gracias a las politicas QoS, en este caso AF31 remarcado con un cuadro verde.

Utilizaremos el comando 4.23 para el servidor, para el puerto 5003:

Comando 4.23 : Iperf para sevidor.

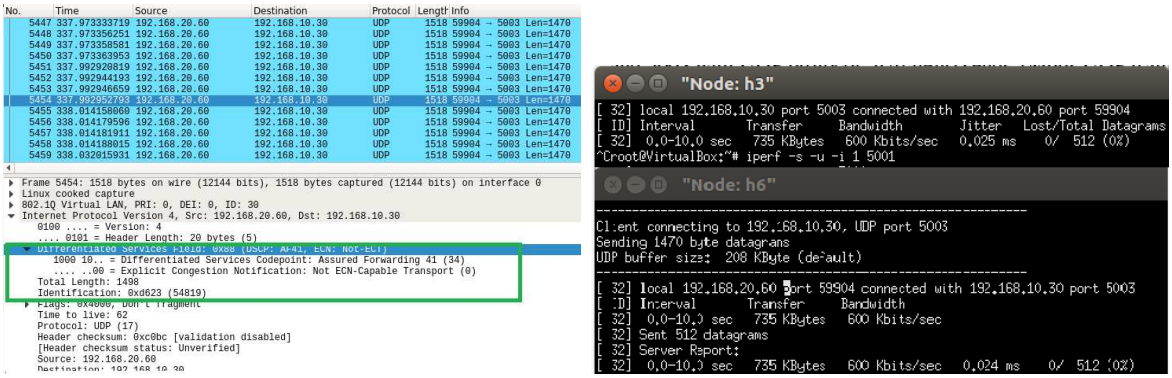
```
iperf -s -u -i 1 -p 5003
```

Utilizaremos el comando 4.24 para los host, para el puerto 5003:

Comando 4.24 : Iperf para el host.

```
iperf -c 192.168.10.30 -p 5003 -u -b 600K
```

Creando un trafico UDP en cada servidor, se generan los siguientes resultados:



(a) Captura de UDP en wireshark (b) Prueba UDP de hots h3 a servidor h6

Figura 4.35: Diffserv en s2 puerto 5003.
Fuente : Propia

En la figura 4.35 vemos la captura en wireshark donde captura trafico UDP que demuestra la clasificacion y marcado de paquetes gracias a las politicas QoS, en este caso AF41 remarcado con un cuadro verde.

4.4.3. Analisis de Resultados *Caso 4*

En este caso nos basamos en las aplicaciones que tiene el controlador RYU:

En el inciso *a)* se mostro que se puede hacer un balance de carga con respecto a las vlan dando prioridad a administrativos.

En el inciso *b)* se mostro que el tráfico marcado con AF41 (enviado al puerto 5003) tiene un ancho de banda garantizado de 500 Kbps, y el tráfico marcado con AF31 (enviado al puerto 5002) tiene un ancho de banda garantizado de 200 Kbps. Por otro lado, el ancho de banda del tráfico de mejor esfuerzo es limitado mientras se comunica el tráfico marcado con la clase AF. De esta manera, podemos confirmar que es posible realizar una QoS utilizando el modelo DiffServ.

5. Conclusiones

En SDN vemos una red que se puede programar, ya que se centraliza el plano de control, permitiendo el controlador maneje los flujos de paquetes, a través de ello se concluye que:

- SDN tiene una clara visión para formar una red de manera eficiente.
- El controlador es el único elemento centralizado y encargado de dar prioridades y/o funcionalidades a los elementos de red.
- El switch (elemento de red) solo es usado para la conmutación, dando paso a Openflow para el manejo eficiente de flujos.

Con el software de emulación Mininet herramienta orientada a demostrar de manera sencilla el entorno y como se maneja SDN.

Se mostraron diferentes casos:

- En el primer caso se vió una manera fácil de abordar SDN con Mininet.
- En el segundo caso se hizo funcional el controlador RYU demostrando los diferentes mensajes controlador a OpenFlow.
- En el tercer caso se vió el diseño de la virtulalizacion de la red IEA se hizo distintas pruebas de throughput tanto TCP como UDP y Jitter, y las tres fueron aceptables:
  - **TCP throughput** El ancho de banda promedio de la red fue 23.6 Gbps y ello nos ayudo a identificar que no existen problemas de conexión en las redes VLANs creadas.
  - **UDP throughput** No existió perdidas de datagramas lo cual hizo posible una transmision fiable.
  - **Jitter** El jitter promedio fue de 0.035ms por lo que la calidad de enlace es buena

Apesar que es una red relativamente pequeña se demostró que puede ahorar varios dispositivos dedicados tales como routers o switches de capa 3.

Para los diferentes objetivos:

En el primer objetivo específico, este proyecto ayudó a comprender cómo funciona SDN en la práctica. (Capitulo 2)

En el segundo objetivo específico, se virtualizo la red y se mostró la facilidad del manejo para el entrutamiento, donde se evita el problema de iniciar sesión en cada dispositivo (Red Tradicional) y las configuraciones se hace globalmente.(Capitulo 4 Seccion 4.3)

En el Tercer objetivo específico, se utilizó el controlador RYU creando diferentes escenarios:

- El primer escenario (Capitulo 4, seccion 4.2) se muestra la implementación de capa 2 (Swithing).
- El segundo escenario (Capitulo 4, seccion 4.3) se muestra la implementación de capa 3 (Routing).
- El tercer escenario (Capitulo 4, seccion 4.3) se muestra la aplicación de Balance de carga y de Servicios diferenciales.

En el posible uso del diseño en la red IEA se formularon las siguientes caracteristicas:

- Una red Programable y manejable deacuerdo a los requisitos.
- Una red Centrealizada ayudaria a los cuellos de botella con respecto a las inscripciones o a el uso de redes sociales.
- Un trafico organizado y restringido.

5.1. RECOMENDACIONES

A pesar de las ventajas anteriores obtenidas a través de SDN, las aplicaciones tiene ciertas limitaciones que deben abordarse:

- Si se crean redes con más de un controlador, se debe considerar STP (Spanning Tree Protocol) debido a los bucles que pueden ir generando.
- Si se considera desplegar la red propuesta en el presente proyecto es necesario considerar la versión de OpenFlow disponible y sus diferentes switch compatibles debido a las diferentes maneras en que despliega los flujos que manda Openflow en sus diferentes versiones.

Para implementar una red SDN se debe considerar un firewall activo o un sistema de seguridad fiable, debido a que el controlador que contiene la información completa de la red.

El tener varios controladores puede significar inconsistencias en las tablas de flujo o bucles en el desarrollo de la red. Más controladores necesariamente deberían tener una comunicación coordinada para manejar las tablas de flujos de OpenFlow.

Mininet al ser el emulador de red permite generar topologías SDN donde se pudo observar algunas inconveniencias al ejecutarse la red en un solo entorno, en este caso

es Ubuntu. Para asegurar el completo funcionamiento de la red se tuvo que limpiar las tablas de flujo creadas anteriormente esta se debe hacer repetidamente para evitar errores.

Es primordial saber que versión de OpenFlow trabaja y saber si el controlador soporta esta versión aún si existen varios controladores se deben verificar, en caso de RYU dispone para se aplicación hasta Openflow 1.4, debido a cada versión tiene diferentes tablas de flujo, o más funcionalidades.

Cuando se trabaja con proyectos OpenSource se considera que debe tener una comunidad que los respalde ya que hacen investigaciones sobre la fiabilidad del proyecto y poder confiar en los resultados de las aplicaciones.

. Acrónimos

SDN	Software Defined Network (Red definida por software).
ARP	Address Resolution Protocol.
MAC	Media Access Control address (Dirección física).
RYU	ŕee-yoohF̈lujo.
REST	Representational State Transfer (transferencia de estado representacional)
JSON	JavaScript Object Notation (Notación de objeto de JavaScript)
XML	Extensible Markup Language (Lenguaje de Marcado Extensible)
BW	BandWidth (Ancho de Banda).
API	Application Programming Interface (Interfaz de programación de aplicaciones).
QoS	Quality of Service (Calidad de Servicio).
IBM	International Business Machines ()
IETF	Internet Engineering Task Force (Grupo de Trabajo de Ingeniería de Internet)
ONF	Open Networking Foundation
IP	Internet Protocol (Protocolo de Internet).
TCP	Transmission Control Protocol (Protocolo de Control de Transmisión).
UDP	User Datagram Protocol (Protocolo de Datagramas de Usuario).
VLAN	Virtual Local Area Network (Red de Área Local Virtual).
ICMP	
MPLS	MultiProtocol Label Switching (Conmutación de Etiquetas MultiProtocolo).
Kbps	Kilobits per second (Kilobits por segundo).
Mbps	Megabits per second (Megabits por segundo).
TCAM	Ternary content-addressable memory
OSI	Open Systems Interconnect (Interconexión de sistemas abiertos)
RAM	Random Access Memory (Memoria de acceso aleatorio)
HTTP	Hypertext Transfer Protocol (Protocolo de transferencia de hipertexto)
OvS	Open vSwitch.
L2	Layer 2 (capa 2 del modelo OSI)
L3	Layer 3 (capa 3 del modelo OSI)

Bibliografia

Cross, G. (02 de Abril de 2017). *Open Networking Foundation.* Obtenido de https://www.opennetworking.org/

Giladi, Ran (2008). *Network Processors: Architecture, Programming, and Implementation.* Systems on Silicon. Morgan Kaufmann. ISBN 978-0-12-370891-5.

Gonzalez, C. A. (2014). *Despliegue de una maqueta de red basada en OpenFlow.* Proyecto de Grado, UNIVERSIDAD DE CANTABRIA, Facultad de Ciencias, Cantabria.

Goransson, P., Black, C., Culver, T. (2014). *Software Defined Networks.* Massachusetts: ELSEVIER.

IETF 7149: "Software-Defined Networking: A Perspective from within a Service Provider Environment". https://tools.ietf.org/html/rfc7149, marzo de 2014

Mininet. (15 de Febrero de 2019). Obtenido de *http://mininet.org/*

Open vSwitch FAQs. (23 de febrero de 2019). Obtenido de *https://github.com/openvswitch/ovs/blob/*

Palomo, R. D. (09 de Septiembre de 2018). Obtenido de *http://sdndesdecero.blogspot.com/p/que-es-sdn-la-separacion-fisica-del_14.html*

Ramires, A. V. (2015). *RED DEFINIDA POR SOFTWARE (SDN) EN BASE A UNA INFRAESTRUCTURA DE SOFTWARE DE LIBRE DISTRIBUCIN.* UNIVERSIDAD TCNICA DE AMBATO, Facultad de Ingenieria de Sistemas, Ambato.

Redeszone. (25 de 02 de 2019). Obtenido de *https://www.redeszone.net/2016/11/29/vlans-que-son-tipos-y-para-que-sirven/*

Rotsos, C. (2012). OFLOPS: *An Open Framework for Open. Springer-Verlag.*

Ryu SDN Framework. (16 de Febrero de 2019). Obtenido de *https://osrg.github.io/ryu/*

SDN. (15 de Agosto de 2017). Obtenido de *http://flowgrammable.org/sdn/* The Open Networking Foundation. (2012). OpenFlow Switch Specification. California.

What is OpenFlow? Definition and How it Relates to SDN. (06 de Agosto de 2018). Obtenido de *https://www.sdxcentral.com/networking/sdn/definitions/what-is-openflow/*

. Anexo A

Mininet

Para la visualización tanto de los Host, Switch y Controladores que tenemos distinguimos las siguientes formas de mininet:

`Hx` = Host
`Sx` = Switch
`Cx` = Controller

Para la interfaz de usuario y su configuración tendremos las siguientes:

`$` = Comando en Shell
`#` = Comandos como Root
`Mininet>` Comando dentro de mininet

Uso de la interfaz de línea de comandos de Mininet

Interacción con host y switches

- Inicio de la topología por defecto en mininet

```
$ sudo mn
```

- Inicio de la topología por defecto en mininet usando un controlador remoto

```
$ sudo mn -controller=remote,ip=[IP_ADDDR],port = [listening  port]
```

- Inicio de la topología tipo custom en mininet

```
$ sudo mn -custom [topo_script_path] − −topo = [topo_name]
```

- Muestreo de los nodos

```
mininet>nodes
```

- Muestreo de los enlaces

```
mininet>link
```

- Muestreo de la información de todos los nodos

```
mininet>dump
```

- Comprueba la dirección IP de un determinado nodo

```
mininet>h1 ifconfig -a
```

- Imprime la lista de procesos desde un proceso host

```
mininet>h1 ps -a
```

- Un enlace up/down en una red de mininet

```
mininet>link h1 s1 up
```

```
mininet>link h1 s1 down
```

Test de conectividad entre host

- Verificando la conectividad por pinging de host1 a host2

```
mininet>h1 ping -c 1 h2
```

- Verificando la conectividad en todos hosts

```
mininet>pingall
```

Ejecutar una prueba de regresión

- Tráfico recibir preparación

```
mininet>iperf -s -u -p [port_num] &
```

- Generación de tráfico desde el cliente.

```
mininet>iperf -c [IP] -u -t [duration] -b [bandwidth]
```

Variaciones de enlace

```
$ sudo mn -link tc,bw=[bandwidth] ,delay= [delay_in_millisecond]
```

Intérprete de Python

- Imprimir variables locales accesibles

```
$ py locals()
```

- Ejecutar un método invocando la API de

```
$ py [mininet_name_space] . [method]
```

- Ejecutar un método invocando la API de

```
$ py [mininet_name_space].[method]
```

. Anexo B

Instalaciones

Instalacion de Iperf

```
sudo apt-get install iperf
```

Instalacion de VLC

1. `sudo addaptrepository ppa:videolan/stabledaily`
2. `sudo aptget update`
3. `sudo aptget install vlc`

Instalacion de Wireshark

1. `sudo addaptrepositoru ppa:pirho/security`
2. `sudo aptget update`
3. `sudo aptget install wireshark`

. Anexo C

Laboratorio Práctico

Objetivo: El principal objetivo de esta práctica es empezar a relacionarse con las tecnologías SDN, y observar el intercambio de mensajes OpenFlow entre el controlador y los Switch virtuales.
Prerequisitos: Conocimiento de python y virtualización.
Procedimiento: Como parte de este módulo práctico, instalará el emulador de red Mininet en Ubuntu. Luego, configurará una red OpenFlow independiente usando Mininet. A continuación, instalará el controlador Ryu en su Ubuntu. Finalmente, configurará la red Mininet para usar un controlador Ryu.

Notas de laboratorio

Para emular una red OpenFlow, usaremos una máquina virtual:

- Ubuntu (versiones arriba de 14.04)

Como mínimo, recomendamos las siguientes especificaciones de hardware:

- Procesador Intel i3, i5 o i7
- 6GB de RAM
- 25 GB de espacio libre en el disco duro

Configuración OpenFlow y Controlador Ryu

Pasos básicos:

1. Instalar Oracle VirtualBox.
2. Instalar Ubuntu.
3. Instalar WireShark en Ubuntu.
4. Instalar Mininet en Ubuntu.
5. Pruebe una red Mininet simple e independiente.

6. Instalar y configurar Ryu en Ubuntu.
7. Pruebe una red simple de Mininet usando el controlador Ryu.
8. Inicie Ryu manager para trabajar con Mininet
9. Cree un flujo desde el administrador de OpenFlow para controlar el tráfico

Parte 1. **Virtual Box es compatible con:**

- Windows
- OS X
- Pasos de instalación de Linux:

1. Descargue el paquete de instalación apropiado para su sistema operativo desde: *https: //www.virtualbox.org/wiki/Downloads*
2. Utilice el instalador suministrado para instalar utilizando los parámetros predeterminados.

Parte 2. **Instalación de Ubuntu.**

1. Descargue el VM de Ubuntu e instale en Virtual desde la pagina web: *http://releases.ubuntu.com/*
2. Utilice el instalador suministrado para instalar utilizando los parámetros predeterminados.

Parte 3. **Instalar WireShark en Ubuntu**

Instalación:

- En la terminal agregar el siguiente comando:

```
sudo apt-get install wireshark
```

Parte 4. **Instalar Mininet Ubuntu.**

Instalación y configuración:

Para ahorrar tiempo, su instructor le proporcionará instrucciones para descargar mininet y las configuraciones o también véase el ANEXO A.

- Pruebe su correcta funcionalidad de mininet con el comando:

```
sudo mn -test pingall
```

Parte 5. **Construye una red independiente de Mininet**

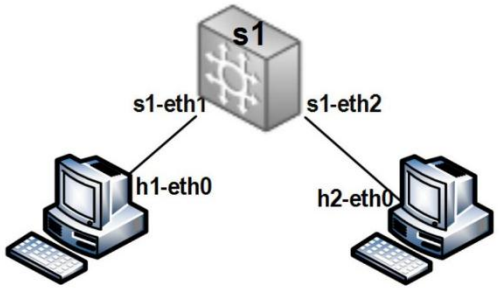
Pasos de instalación y configuración:

1. Cree una red simple en Ubuntu desde la terminal:

```
sudo mn -mac -controller="none"
```

El comando anterior crea una red con:

- Switch1, s1.
- Hosts 2 hosts, h1 y h2.
- h1-eth0 conectado a s1 eth0.
- h2-eth0 conectado a s1 eth1.
- No hay controlador OpenFlow.



El comando anterior te pondrá en el shell de Mininet:

```
mininet>
```

2. Pruebe los siguientes comandos en la CLI de Mininet para comprender la red:

```
mininet>nodos
mininet>net
mininet>dump
```

3. Intente hacer pings entre el host:

```
mininet>h1 ping h2
mininet>h2 ping h1
```

¿Los pings tienen éxito? ¿Por qué?

4. Compruebe la tabla de flujo del interruptor:

```
mininet>dpctl dump-flows
```

¿Puedes explicar la razón de lo que ves?

5. Salir del Mininet CLI:

```
mininet>exit
```

Parte 6. **Construye una red en Mininet con el controlador**

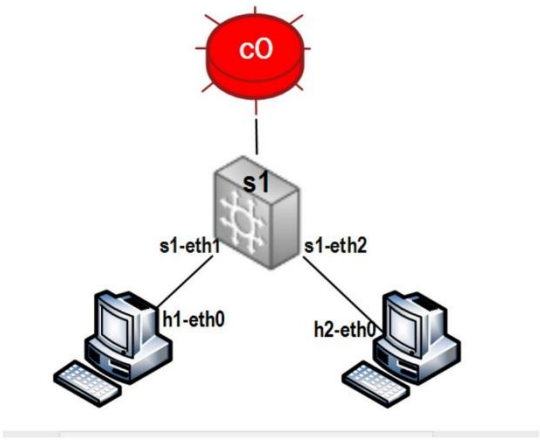
Pasos de instalación y configuración:

1. 6. Cree otra red simple en Ubuntu desde la terminal, pero esta vez use el controlador ryu.

```
sudo mn -mac -controller=x
```

El comando anterior crea una red con:

- Switch1, s1.
- Hosts 2 hosts, h1 y h2.
- h1-eth0 conectado a s1 eth0.
- h2-eth0 conectado a s1 eth1.
- No hay controlador OpenFlow.



El comando anterior te pondrá en el shell de Mininet:

```
mininet>
```

2. Pruebe los siguientes comandos en la CLI de Mininet para comprender la red:

```
mininet>nodos
mininet>net
mininet>dump
```

3. Abra otra terminal o Xterm y añada el siguiente comando:

```
ryu-manager -verbose ryu.app.simple_switch_13
```

4. Intente hacer pings entre el host:

```
mininet>h1 ping h2
mininet>h2 ping h1
```

¿Los pings tienen éxito? ¿Por qué?

5. Verifique nuevamente la tabla de flujo del interruptor:

```
mininet>dpctl dump-flows
```

¿Qué puedes ver ahora? Analizar las entradas en la tabla.

6. Salir del Mininet CLI:

```
mininet>exit
```

. Anexo D

Codigo en python para el Switching en RYU

```
from ryu.base import app_manager  # iniciar de la App manager de ryu

from ryu.controller import ofp_even # Para recibir eventos openfow
#Ryu implemeta el controlador de eventos respect al mensaje recibido como #
                                          CONFIG_DISPATCHER recibe las configuraciones
                                           del switch y MAIN_DISPATCHER manda se al
                                          de #normalidad en su estado.
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
mensaje recibido y el estado del conmutador OpenFlow
from ryu.ofproto import ofproto_v1_3  # Indica el modulo ofproto que se utiliza en este
#caso OpenFlow1.3


# RYU utiliza ryu.lib.packet para codificar o descodificar un paquete
from ryu.lib.packet import packet #
from ryu.lib.packet import ethernet  # para la entrada de encabezado ethernet


class SimpleSwitch13(app_manager.RyuApp): # Inicializacion de una aplicacion RYU
OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION] # Especifica la version de OpenFlow a #utilizar

def __init__(self, *args, **kwargs):
super(SimpleSwitch13, self).__init__(*args, **kwargs)
#inicializa la tabla de direcciones MAC
self.mac_to_port = {}
#luego de inicializar con las tablas MAC se inicializa con la creacion Table-miss
@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)

def add_flow(self, datapath, priority, match, actions):
ofproto = datapath.ofproto
parser = datapath.ofproto_parser
#contruccion de mensaje y enviarlo
inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,actions)]
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):
msg = ev.msg
datapath = msg.datapath
ofproto = datapath.ofproto
parser = datapath.ofproto_parser
```

```
#obtencion de datapath ID para identificar el switch openflow
dpid = datapath.id
self.mac_to_port.setdefault(dpid, {})

#analizando los paquetes recibidos a trav s de ryu lib
pkt = packet.Packet(msg.data)
eth = pkt.get_protocols(ethernet.ethernet)[0]
dst = eth.dst
src = eth.src

#recibe el numero de Puerto de mensaje de packet_in
in_port = msg.match[  in_port  ]
self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)

# Aprende la direccion mac para evitar una inundaciod la pr xima vez.
self.mac_to_port[dpid][src] = in_port

# Crea la condici n que si direccion MAC de destino esta aprendida decide a cual puerto
                                        #de salida mandar en otro caso realizar una
                                        inundacion
if dst in self.mac_to_port[dpid]:
out_port = self.mac_to_port[dpid][dst]
else:
out_port = ofproto.OFPP_FLOOD

#Construccion de la lista de Action
actions = [parser.OFPActionOutput(out_port)]

# instalacion de flujos para evitar Packet_in la preoxima vez.
if out_port != ofproto.OFPP_FLOOD:
match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
self.add_flow(datapath, 1, match, actions)

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 E

Codigo en python para la IEA

```
#!/usr/bin/python
from mininet.net import Mininet
from mininet.cli import CLI
from mininet.topo import Topo
from mininet.node import OVSSwitch
from mininet.node import RemoteController
import os, sys
import re
import subprocess
import pexpect
from sets import Set
from time import sleep

vlanIDs = [10,20,30]

class MyTopo(Topo):
    def __init__(self):
        "Create custom topo."
        # Iniciando la topologia
        Topo.__init__(self)

        # Anadiendo host
        hosts = [self.addHost('h%d' % n) for n in range(1, 10, 1)]
        # Anadiendo switch
        switches = [self.addSwitch('s%d' % n) for n in range(1, 4, 1)]

        # Enlazando Switches con Host
        self.addLink(hosts[0], switches[0])
        self.addLink(hosts[1], switches[0])
        self.addLink(hosts[2], switches[0])
        self.addLink(hosts[3], switches[1])
        self.addLink(hosts[4], switches[1])
        self.addLink(hosts[5], switches[1])
        self.addLink(hosts[6], switches[2])
        self.addLink(hosts[7], switches[2])
        self.addLink(hosts[8], switches[2])

    # Enlazando los switches
        self.addLink(switches[0], switches[1])
        self.addLink(switches[1], switches[2])

      # Ingresando a cada Host de la red
    def setHosts(net):
       for host in net.hosts:
         intfName=host.intfs[0].name
         lastPartOfiPaddr = host.intfs[0].ip.split('.')[3]

        # Borrando el ip por defecto designado por mininet
        host.cmdPrint('ip addr del 10.0.0.' + lastPartOfiPaddr + '/8 dev ' + intfName)

        # Designando direcciones IP para el enrrutamiento
        if lastPartOfiPaddr in ['1','2','3']:
```

```
            virtIpAddr = '192.168.10.' + lastPartOfiPaddr + '0'
            virtGw='192.168.10.1'
            virtRemoteSubnets=['192.168.20.0/24','192.168.30.0/24']

        elif lastPartOfiPaddr in ['4','5','6']:
            virtIpAddr = '192.168.20.' + lastPartOfiPaddr + '0'
            virtGw='192.168.20.1'
            virtRemoteSubnets=['192.168.10.0/24','192.168.30.0/24']

        elif lastPartOfiPaddr in ['7','8','9']:
            virtIpAddr = '192.168.30.' + lastPartOfiPaddr + '0'
            virtGw='192.168.30.1'
            virtRemoteSubnets=['192.168.10.0/24','192.168.20.0/24']

        if lastPartOfiPaddr in ['1','4','7']:
            virtIntfName = intfName + '.10'
            host.cmdPrint('ip link add link ' + intfName + ' name ' + virtIntfName + '
                                                    type vlan id 10')
            host.cmdPrint('ip addr add ' + virtIpAddr + '/24 dev ' + virtIntfName)
            host.cmdPrint('ip link set dev ' + virtIntfName + ' up')

        elif lastPartOfiPaddr in ['2','5','8']:
            virtIntfName = intfName + '.20'
            host.cmdPrint('ip link add link ' + intfName + ' name ' + virtIntfName + '
                                                    type vlan id 20')
            host.cmdPrint('ip addr add ' + virtIpAddr + '/24 dev ' + virtIntfName)
            host.cmdPrint('ip link set dev ' + virtIntfName + ' up')

        elif lastPartOfiPaddr in ['3','6','9']:
            virtIntfName = intfName + '.30'
            host.cmdPrint('ip link add link ' + intfName + ' name ' + virtIntfName + '
                                                    type vlan id 30')
            host.cmdPrint('ip addr add ' + virtIpAddr + '/24 dev ' + virtIntfName)
            host.cmdPrint('ip link set dev ' + virtIntfName + ' up')

        for virtSubnet in virtRemoteSubnets:
            host.cmdPrint('ip ro add ' + virtSubnet + ' via ' + virtGw + ' dev ' +
                                                    virtIntfName)


#Anadiendo el gateway a cada host
def routeAdd(net):
    for host in net.hosts:
        intfName=host.intfs[0].name
        lastPartOfiPaddr = host.intfs[0].ip.split('.')[3]

        if lastPartOfiPaddr in ['1','2','3']:
            virtGw='192.168.10.1'
            host.cmdPrint('ip route add default '+ virtGw)
        elif lastPartOfiPaddr in ['4','5','6']:
            virtGw='192.168.20.1'
            host.cmdPrint('ip route add default '+ virtGw)
        elif lastPartOfiPaddr in ['7','8','9']:
            virtGw='192.168.30.1'
            host.cmdPrint('ip route add default '+ virtGw)

#DESDE EL CONTROLADOR
#Anadiendo las rede IP para el erutamiento
def setController(net):
    #set the addresses for routers
    #s1
    #vlan 10,20,30
    for vlan in vlanIDs:
        subprocess.call('curl -X POST -d \'{\"address\":\"192.168.10.1/24\"}\' http://
                                                localhost:8080/router/
                                                0000000000000001/'+str(vlan),shell=
                                                True)
        subprocess.call('curl -X POST -d \'{\"address\":\"172.16.10.5/24\"}\' http://
```

```
                                                    localhost:8080/router/
                                                    0000000000000001/'+str(vlan),shell=
                                                    True)

    #s2
    # #vlan 10,20,30
    for vlan in vlanIDs:
        subprocess.call('curl -X POST -d \'{\"address\":\"192.168.20.1/24\"}\' http://
                                                    localhost:8080/router/
                                                    0000000000000002/'+str(vlan),shell=
                                                    True)
        subprocess.call('curl -X POST -d \'{\"address\":\"172.16.10.10/24\"}\' http://
                                                    localhost:8080/router/
                                                    0000000000000002/'+str(vlan),shell=
                                                    True)
        subprocess.call('curl -X POST -d \'{\"address\":\"172.16.20.15/24\"}\' http://
                                                    localhost:8080/router/
                                                    0000000000000002/'+str(vlan),shell=
                                                    True)

    #s3
    #vlan 10,20,30
    for vlan in vlanIDs:
        subprocess.call('curl -X POST -d \'{\"address\":\"192.168.30.1/24\"}\' http://
                                                    localhost:8080/router/
                                                    0000000000000003/'+str(vlan),shell=
                                                    True)
        subprocess.call('curl -X POST -d \'{\"address\":\"172.16.20.20/24\"}\' http://
                                                    localhost:8080/router/
                                                    0000000000000003/'+str(vlan),shell=
                                                    True)

#Designando las direcciones IP para el  enrutamiento
    #Routing:
    #s1
    for vlan in vlanIDs:
        subprocess.call('curl -X POST -d \'{\"gateway\": \"172.16.10.10\"}\' http://
                                                    localhost:8080/router/
                                                    0000000000000001/'+str(vlan),shell=
                                                    True)

    #s2
    for vlan in vlanIDs:
        subprocess.call('curl -X POST -d \'{\"gateway\": \"172.16.10.5\"}\' http://
                                                    localhost:8080/router/
                                                    0000000000000002/'+str(vlan),shell=
                                                    True)
        subprocess.call('curl -X POST -d \'{\"destination\":\"192.168.30.0/24\",\"gateway
                                                    \": \"172.16.20.20\"}\' http://
                                                    localhost:8080/router/
                                                    0000000000000002/'+str(vlan),shell=
                                                    True)

    #s3
    for vlan in vlanIDs:
        subprocess.call('curl -X POST -d \'{\"gateway\": \"172.16.20.15\"}\' http://
                                                    localhost:8080/router/
                                                    0000000000000003/'+str(vlan),shell=
                                                    True)


#Iniciando las Aplicaciones del controlador RYU
def startController():
    ctrlr = pexpect.spawn(
       'xterm -geometry 100x24+0+0 -e ryu-manager ryu.app.rest_qos ryu.app.
                                                   qos_rest_router ryu.app.
                                                   rest_conf_switch')
    return ctrlr
```

```
def genericTest(topo):
    ctrlr=startController()
    sleep(1)
    #ctrlr1=wireshark()
    sleep(1)
    c0=RemoteController( 'c0', ip='127.0.0.1', port=6633 )
    net = Mininet(topo=topo, switch=OVSSwitch,
                  controller=c0)
    net.start()

    setHosts(net)
    #routeAdd(net)
    sleep(1)
    setController(net)
    CLI(net)

    net.stop()
    ctrlr.sendcontrol('c')
    ctrlr.close()

def main():
    topo = MyTopo()
    genericTest(topo)

if __name__ == '__main__':
    main()
```