DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERÍA ELECTRÓNICA, REDES DE LA INFORMACIÓN Y COMUNICACIÓN DE DATOS

TESIS PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA

TEMA: DISEÑO E IMPLEMENTACIÓN DEL PROTOTIPO DE UNA RED DEFINIDA POR SOFTWARE (SDN) EN LA UNIVERSIDAD DE LAS FUERZAS ARMADAS "ESPE"

AUTORES: ALBÁN RUIZ, PABLO FERNANDO

BRITO LÓPEZ, DARÍO XAVIER

DIRECTOR: ING. GORDILLO, RODOLFO

CODIRECTOR: ING. ROMERO, CARLOS

SANGOLQUÍ, 2015

CERTIFICACIÓN

Certificamos que el proyecto titulado “DISEÑO E IMPLEMENTACIÓN DEL PROTOTIPO DE UNA RED DEFINIDA POR SOFTWARE (SDN) EN LA UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE”, ha sido guiado y revisado periódicamente y cumple normas estatutarias establecidas por la ESPE, en el Reglamento de Estudiantes de la Universidad de las Fuerzas Armadas.

Debido a que se trata de un trabajo de investigación se recomienda su publicación.

El mencionado trabajo consta de un documento empastado y un disco compacto el cual contiene los archivos en formato portátil de Acrobat (pdf). Autorizan al Sr. CAPT. Albán Ruiz Pablo Fernando con C.I: 171251016-1 y al Sr. Brito López Darío Xavier con C.I: 171809248-7 que lo entreguen al Doctor Nikolai Espinosa, en su calidad de Director de la Carrera.

Atentamente,

Ing. Rodolfo Gordillo.

DIRECTOR

Ing. Carlos Romero.

CODIRECTOR

DECLARACIÓN DE RESPONSABILIDAD

PABLO FERNANDO ALBAN RUIZ
DARIO XAVIER BRITO LOPEZ

DECLARAMOS QUE:

El proyecto de grado denominado “DISEÑO E IMPLEMENTACIÓN DEL PROTOTIPO DE UNA RED DEFINIDA POR SOFTWARE (SDN) EN LA UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE”, ha sido desarrollado con base a una investigación exhaustiva, respetando derechos intelectuales de terceros, conforme las citas que constan al pie de las páginas correspondientes, cuyas fuentes se incorporan en la bibliografía.

En virtud a esta declaración, nos responsabilizamos del contenido, veracidad y alcance del proyecto de grado en mención.

Sangolquí, 20 febrero del 2015.

Albán Ruiz Pablo Fernando

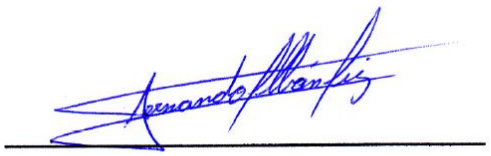
Brito López Darío Xavier

AUTORIZACIÓN

PABLO FERNANDO ALBAN RUIZ
DARIO XAVIER BRITO LOPEZ

Autorizamos a la Universidad de las Fuerzas Armadas “ESPE” la publicación, en la biblioteca virtual de la institución el trabajo “DISEÑO E IMPLEMENTACIÓN DEL PROTOTIPO DE UNA RED DEFINIDA POR SOFTWARE (SDN) EN LA UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE”, cuyo contenido, ideas y criterios son de nuestra exclusiva responsabilidad y autoría.

Sangolquí, 20 febrero del 2015.



Albán Ruiz Pablo Fernando

Brito López Darío Xavier

DEDICATORIAS

“No tenía miedo a las dificultades: lo que me asustaba era la obligación de tener que escoger un camino. Escoger un camino significaba abandonar otros”. Paulo Coelho

A Dios y al divino Niño Jesús.
A mi familia, los amo con mi alma y corazón.
Albán Ruiz Pablo Fernando

A todas las personas que creyeron en mí y que me daban palabras de aliento para seguir adelante, a aquellos que esperaban que lograra terminar mi carrera y que pudiera dar un paso importante en mi vida. Les dedico la culminación de este proyecto a mis padres, familiares y amigos que me dieron apoyo moral siempre e incondicionalmente.
Brito López Darío Xavier

AGRADECIMIENTOS

Agradezco a Dios y al divino Niño Jesús por ayudarme a concluir con éxito mi carrera. A mi Padre que desde el cielo me cuida, ilumina y protege, a mi madre por su fortaleza, sacrificio, enseñanzas, comprensión y amor. A mi familia, profesores y amigos por su amistad, consejos, ayuda, confianza, apoyo y motivación. Por siempre gracias a todos.

Albán Ruiz Pablo Fernando

Mi familia me ha inducido valores y conocimientos muy significativos en el transcurso de toda mi vida hasta ahora, es por ello que quiero agradecer en primer lugar a todos los miembros que la conforman, especialmente a mis padres, Gladys y Javier, que me han apoyado siempre e incondicionalmente, tanto en mis aciertos como también en mis equivocaciones, dándome consejos que me han ayudado a siempre salir adelante e indicándome que siempre hay un camino que seguir.

Agradezco a mis hermanos, Cristina y David que son muy importantes también en mi vida, por nunca dejarme solo y siempre acompañarme a toda hora y en todo lugar, dando siempre un paso a la vez pero siempre juntos.

Un sincero agradecimiento a mi director, Rodolfo Gordillo, y a mi codirector, Carlos Romero, por el tiempo, por sus sugerencias e ideas que han servido mucho para el avance del proyecto, por el respaldo y la amistad.

Y por último pero no menos importante quiero agradecer a mis amigos, que han llegado a ser parte de mi familia, porque siempre han estado conmigo a pesar de que he cometido muchos errores. A mi amigo y compañero de tesis, Pablo, ya que con él hemos superado todos los obstáculos que se han presentado. Gracias Dios por poner en mi camino a todas las personas que para mí son muy importantes.

Brito López Darío Xavier

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

En este proyecto se busca diseñar e implementar un prototipo de una red definida por software para la Universidad de Fuerzas Armadas ESPE, utilizando la infraestructura del Consorcio Ecuatoriano para el Desarrollo de Internet Avanzado, desarrollando aplicaciones de controladores OpenFlow que se ajuste a cualquier red, que permitirá una mayor eficiencia en el servicio brindado a sus clientes. Dentro de las aplicaciones desarrolladas se comparan los diferentes controladores y lenguajes de programación que cada uno de ellos utiliza, buscando la mejor disponibilidad y optimización de la capacidad de la red, la comparación entre controladores se la realiza mediante el análisis de paquetes enviados entre controlador y dispositivos de la red y esto permite verificar el estado y funcionamiento de las aplicaciones desarrolladas. La aplicación de nuevas tecnologías orientadas a la virtualización de las redes facilitan el intercambio de datos, esto puede ser aplicado en la administración de diferentes tipos de redes. El propósito de este trabajo es diseñar una aplicación basada en el controlador POX en el lenguaje de programación Python, asegurando la optimización de recursos para su implementación. Se detalla la arquitectura, programación y estudio de los resultados de la aplicación.

PALABRAS CLAVES:

- PYTHON
- MININET
- RED DEFINIDA POR SOFTWARE
- OPENFLOW

ABSTRACT

This project seeks to design and implement a prototype of a software defined network for ESPE University, using the infrastructure of CEDIA, developing OpenFlow controllers' applications that will fit any network, allowing network higher efficiency in the service provided to customers. Among the various controllers developed applications and programming languages that each uses, looking for the best availability and optimization of network capacity, the comparison between controllers is done by analyzing packets sent between controller are compared and network devices and this allows to check the status and operation of developed applications. The implementation of new technologies oriented networks facilitate exchange of data, this can be applied in the administration of different types of networks. The purpose of this paper is to design an application based on the POX controller in the Python programming language, ensuring the optimization of resources for implementation. Architecture, programming and studying the results of the implementation is detailed.

DISEÑO E IMPLEMENTACIÓN DEL PROTOTIPO DE UNA RED DEFINIDA POR SOFTWARE (SDN) EN LA UNIVERSIDAD DE LAS FUERZAS ARMADAS “ESPE”

El presente proyecto abarca el proceso de diseño e implementación de un prototipo de una red SDN empleando el lenguaje de programación Python en el controlador.

Inicialmente en el primer capítulo se desarrolla los contenidos del proyecto como son: la presentación, resumen, objetivos, justificación e importancia y alcance del proyecto, inmediatamente se describe en el segundo capítulo con los fundamentos teóricos que se requieren para poder conocer y comprender las redes definidas por software, sus características generales, su arquitectura y dispositivos que se pueden encontrar dentro de una red de este tipo, además en este capítulo se describe el protocolo OpenFlow, encargado de la comunicación entre los dispositivos dando mayor importancia al controlador por su relevancia, ya que es el dispositivo inteligente de la red.

En el tercer capítulo se describe el entorno virtual de la herramienta Mininet usada para realizar las respectivas simulaciones y pruebas de una red definida por software con diferentes topologías y distintos componentes de hardware y de software.

El cuarto capítulo está dedicado a la implementación de las diferentes topologías mediante una plataforma de hardware con un servidor controlador, conmutadores y host, tomando en cuenta las distintas opciones de controladores.

En el quinto capítulo se presentan de pruebas y resultados obtenidos de la realización tanto de la simulación como en la implantación de la red definida por software, donde se evidencia las ventajas y desventajas,

semejanzas y diferencias entre los distintos controladores y sus lenguajes de programación.

El sexto capítulo presenta las conclusiones y recomendaciones del proyecto, además se incluyó la información, respaldada en formato digital, necesaria para la ejecución del proyecto, tanto en simulación como en implementación del prototipo de la red definida por software.

CAPITULO 1: INTRODUCCIÓN

1.1. Presentación

Hoy en día las redes de computadoras son de suma importancia en nuestra vida laboral, estudiantil, social, etc. Ya que son un conjunto de equipos de comunicación conectados por medio de cables, señales, ondas o cualquier otro método de transporte de datos, que comparten información, recursos, servicios, etc.

Uno de los inconvenientes de las redes actuales es que no poseen flexibilidad concerniente a la programabilidad de sus elementos, para que la calidad de servicio sea más dinámica, personalizada y adaptable a las necesidades de la organización en particular.

Una red definida por software (SDN) es una tecnología que aborda la creación de redes donde el control se desprende del hardware. La administración se enfoca en una aplicación de software llamada controlador donde se ejecutan una serie de reglas programadas (Principia technologica, 2014).

La solución SDN permite cambiar el comportamiento de una red dinámicamente, mediante la creación de aplicaciones personalizadas. Esta nueva tecnología permitirá establecer de forma instantánea los servicios, ya que las redes serán más flexibles, escalables, administrables y rentables.

La Universidad de Fuerzas Armadas (ESPE) no cuenta con temas de investigación referentes a las Redes Definidas por Software, por esta razón se quiere incursionar en este tema de actualidad. El cambio constante de la tecnología se siente a nivel mundial, ante esto nuestro país no se puede quedar aislado del desarrollo tecnológico (Principia technologica, 2014).

1.2. Objetivos

1.2.1. General

Diseñar e implementar un prototipo de una red definida por software (SDN) con solución basada en hardware para la Universidad de Fuerzas Armadas ESPE.

1.2.2. Específicos

- Proporcionar una introducción a las redes definidas por software (SDN) y el protocolo Openflow, sus características generales, su arquitectura, ventajas y desventajas.

- Diseñar, abstraer y programar los elementos de la Red mediante la simulación en el programa Mininet. Seleccionar diferentes controladores dependiendo de la operatividad, funcionalidad y facilidad de uso de los mismos.

- Implementar, controlar y gestionar de forma centralizada los dispositivos que comprenden la topología planteada en el diseño de la Red Definida por Software (SDN).

- Realizar la evaluación de la funcionalidad y flexibilidad de la Red Definida por Software (SDN) de la ESPE.

1.3. Justificación e importancia del proyecto

La Universidad de Fuerzas Armadas ESPE en la actualidad no posee flexibilidad en la administración de su red, por lo tanto se ha considerado la propuesta de implementar un prototipo de redes definidas por software (SDN) y evaluar diferentes controladores, debido a que:

- Las SDN migran el “plano de control” de cada dispositivo a un controlador central que gestiona todos los recursos, permitiendo cambiar el comportamiento de la red dinámicamente mediante software. El control y monitoreo se facilitará para el administrador de la red, ya que la red podrá reconfigurarse convergiendo a una posible solución ante un fallo (Azodolmolky, 2013).

- Por ser una tecnología nueva no se encuentra implementada en la red de la Universidad de Fuerzas Armadas ESPE ni en nuestro país. Las SDN mediante su flexibilidad permiten la innovación para nuevas tecnologías.

- Con la aplicación de las SDN se tendrá la posibilidad de automatizar una red y de garantizar confidencialidad de la información, aplicable en los campos sociales, militares, académicos e industriales.

- Con la utilización de esta nueva tecnología, no se encuentra el usuario atado a la adquisición de ciertos equipos por su marca, ya que las redes definidas por software son configurables en todos los equipos que soporten el protocolo OpenFlow.

La importancia de este proyecto consiste en que es necesario garantizar la flexibilidad de la red con un sistema de control mediante la programabilidad. El prototipo de SDN planteado para la ESPE pondrá a disposición de docentes y estudiantes una plataforma para impulsar el desarrollo de investigaciones en esta nueva tecnología, donde se podrá cambiar el comportamiento de la red dinámicamente, obteniendo una red segura y adaptable. Además se podrá evitar los sistemas de redes cerrados aprovechando la utilización de software libre.

1.4. Alcance del proyecto

El alcance del Proyecto incluye:

- Levantamiento de la información de la arquitectura de redes definidas por software (SDN) y el protocolo OpenFlow, software Mininet y controladores SDN.

- Instalación del entorno de simulación Mininet, estudio y simulación de varias opciones de controladores. Crear la infraestructura de la topología a simular y evaluar los diferentes controladores dependiendo de las aplicaciones requeridas.

- Instalación de la topología de red, conforme el estudio del proyecto del Consorcio Ecuatoriano Para el Desarrollo de Internet Avanzado (CEDIA), con el tema "Implementación de un testbed para una SDN empleando la infraestructura de CEDIA" (CEPRA, CEDIA, 2014), con equipos reales que soporten el protocolo OpenFlow, en las instalaciones de la Universidad De Fuerzas Armadas ESPE.

- Evaluación del funcionamiento de la red mediante pruebas y análisis de resultados tanto en la simulación como en la implementación de la red.

CAPITULO 2: MARCO TEÓRICO

2.1. Características generales de las SDN (Software Defined Networks)

En el mundo actual se necesitan que las redes se ajusten y respondan de forma dinámica bajo políticas actualizadas, con el fin de poder reducir el trabajo manual y los costos asociados con la administración de las redes. Es decir, se necesita poder implementar y ejecutar con rapidez nuevas aplicaciones dentro y por encima de las redes; este ambicioso desafío es afrontado por las SDN.

Por este motivo en este capítulo se estudiará las SDN que surgen como opción en el mundo ya que las redes han estado limitadas por la forma en que el software las ha configurado, suministrado y gestionado además las ha actualizado de forma compacta y las ha gestionado mediante líneas de comandos limitándolas.

Después se estudiará al encargado de la comunicación entre el controlador y los dispositivos de la red “el protocolo OpenFlow” y para cerrar el capítulo se estudiará el entorno de simulación Mininet y los diferentes controladores los cuales se emplearán en el desarrollo del presente proyecto.

2.1.1. Definición

Las redes definidas por software (SDN: Software Defined Networking) son aquellas que brindan la posibilidad de llevar a cabo un control mucho más directo de su comportamiento y posibilitan la interactuación directa con la red como si fuera un todo. Un conmutador, un router, un firewall, etc, puede considerarse compuesto de dos funciones básicas conocidas como planos y a estos se les da nombres correspondientes a su función: el plano de control y el plano de datos.

El primero resuelve cómo se trasladan los datos como por ejemplo: entre qué dos puertos de un conmutador, hacia qué enlace en un router, si se filtran o no en un firewall, etc.

Y el segundo plano se encarga de hacer seguras estas transferencias de esos datos.

Por este motivo se puede decir que cada elemento de red tiene una instancia de ambos planos además que es necesario que estos dos elementos coordinen y comuniquen su estado entre si ya que cada plano de control se comunica internamente con su correspondiente plano de datos (Kumar, 2013).

En SDN, el plano de control y el de datos se desacoplan totalmente. El elemento del plano de control, el controlador y los elementos del plano de datos, los conmutadores, son normalmente elementos separados, incluso físicamente, y se comunican por medio de un protocolo estándar.

Las SDN permiten tener redes más maleables y escalables que se adaptan cómodamente a cambios en ella.

En las SDN la manera estándar de emplearlas es con un solo controlador que administra la actuación de los conmutadores en una red. Así se percibe que toda la red es una sola unidad, ya que el controlador precisa la conducta de la red como un todo. Esta particularidad permite mejoras en la seguridad y monitorización (Lopez, 2012).

2.1.2. Arquitectura SDN

La arquitectura SDN está basada en tres componentes: Infraestructura de Red, Controlador y Aplicaciones.

La infraestructura de Red hace relación a los dispositivos de red como por ejemplo routers y switches, estos a su vez pueden ser físicos (hardware) y/o virtuales (software).

El Controlador es el software instalado en un servidor que admite controlar de una forma lógica y centralizada a la red y que dispone de una interface de programación de aplicaciones (API) abierta para participar con los dispositivos de red (Principia technologica, 2014).

Las aplicaciones en las SDN como por ejemplo: Aplicaciones de voz, video, servicios de red pueden comunicarse con el controlador usando una API abierta, y solicitar el tratamiento que requieran (Azodolmolky, 2013).

OpenFlow es el protocolo empleado para comunicación entre el controlador (plano de control) y el switch (plano de datos), hay que recalcar que no es el único protocolo para este propósito pero entre sus características es que es el único estándar abierto.

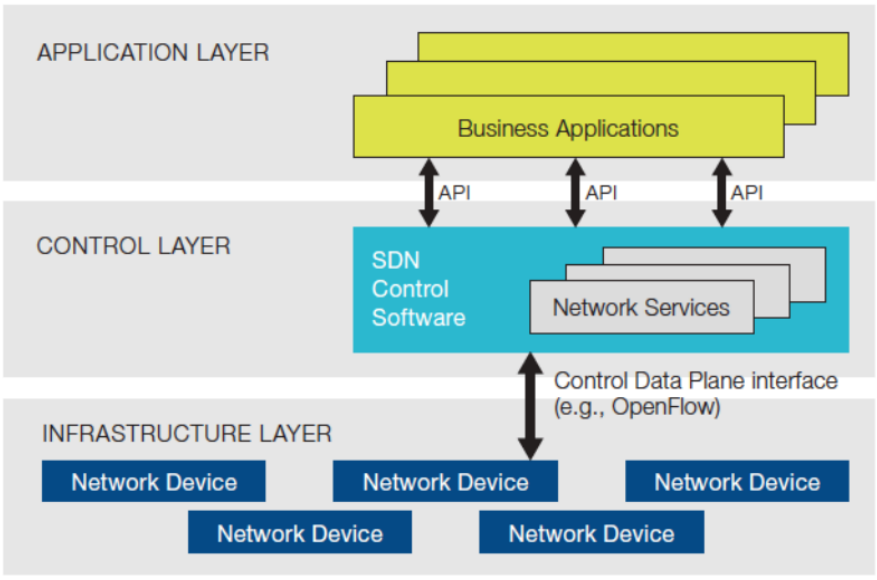


Figura 1. Arquitectura SDN (Principia technologica, 2014)

2.1.3. Ventajas de la arquitectura SDN

Dentro de los beneficios que brindan de las SDN a sus usuarios dentro de su implementación podemos citar:

- Dividen el software de red en cuatro planos o niveles: administración, servicios, control y reenvío ofreciendo así el apoyo ordenado necesario para mejorar cada plano dentro de la red.

- Concentran la información necesaria de los niveles de administración, servicios y control abreviando el diseño de red y así comprimir los costos operativos (Principia technologica, 2014).

- Manejan la nube para la obtención de una implementación flexible, adaptable y escalable permitiendo una distribución de costos basada en el uso con el fin de reducir el tiempo de servicio.

- Implantan un escenario para las aplicaciones, los servicios y la integración de la red en los sistemas de administración, lo que permite el perfeccionamiento de nuevas soluciones.

- Admiten una estandarización de los protocolos, lo que da lo posibilidad de elección entre proveedores reduciendo costos y así poder tener diversa asistencia (Principia technologica, 2014).

- Emplean con una extensa proyección los principios de las SDN a todos los servicios de red, incluidos los de seguridad. En todo tipo de redes inclusive redes móviles e inalámbricas manejadas por los proveedores de servicio.

2.1.4. Servidor Controlador SDN

El controlador SDN es la parte central de la arquitectura SDN, debido a que es el que mantiene toda la inteligencia de la red. En el controlador, el administrador se encarga de definir reglas para administrar el flujo de datos en la red. Esto permite una configuración rápida, que supone una ventaja frente a las redes tradicionales, en las cuales se debe esperar a que el fabricante lo haga, al igual que la implementación de nuevas aplicaciones y servicios (Kumar, 2013).

El controlador ejecuta un software que le permite llevar a cabo sus tareas, que puede ser: NOX, POX, Beacon, Floodlight u otro, diferenciándose uno de otro solamente por el lenguaje de programación con el cual se definan las reglas para el flujo de datos.

2.2. Arquitectura de red tradicional vs arquitectura SDN

En la comparación entre arquitecturas de red tradicional y la arquitectura SDN vamos a observar los causantes que originan un cambio de tecnología en las redes, así como la propensión marcada hacia las SDN.

Las redes tradicionales han ido avanzado a lo extenso de la historia con la intención de compensar las necesidades tecnológicas de las empresas, de los operadores y de los usuarios.

Las redes actuales ya no satisfacen las necesidades tecnológicas. Otras tecnologías han evolucionado de manera más rápida, haciendo de la red el lugar de fisura que no permite el crecimiento de éstas. Nos referimos a que las redes tradicionales no son suficientemente rápidas como para reprogramarse, reconfigurarse y reajustarse a los despliegues de nuevos servicios, les hace falta flexibilidad.

Se puede decir que los esquemas de tráfico han cambiado con los dispositivos móviles como smartphones, tablets, notebooks, la virtualización, etc.; requieren diferentes puntos de acceso a la red y se pretende mantener el estándar de seguridad y resguardo de datos, las actuales redes para permitir este tipo de accesos requieren acciones a nivel físico en los dispositivos, a veces nuevo cableado entre equipos, haciendo una vez más lentos el despliegue de nuevos servicios (Tiwari, 2013).

El tráfico generado supone la mayor carga de trabajo sobre la red, en las arquitecturas tradicionales gran parte del tráfico de red viaja hacia los clientes, donde reside parte de la lógica de las aplicaciones, por lo que estas redes no están preparadas para ese esquema de tráfico, porque son arquitecturas jerárquicas y a pesar de ser estables también son altamente estáticas, lo que impide flexibilizar la red. Sea cual fuere la solución, mayor tráfico implica flexibilizar la red para estas nuevas necesidades.

En la arquitectura tradicional una aplicación corre en un servidor e intercambiaba tráfico fundamentalmente con sus clientes. Pero hoy, las aplicaciones están distribuidas a lo largo de múltiples máquinas virtuales, que intercambian tráfico unas con otras y que pueden ser movidas de servidor físico para optimizar los servicios y balancear la carga de los servidores. Mientras que cada aplicación sigue teniendo sus requerimientos de red (Redclara).

Hoy en día se necesita miles de servidores trabajando en procesamiento paralelo, lo que genera una inimaginable cantidad de datos que debe poder conectarse any -to- any. Esto implica una vez más la capacidad de adaptar la red a una nueva necesidad de ancho de banda, escalabilidad que no disponen las redes actuales.

En función de especificaciones determinadas de topología, de velocidad, de distancia, de número de hosts a conectar, de ancho de banda las redes

se fueron diseñando y se han creado protocolos fiables y seguros que funcionan correctamente bajo dichas especificaciones.

Las redes tienden a ser más convergentes, para operar tráfico de voz, vídeo y datos en una misma red con diferentes calidades de servicio. Para ello, es necesario dotar al equipamiento físico y lógico de más dinamismo y flexibilidad (Principia technologica, 2014).

Las redes tradicionales están descentralizadas en el control. La complejidad de las redes de hoy en día hace que sea muy difícil aplicar un conjunto coherente de acceso, seguridad, calidad de servicio y otras políticas a los usuarios cada vez más móviles, lo que deja a la empresa vulnerable a violaciones de seguridad, incumplimiento de las normas y otras consecuencias negativas.

La compleja incorporación de cientos o miles de dispositivos que deben ser configurados y gestionados da lugar a la dificultad de aplicar políticas en las redes tradicionales ya q son incapaces de escalar.

La falta de interfaces estándar y libres limita la capacidad de los operadores para adaptar la red a sus entornos particulares. Este desajuste entre las necesidades del mercado y las capacidades de la red ha llevado a la industria a un punto de inflexión. En respuesta, la industria ha creado el Software- Defined Networking (SDN), la arquitectura y el desarrollo de normas asociadas (Principia technologica, 2014).

2.3. El protocolo OpenFlow

El emergente estándar abierto de OpenFlow, definido por la Open Networking Foundation (ONF) desde 2011 y es el protocolo que permite acceder directamente y manipular el plano de redireccionamiento de dispositivos de red como switches y enrutadores, ya sean físicos o virtuales ya que es un protocolo de comunicación de estándar abierto. La interfaz de

OpenFlow ofrece acceso y comunicación entre las capas de control e infraestructura de la arquitectura de SDN, tanto de forma física como virtual. Al centralizar el control de los dispositivos de capas de infraestructuras, OpenFlow simplifica la administración de las redes y la capacidad de programación que promete SDN (Azodolmolky, 2013).

Entre sus características tenemos:

- Simplifica la administración de redes y la programación de dispositivos de redes.
- Permite cambios dinámicos en el flujo de tráfico.
- Permite que la red tenga mayor capacidad de respuesta para las necesidades empresariales.

En resumen, el protocolo OpenFlow indica al tráfico como fluir, para lo que se toman en consideración varios parámetros como el patrón de uso de la red, las aplicaciones y los requerimientos específicos de cada una de ellas. Las reglas son definidas para cada flujo, con lo que se obtiene un control sumamente granular que permite a la red tener una gran capacidad de respuesta a los cambios que pueden originarse (Ferro, 2014).

El protocolo OpenFlow toma decisiones de cómo se reenvían los paquetes de los switches a los controladores. Una aplicación de gestión se ejecutará en las interfaces del controlador que une todos los switches en la red, facilitando la configuración de caminos de reenvío que utilizarán todo el ancho de banda disponible. La especificación define el protocolo entre el controlador y los switches y un conjunto de operaciones que se pueden realizar entre ellos (Tiwari, 2013).

Las instrucciones de reenvío se basan en el flujo, que consiste en que todos los paquetes comparten una serie de características comunes como por ejemplo algunos criterios que podemos incluir:

- los puertos por donde se reciben los paquetes cuando llegan
- el puerto Ethernet de origen
- la etiqueta VLAN
- el destino Ethernet o el puerto IP, etc.

Un nuevo flujo se debe crear cuando un paquete que llega no encuentra ninguna coincidencia con ninguna entrada de la tabla. El switch debería tener configurado un descartado de paquetes para el flujo que no haya sido definido, pero en la mayoría de los casos, el paquete será enviado al controlador. El controlador entonces define un nuevo flujo para ese paquete y crea una o más entradas para la tabla. Éste envía la entrada o entradas al switch para que sean añadidas a las tablas de flujo. Finalmente, el paquete se envía de vuelta al switch para ser procesado con las nuevas entradas creadas.

2.3.1. Conmutador OpenFlow

Un conmutador OpenFlow trabaja con lo que se denominan tablas de flujos y aprovecha que la mayoría de los switches y routers Ethernet modernos contienen dichas tablas de flujos. A pesar de que cada flow-table es diferente para cada fabricante, se han identificado un conjunto de funciones comunes o genéricas entre ellos.

El camino de datos de un conmutador OpenFlow consiste en una flow-table y una acción asociada a cada entrada. El conjunto de acciones soportadas es extensible, pero existe un conjunto de requerimientos mínimo para todos los switches. Para obtener un alto rendimiento y un bajo coste (Azodolmolky, 2013).

Las tablas de flujos (flow-table) tienen una serie de entradas con una acción asociada a cada una, para indicar al switch la manera de gestionar el flujo de entrada (flow-entry).

Una flow-table se compone de distintas entradas. Cada entrada en la flow-table contiene los siguientes campos:

- Match Fields: para detectar los paquetes. Compuesto por el puerto de entrada y las cabeceras de los paquetes.
- Priority: orden de preferencia de "matching" del flujo de entrada.
- Counters: para actualizar los paquetes coincidentes.
- Instructions: para modificar el conjunto de acciones o el procesamiento del pipeline.
- Timeout: tiempo para que expire el flujo de entrada en el switch.
- Cookie: es un valor opaco que usa el controlador SDN. No se emplea para procesar flujos.

Cada flujo de entrada (flow-entry) es analizado en la tabla:

1. En primer lugar se busca el paquete entrante coincidente con la mayor prioridad.
2. Una vez detectado se aplican las distintas instrucciones.
3. Por último, se envía el dato coincidente a la siguiente tabla (flow-table) junto con el conjunto de acciones.

El conmutador OpenFlow requiere un canal seguro para la comunicación con el controlador SDN. Es a través de este canal que ambos dispositivos hablan el protocolo OpenFlow (Tiwari, 2013).

El análisis de estos flujos de entrada puede ser tan fino como se quiera. Es aquí donde reside la versatilidad y la capacidad de programación de SDN.

Los flujos están ampliamente definidos y sólo están limitados por las capacidades de la implementación de la flow-table en particular. Por ejemplo, un flujo puede ser una conexión TCP, o todos los paquetes de una

dirección MAC o IP en particular, o todos los paquetes con la misma arquitectura dentro de una VLAN, o todos los paquetes pertenecientes al mismo puerto del switch. Para los experimentos que no incluyan paquetes IPv4, un flujo puede ser definido como todos los paquetes que coincidan con una cabecera específica (pero no estándar).

Cada flujo de entrada tiene una acción simple asociada las tres más básicas son las siguientes:

- Reenvío de los paquetes de un flujo particular a un determinado puerto (o conjunto de puertos). Esto permite que los paquetes sean enrutados a través de la red.

- Encapsular y reenviar los paquetes de un flujo a un controlador. El paquete se entrega al canal seguro, donde se encapsula y se envía a un controlador. Normalmente se utiliza para el primer paquete en un flujo nuevo, para que el controlador pueda decidir si el flujo debe ser añadido a la tabla de flujos. Alternativamente, se podría utilizar como sniffer para transmitir todos los paquetes a un controlador para su procesado.

- Descartar los paquetes de ese flujo. Puede ser utilizado como mecanismo de seguridad.

2.4. Mensajes OpenFlow

Los mensajes OpenFlow son los que permiten transmitir acciones e información desde el servidor controlador hacia el o los dispositivos de red hacia el controlador. Se dividen en: mensajes del controlador al conmutador, mensajes asincrónicos y mensajes simétricos (Azodolmolky, 2013).

2.4.1. Mensajes del controlador al conmutador

Mensajes remitidos por el controlador y no precisamente requieren de una respuesta por parte del conmutador (Azodolmolky, 2013).

- Features: mensaje remitido cuando el controlador requiere conseguir las particularidades del conmutador. El conmutador le responde con sus características. Normalmente usado en el establecimiento de una conexión OpenFlow.

- Configuration: mensajes de consulta de los parámetros de configuración del conmutador.

- Modify-State: mensaje remitido desde el controlador para la gestión de estados en el conmutador (añadir o quitar flujos o cambiar el estado de un determinado puerto).

- Read-State: mensaje enviado por el controlador para censar las características del conmutador.

- Packet-Out: mensaje usado por el controlador para enviar paquetes por un determinado puerto del conmutador y/o para redireccionar paquetes Packet-In. Este mensaje contiene el paquete a ser redireccionado y las acciones que se aplicarán al mismo.

- Barriel: mensaje usado por el controlador para cerciorarse que las dependencias de un mensaje se han cumplido o para recibir notificaciones por operaciones finalizadas.

2.4.1.1. Mensajes asincrónicos

Estos mensajes son generados por los conmutadores cuando reciben un paquete y ha ocurrido un cambio en su estado o ha ocurrido un:

- Packet_In: mensaje remitido del conmutador al controlador cuando no tiene una entrada en su tabla de flujos que concuerde con el paquete entrante. El controlador procesa el paquete y responde con un mensaje Packet-Out.

- Flow-Removed: mensaje que puede ser enviado tanto por el controlador como por el conmutador cuando el tiempo de inactividad de un flujo o el tiempo de vida del flujo vence.

- Port-Status: mensaje remitido del conmutador al controlador cuando la configuración de un puerto cambia.

- Error: mensaje remitido del conmutador al controlador notificando si existen problemas con esos paquetes (Azodolmolky, 2013).

2.4.1.2. Mensajes simétricos

Estos mensajes se envían en cualquier dirección sin una solicitud previa. En esta clasificación se encuentran los siguientes mensajes:

- Hello: mensajes que se intercambian entre el conmutador y el controlador al momento del establecimiento de la conexión.

- Echo: mensaje usado para medir la latencia o el ancho de banda o para verificar que un dispositivo esté activo. Puede ser generado por el conmutador o el controlador. Por cada petición que llegue al destino, se generará una respuesta que será enviada al origen.

- Experimenter: mensaje que permite al conmutador OpenFlow tener funcionalidades adicionales, paquete planeado para futuras versiones de OpenFlow (Tiwari, 2013).

2.5. Python

Python es uno de los idiomas más fáciles de computadora para entender, aprender y utilizar, debido a su la sintaxis legible, similitud con otros lenguajes orientados a objetos, y muchas bibliotecas útiles (Doxygen, 2015).

2.6. Mininet

Existen varios programas de simulación que nos permiten interactuar con equipos de red virtuales que nos ayudan a no depender de los reales, es decir, se puede crear varias topologías de red con equipos virtuales. Uno de los entornos de software de simulación es Mininet que permite crear switches, hosts y controladores SDN con tan solo pocas líneas de comando.

Lo que nos permite adentrarnos en el mundo de las redes definidas por software sin la necesidad de tener equipos SDN reales. Existen otros simuladores de red tales como OpenFlow VMS o Netkit pero a diferencia de Mininet, estos no son tan robustos. Mininet dispone además de rapidez, se consigue crear topologías y ponerlas a trabajar en tan solo unos segundos. Además se puede crear y ejecutar experimentos de red escribiendo o reutilizando códigos en Python. Una de las desventajas de Mininet es que éste simulador utiliza un controlador específico y no un personalizado.

Para poder utilizar Mininet y Netkit es necesario un equipo con sistema operativo GNU/Linux o a su vez un equipo con un motor de máquina virtual tal como Virtual Box, VMWare, etc. En el presente proyecto se utilizará Virtual Box con una máquina virtual que contiene Linux con la versión de

kernel 3.5.0-17, y que a su vez tiene instalado la versión 2.1.0 de Mininet (Doxygen, 2015).

En el actual proyecto usamos Mininet por muchas razones entre ellas: Es rápido ya que en una red simple tarda sólo unos segundos. Puede crear topologías personalizadas con un solo interruptor o grandes topologías tipo Internet, incluso un centro de datos.

Además Mininet puede ejecutar programas reales ya que todo lo que se ejecuta en Linux está disponible, desde los servidores de Internet para TCP hasta las herramientas de monitoreo como Wireshark.

Se puede personalizar el reenvío de paquetes mediante interruptores de Mininet ya que son programables usando el protocolo OpenFlow. Los diseños de red definida por software personalizados que se ejecutan en Mininet se pueden transferir fácilmente a los switches OpenFlow reales para el envío de paquetes a velocidad de línea.

También se puede ejecutar Mininet en su computadora portátil, en un servidor, en una máquina virtual, en una máquina Linux nativo o en la nube. Puede compartir y replicar los resultados ya que cualquier persona con una computadora puede ejecutar el código.

Dentro de otras características tenemos que se puede utilizar fácilmente ya que se puede crear y ejecutar experimentos tanto simples como complejos en Mininet utilizando scripts de Python y como Mininet es un proyecto de código abierto se puede examinar su código fuente, corregir los errores, problemas de archivos, peticiones de características, y enviar solicitudes de parches.

Si bien es cierto solo se ha topado lo positivo de Mininet pero sabemos que está bajo desarrollo activo por lo que en la comunidad de usuarios y

desarrolladores de Mininet están en constante investigación para solucionar algún inconveniente que pueda presentarse y dentro de las limitaciones que encontramos tenemos que es conveniente ejecutarlo en un solo sistema por lo que impone límites de recursos, por ejemplo si su servidor tiene 3 GHz de velocidad en su CPU y puede cambiar alrededor de 3 Gbps de tráfico simulado, se tendrán que equilibrar y compartir esos recursos entre los hosts virtuales y switches (Doxygen, 2015).

También se conoce que Mininet utiliza un solo núcleo de Linux para todos los hosts virtuales y además no escribirá solo el enrutamiento personalizado o el comportamiento de conmutación, sino se tiene que encontrar o desarrollar un controlador con las características que se requiere.

Actualmente Mininet no realiza NAT, esto significa que sus máquinas virtuales se pueden aislar de la red LAN de forma predeterminada; significa que sus anfitriones no pueden hablar directamente a Internet a menos que proporcione un medio para que lo hagan como agregar una interfaz física.

Cabe recalcar que Mininet no tiene una fuerte noción de tiempo virtual; esto significa que las mediciones de temporización se basarán en tiempo real.

Lo más importante en el rendimiento para los experimentos de red limitada es que probablemente se tendrá que utilizar los enlaces más lentos, por ejemplo 10 o 100 Mb/s en lugar de 10 Gb/s, debido al hecho de que los paquetes son transmitidos por un conjunto de conmutadores de software, los recursos de la CPU que comparten y de memoria, y por lo general tienen un menor rendimiento que el hardware de conmutación dedicada. Si se preocupa principalmente de la corrección funcional, puede ejecutar Mininet sin límites específicos de ancho de banda así se consigue la manera rápida

y fácil de ejecutar Mininet, y también proporciona el más alto rendimiento a expensas de la precisión de sincronización con carga (Doxygen, 2015).

2.7. Controladores SDN

En la actualidad se han desarrollado varios controladores SDN, los cuales son el motor de todas las políticas o acciones que se debe tomar en una red SDN. En teoría, un controlador SDN ofrece servicios que pueden realizar un plano de control distribuido que se han creado con código abierto ya sea por empresas o por colaboradores de entidades sin fines de lucro.

La diferencia entre cada controlador está en el lenguaje de programación y la plataforma en la que éste trabaje, pero la funcionalidad es la misma ya que todos tienen que transmitir y recibir paquetes OpenFlow para comunicarse con los dispositivos SDN.

2.7.1. POX (Python)

El controlador POX es la evolución por así decirlo del controlador NOX clásico, particularmente éste controlador permite un rápido desarrollo y creación de prototipos de software de control de red usando el lenguaje de programación Python. Es un controlador de varios existentes como FloodLight, Reflector, Trema, etc., los cuales nos permiten crear aplicaciones para gestionar una red e interactuar con switches OpenFlow.

POX es también un controlador base para algunos experimentos en Redes Definidas por Software (SDN), básicamente a éste controlador se lo ha usado para implementar varios prototipos de SDN. Es por esto que se lo ha escogido como controlador para el presente proyecto. Ahora, POX se halla constantemente en perfeccionamiento pero el objetivo principal es obtener una API para fines de experimentación, investigación y estudio.

Para un correcto funcionamiento POX requiere de la versión 2.6 o 2.7 de Python y puede ser ejecutado en prácticamente cualquier sistema operativo que tengan o soporten dichas versiones de Python.

POX actualmente trabaja con switches que soportan la versión 1.0 del protocolo OpenFlow. Para ejecutar dicho controlador es necesario llamar al subprograma pox.py seguido de los diferentes módulos con los que se vaya a trabajar y se va armando una especie de línea de código, esta línea es depurada y se anuncian todos los errores existentes en cada módulo, si existen, para luego corregirlos y poder correr el controlador e identificar a todos los switches con OpenFlow existentes en la red. También existen opciones que se las agrega para especificar por ejemplo la dirección y el puerto del controlador (McCauley, 2014).

2.7.2. Pyretic (Python)

Pyretic es un lenguaje de programación SDN que junto con el controlador POX permiten a los administradores de red realizar aplicaciones modulares y robustas, reutilizando módulos y código ya anteriormente escrito, o partiendo de un punto determinado. Pyretic es un lenguaje asociado con Python, ya que todos los módulos se los escribe en dicho lenguaje, y todas las aplicaciones realizadas en Pyretic pueden ser perfectamente ejecutadas en un servidor controlador que trabaje a la par con conmutadores de red que trabajen con el protocolo OpenFlow.

Pyretic también soporta la ejecución en paralelo pero ésta tiene que ser secuencial. Es decir, un administrador que esté usando Pyretic puede dar prioridades a diferentes módulos, es decir, podría especificar que se va a ejecutar un módulo de DHCP antes que un módulo de balanceo de carga por ejemplo. Los programadores también pueden especificar políticas que pueden cambiar o no con el tiempo (GitHub, 2014).

El equipo Pyretic está añadiendo características como el soporte para la calidad del servicio y se está desarrollando aplicaciones, incluyendo servidores RADIUS y DHCP e incluso se está mejorando la eficacia de su compilador y la verificación de la generación de código correcto.

Con este tipo de lenguaje podemos establecer políticas o reglas mejor estructuradas pero más simplificadas y menos propensas a errores. Cuando declaramos reglas para nuestro controlador debemos establecerlas tomando en cuenta la acción que queremos que se tome para así garantizar que todos los comandos se ejecuten en el orden adecuado (Frenetic - lang, 2011).

2.7.3. PyResonance (Python)

PyResonance es una plataforma Resonance implementada con Pyretic.

Resonance es una plataforma de control SDN que se implementó inicialmente con NOX, y que trabaja en un plano de gestión donde los administradores de red implementan las reglas de control como una máquina de estados finita.

Resonance hace que las redes sean incluso más fáciles de manejar mediante la automatización de las acciones que el controlador debe tomar ya que el cambio entre los diferentes estados son provocados por varios eventos improvistos como por ejemplo la detección de un intruso o la autenticación de diferentes hosts en la red (GitHub, 2013).

PyResonance está construido en la cima de Pyretic, presentando un entorno atractivo y atrayente para la implementación de los controladores basados en resonancia. Con esto se puede establecer todas las políticas y se las puede fusionar en una sola política que abarque reglas secuenciales o reglas en paralelo.

Con PyResonance, los administradores/programadores de red pueden construir varias reglas en una sola regla en general que puede cambiar dinámicamente el comportamiento de la red en función de diversos tipos de eventos imprevistos. PyResonance apoya la gestión de red federada, donde varios operadores están a cargo de las diferentes partes de una empresa o de la red del campus (GitHub, 2013).

CAPITULO 3: SIMULACIÓN DE LA RED

3.1. Entorno de simulación en hardware y software

Para realizar la simulación de las computadoras y de los switches SDN se utilizó Virtual Box con una máquina virtual que contiene Linux con la versión de kernel 3.5.0-17, y que a su vez tiene instalado la versión 2.1.0 de la herramienta de simulación de Mininet. La máquina virtual se la instaló en dos computadoras de prueba una con sistema operativo Windows 7 y otra con la versión 10.9.3 de OS X. Para la descarga de Virtual Box en Windows, Linux y OS X, se puede utilizar el siguiente link: https://www.virtualbox.org/wiki/Downloads.

Es necesario una conexión desde la máquina virtual y la PC y también es necesario ver el terminal de las PC's simuladas en Mininet por lo tanto se debe utilizar un terminal SSH y un X Server. Para Windows instalar Xming y Putty. Para MAC OS instalar XQuartz.

La conexión SSH es importante establecerla ya que se utilizará dos o más sesiones con la máquina virtual. La primera sesión se utilizará para conectarnos con la máquina virtual y estrictamente levantar el entorno de simulación Mininet y las demás para conectarnos con uno o varios de los controladores.

Para la configuración de la Máquina Virtual, se inicia Virtual Box, a continuación, se selecciona Archivo e Importar servicio virtualizado y se selecciona la máquina virtual que se desea importar. Pulse el botón "Siguiente" y luego en el botón "Importar", Figura 2 y 3.

Figura 2. Importando la Máquina Virtual a VirtualBox

Figura 3. Preferencias del Servicio de Máquina Virtual

Se debe hacer doble clic en la máquina virtual dentro de la ventana de VirtualBox.

En la consola de la máquina virtual, se debe iniciar sesión con el nombre de usuario y contraseña.

Nombre de usuario: mininet

Contraseña: mininet

Para que se pueda ejecutar comandos con permisos de root se debe teclear dichos comandos anteponiendo sudo, donde comando es el comando que desea ejecutar con permisos de root.

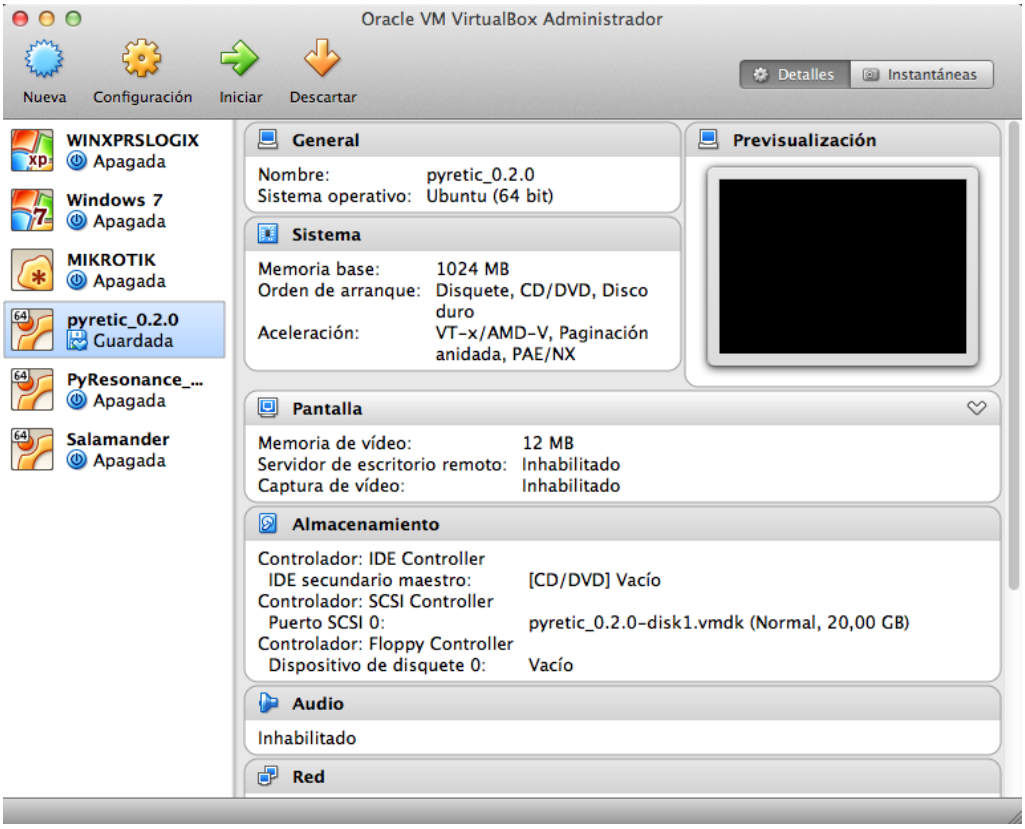


Figura 4. Preferencias del Servicio de Máquina Virtual

3.2. Topologías usadas en la simulación

Como se vimos en el capítulo anterior, Mininet es un conjunto de funciones incorporadas en Linux que se divide en un grupo de "contenedores" más pequeños, cada uno con una asignación fija de potencia de procesamiento. Además Mininet internamente emplea características de virtualización de peso ligero en el kernel de Linux, incluyendo a los grupos

de proceso, CPU, aislamiento de ancho de banda y espacio de nombres de la red. Estas características producen un sistema que se inicia más rápido y para más hosts emulados que utilizan máquinas virtuales completas (Doxygen, 2015).

Una red Mininet consta de los siguientes componentes:

Hosts aislados: Es un grupo de procesos a nivel de usuario que tienen propiedades exclusivas como: interfaces, puertos y tablas de enrutamiento.

Enlaces emulados: Tienen la velocidad de transmisión de datos para cada enlace, cumpliendo con el control de tráfico de Linux que se puede ajustar a una tasa configurada. Cada host emulado tiene su propia interfaz Ethernet virtual que se encuentra conectado a un enlace emulado con otra interfaz virtual o puerto de conmutación virtual; los paquetes enviados a través de una interfaz se entregan a la otra, y cada interfaz aparece como un puerto de Ethernet completamente funcional para todos los sistemas y software de aplicación.

Interruptores emulados: Mininet normalmente utiliza el bridge por defecto de Linux u Open vSwitch y se ejecuta en modo kernel para cambiar paquetes a través de las interfaces por lo que puede modificar con facilidad.

En Mininet con unas pocas líneas de código Python, es posible crear una topología flexible que se puede configurar en función de los parámetros necesarios y que se los puede reutilizar.

3.2.1. Importantes clases, métodos, funciones y variables

Topo: la clase base para las topologías Mininet.

AddSwitch (): añade un interruptor en una topología y devuelve el nombre del interruptor.

AddHost (): añade un host para una topología y devuelve el nombre de host.

AddLink (): añade un enlace bidireccional a una topología. Los enlaces en Mininet son bidireccionales menos que se indique lo contrario.

Mininet: clase principal para crear y administrar una red.

start (): inicia su red.

pingAll (): prueba la conectividad al tratar de tener ping entre todos los nodos.

stop (): detiene la red.

net.hosts: todos los anfitriones en una red.

setLogLevel ('info' | 'debug' | 'output'): ajustar el nivel de salida por defecto de Mininet; 'info' proporciona información útil (Doxygen, 2015).

3.3. Parámetros de rendimiento

Para la medición de rendimiento se recomiendan parámetros como:

- Ancho de banda
- Latencia
- Colas
- TCP estadísticas
- Uso de la CPU

Se es libre de utilizar cualquier herramienta que se esté familiarizado para medir estos diferentes parámetros para lo cual se puede utilizar

programas como Wireshark (analizador de paquetes) y The Dude (gestor de red de Mikrotik).

Mininet proporciona la posibilidad de limitar el rendimiento, además posee características de aislamiento, a través de las clases “CPULimitedHost” y “TCLink”.

Hay que especificar el host y el enlace, también se especifica los parámetros adecuados de la topología.

Además se puede especificar una fracción del total de recursos de la CPU del sistema que se asignará a la máquina virtual con la línea de comando “self.addHost (nombre, cpu = f)”.

También se puede añadir un enlace bidireccional con ancho de banda (Mb/s), retardo (unidades) y pérdida de características (%), con un tamaño máximo de la cola de 1000 paquetes usando un limitador de velocidad jerárquica (paquetes).

Puede que resulte útil crear un diccionario de Python para hacer más fácil para pasar los mismos parámetros en varias llamadas a métodos.

En la rama principal actual de Mininet, sólo tiene que utilizar llaves para recuperar un nodo dado por su nombre.

3.4. Interfaz de línea de comando (CLI)

Mininet incluye una interfaz de línea de comandos (CLI), que puede ser invocado en una red, y proporciona una variedad de comandos útiles, así como la capacidad de mostrar ventanas “xterm” y ejecutar mandatos en nodos individuales en la red. Puede invocar la CLI en una red pasando el objeto de red en el “CLI ()” constructor.

Iniciada la línea de comandos se la puede utilizar para la depuración de la red, ya que permite ver la topología de la red con el comando “net”, realizar pruebas de conectividad con el comando “pingall”, y enviar comandos a los hosts individuales.

3.5. Descripción de la Interfaz Programable de Aplicaciones (API)

Las API de Mininet incluyen clases como “Topo”, “Mininet”, “Host”, “Interruptor”, “Enlace” y sus subclases. Es conveniente dividir estas clases en los niveles o capas, ya que en general, las APIs de alto nivel se construyen utilizando las API de nivel inferior.

API de Mininet está construido en tres niveles principales:

- **API de bajo nivel:** La API de bajo nivel consta de las clases de nodo base y de enlace tales como “Host”, “interruptor” y “Enlace", que en realidad puede crear una instancia de forma individual y se utilizan para crear una red, pero es un poco difícil de manejar.

- **API de nivel medio:** La API de nivel medio añade el objeto “Mininet” que sirve como un contenedor para los nodos y enlaces. Nos ofrece una variedad de métodos como “AddHost ()”, “AddSwitch ()”, y “AddLink ()” para agregar nodos y vínculos a una red, así como la configuración de la red, iniciar y detener particularmente, “start ()” y “stop ()”.

- **API de alto nivel:** La API de alto nivel añade una abstracción de plantilla de topología, la clase “Topo” ofrece la posibilidad de crear plantillas de topología reutilizables y parametrizadas. Estas plantillas se pueden pasar a comando de “mn” a través de la opción “- custom”.

Es valioso entender cada uno de los niveles de la API. En general cuando se desea controlar los nodos y conmutadores directamente, se utiliza la API de bajo nivel. Cuando desee iniciar o detener una red, generalmente utiliza la API de nivel medio en particular la clase “MiniNet”.

Cuando se empieza a pensar en la creación de redes completas se pueden crear usando cualquiera de los niveles de la API, pero por lo general usted querrá elegir la API de nivel medio por ejemplo “Mininet.add * ()”, o la API de alto nivel como es “Topo. añadir * ()” , para crear este tipo de redes.

La API de nivel medio es un poco más simple, ya que no requiere la creación de una clase de topología. Las APIs de bajo nivel y de nivel medio son flexibles y de gran alcance, pero puede ser menos conveniente para la reutilización en comparación con el alto nivel.

Tenga en cuenta también que el alto nivel actualmente no soporta múltiples enlaces entre nodos, pero las API de bajo nivel lo hacen. Con las API de nivel medio y bajo nivel, puede iniciar manualmente los switches si lo desea y pasar la lista adecuada de los controladores para cada switch (Doxygen, 2015).

3.6. Actualización de Mininet

Si tenemos que hacer cambios o adiciones a Mininet para corregir errores u otros problemas, y ha instalado Mininet de la fuente, es posible que desee actualizar su copia de Mininet. Esto se hace fácilmente usando uno de dos métodos:

Actualización mediante enlaces simbólicos a la fuente Mininet, esto hace que sea fácil de actualizar el código python de Mininet:

“cd ~ / MiniNet

git checkout master # suponiendo que usted desea actualizar a la rama principal actual

sudo make desarrollar # esto sólo hay que hacer al principio y cuando cambia mnexec.c

git fetch

git pull – rebase"

Actualizar copiando fuente Mininet en / usr / lib / python, y esto le permite borrar o mover el árbol fuente de Mininet:

"*cd ~ / MiniNet*

git checkout master # suponiendo que usted desea actualizar a la rama principal actual

git fetch

git pull - rebase

sudo make install"

3.7. Creación de topologías en Mininet

Para la creación de topologías por defecto en Mininet primero debemos tener en claro la simbología y lo realizamos mediante la siguiente leyenda:

Hx → Host

Sx → Switch

Cx → Controller

$ → comando en shell

→ comandos como root

Mininet> → comandos dentro de Mininet

$Sudo mn

Inicia Mininet con una topología default con 1 switch, 1 controlador y 2 hosts.

Dentro de Mininet use los comandos nodes y net para ver los nodos y sus enlaces.

Syntaxys:

mininet>[nodo] comando

Colocando el nodo frente a un comando, indicamos que el comando está siendo ejecutado en aquel nodo.

Es posible utilizar el nombre del nodo para sustituir la IP.

mininet>h2 ping -c5 h3

Comandos:

Para salir de Mininet: *mininet>exit*

Para abrir un terminal para el nodo: *mininet>xterm [nodo]*

Para crear o eliminar un o link entre dos nodos: *mininet>link [node1][node2][up or down]*

Para prueba de conectividad entre dos nodos: *mininet>pingall*

Ayuda, muestra lista de comandos de Mininet: *mininet>help*

Para limpiar la topología: *#mn -c*

La creación de topologías es fácil y rápida, pero restringida al momento de pretender hacer una topología personalizada.

A continuación se explicarán 4 tipos de topologías que pueden ser utilizadas en Mininet:

- Single
- Linear
- Tree
- Custom

3.7.1. Single o única:

Consiste de un único conmutador conectado a un número determinado de hosts.

Un switch conectado a N hosts. Se crea con el siguiente comando:

$sudo mn --topo single,N

N es el número de hosts deseados.

La desventaja de esta topología es que el usuario se encuentra amarrado a usar un solo conmutador.

Creación de una topología de 3 host (h1,h2 y h3) conectados a un switch (s1):

mininet@mininet-vm:~$ sudo mn --topo single,3
**** Creating network*
**** Adding controller*
**** Adding hosts:*
h1 h2 h3
**** Adding switches:*
s1
**** Adding links:*
(h1, s1) (h2, s1) (h3, s1)
**** Configuring hosts*
h1 h2 h3
**** Starting controller*
**** Starting 1 switches*
s1

3.7.2. Linear o lineal:

Esta topología consta de un determinado número de conmutadores interconectados de forma lineal.

Cada conmutador tiene un host conectado a él.
Para crear esta topología se usa el siguiente comando:
$sudo mn –topo linear,N

En donde N es el número de conmutadores y hosts que se desean añadir a la topología.

Esta topología permite un manejo más flexible en cuanto a la adición de conmutadores, pero la limitante es que el número de conmutadores debe ser igual al número de hosts, ya que necesariamente cada host se conecta a un conmutador. Limitando el uso de esta topología en las topologías personalizadas.

$sudo mn --topo linear,N

N será el número de switches y hosts.

Creación de una topología lineal de 4 host y 4 switches:

mininet@mininet-vm:~$ sudo mn --topo linear,4
*** *Creating network*
*** *Adding controller*
*** *Adding hosts:*
h1 h2 h3 h4
*** *Adding switches:*
s1 s2 s3 s4
*** *Adding links:*
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (s1, s2) (s2, s3) (s3, s4)
*** *Configuring hosts*
h1 h2 h3 h4
*** *Starting controller*
*** *Starting 4 switches*
s1 s2 s3 s4
*** *Starting CLI:*
mininet>

3.7.3. Tree o árbol:

En este caso es posible crear una topología en forma de árbol. El comando para la creación de esta topología es:

$sudo mn –topo tree,depth=N,fanout=M

N que es el nivel de profundidad del árbol que se desea tener.

M es un parámetro denominado esparcimiento.

Si los parámetros M y N son iguales a dos, se tendrán dos niveles, el primero con un conmutador y el segundo con dos, y conectados a cada conmutador del nivel dos, se tendrán dos hosts.

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

Crea una topología de árbol con profundidad N y anchura M.

Ejemplo de creación de una topología en forma de árbol de 2 de profundidad y 2 de ancho:

mininet@mininet-vm:~$ sudo mn --topo tree,depth=2,fanout=2
**** Creating network*
**** Adding controller*
**** Adding hosts:*
h1 h2 h3 h4
**** Adding switches:*
s1 s2 s3
**** Adding links:*
(h1, s2) (h2, s2) (h3, s3) (h4, s3) (s1, s2) (s1, s3)
**** Configuring hosts*
h1 h2 h3 h4
**** Starting controller*
**** Starting 3 switches*
s1 s2 s3
**** Starting CLI:*
mininet>

3.7.4. Custom o personalizado

Para topologías personalizadas es necesario crear un archivo en python con su topología (Doxygen, 2015).

$sudo mn --custom mytopo.py --topo mytopo

Las topologías customizadas se quedan en ./mininet/custom

Creación de topologías personalizadas en Mininet.

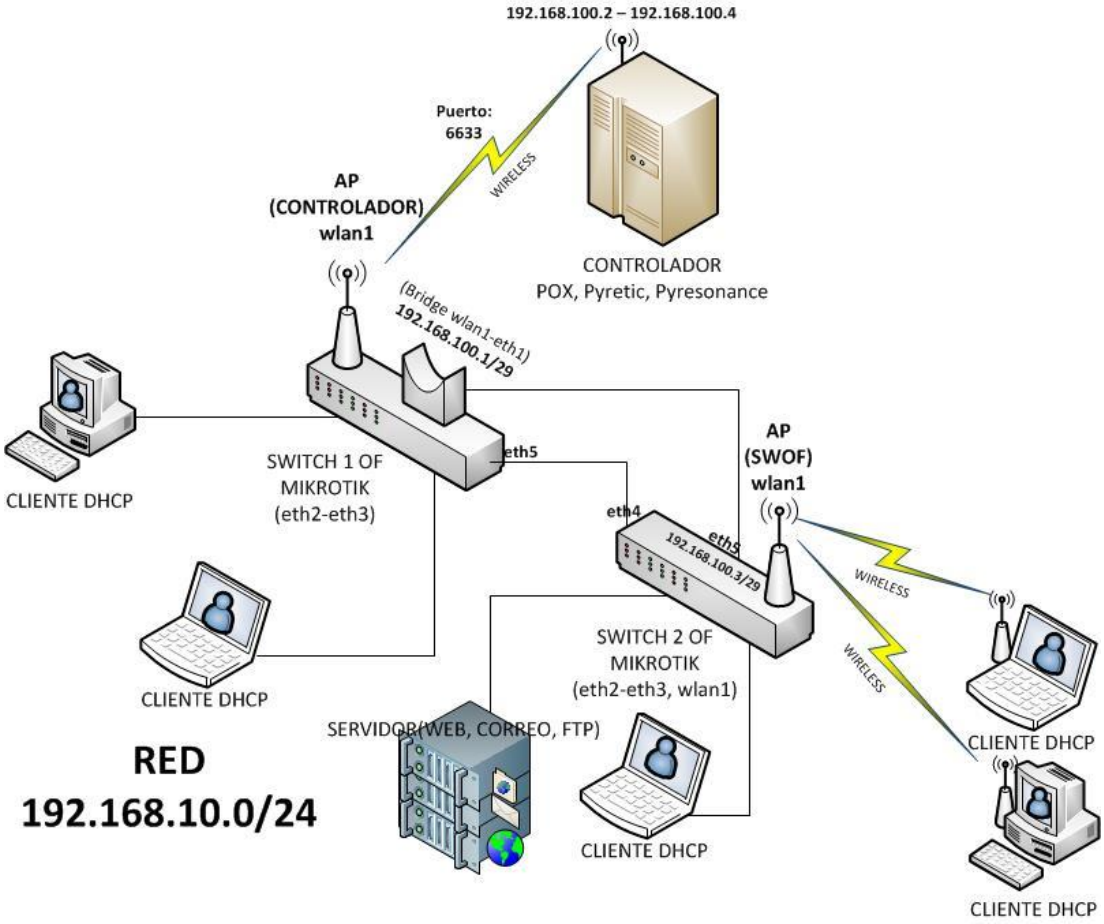


Figura 5. Topología usada para la simulación de la red.

Para ejecutar un archivo con una topología personalizada, nos dirigimos a la ubicación del mismo y ejecutamos el siguiente comando:

sudo python nombre_de_archivo.py

En este caso se utilizó el archivo topologiaSDN.py ubicada en “/mininet” de la máquina virtual utilizada y en el código se activa un controlador remoto que se lo debe inicializar.

Ejecución del archivo que contiene la topología del presente proyecto:

mininet@mininet-vm:~$ cd mininet

mininet@mininet-vm:~/mininet$ sudo python topologiaSDN.py
Registro de las conexiones de los hosts
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s2-eth1
h4 h4-eth0:s2-eth2
mininet>

El controlador que está activo es POX y se están ejecutando dos módulos, uno que es el módulo learning_switch, que permite que el controlador actué como un switch capa 2, junto con un módulo de DHCP, que permite la asignación de parámetros de red a los hosts.

Comando para inicializar el controlador:

mininet@mininet-vm:~$ pox.py forwarding.l2_learning
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.

Si no inicializamos el controlador se despliega el siguiente mensaje:

Unable to contact the remote controller at 127.0.0.1:6633
Código del archivo topologiaSDN.py:

```
from mininet.net import Mininet
from mininet.cli import CLI
from mininet.node import RemoteController
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel
net = Mininet(controller=None)
net.addController('c0', controller=RemoteController, ip='127.0.0.1', port=6633)
# Creación de nodos
```

```
h1 = net.addHost('h1')
h2 = net.addHost('h2')
h3 = net.addHost('h3')
h4 = net.addHost('h4')
s1 = net.addSwitch('s1')
s2 = net.addSwitch('s2')
# Creación de enlaces
net.addLink(h1, s1)
net.addLink(h2, s1)
net.addLink(h3, s2)
net.addLink(h4, s2)
net.addLink(s1, s2)
# Inicio de funcionamiento
net.start()
# Configuración de clientes dhcp
h1 = net.get('h1')
h1.cmd('dhclient')
h2 = net.get('h2')
h2.cmd('dhclient')
h3 = net.get('h3')
h3.cmd('dhclient')
h4 = net.get('h4')
h4.cmd('dhclient')
print "Registro de las conexiones de los hosts"
dumpNodeConnections(net.hosts)
CLI(net)
net.stop()
```

CAPITULO 4: IMPLEMENTACIÓN DE LA RED

4.1. Topología

En el presente proyecto se utiliza una topología de dos conmutadores OpenFlow conectados entre sí, y estos a su vez a un servidor controlador mediante un bridge entre la interfaz inalámbrica del switch1 (Wlan1) con una interfaz Ethernet 1(eth1) del mismo dispositivo y este a su vez está conectado a la interfaz Ethernet 5 (eth5) del Switch 2 obteniendo conectividad con el controlador.

Cada conmutador tiene conectados hosts; en el primer conmutador alámbricamente y en el segundo conmutador se agregó una interfaz inalámbrica con la cual se encuentra abierta la posibilidad de que se conecten más hosts a la red SDN.

Para la conexión entre los dos switches se utilizó las interfaces Ethernet 5 del primero y Ethernet 4 del segundo conmutador.

En cada uno de los conmutadores se reservó un puerto para poder configurarlos mediante el programa Winbox (Ver anexo 3), en el caso del primero es el puerto Ethernet 4 y en el segundo el puerto Ethernet 1.

Tabla 1. ***Tabla de Direccionamiento***

Dispositivo	Interfaz	Dirección IP	Máscara	Red
SW1	WLAN1	192.168.100.1	255.255.255.248	BRIDGE
	ETH1			
	ETH2	DHCP	DHCP	ETHERNET LAN
	ETH3			
	ETH5			
	ETH4	-	-	WINBOX
SW2	ETH5	192.168.100.3	255.255.255.248	LAN
	ETH2	DHCP	DHCP	ETHERNET LAN
	ETH3			
	ETH4			
	WLAN1	DHCP	DHCP	WLAN
	ETH1	-	-	WINBOX
CONTROLADOR POX	ETH1	192.168.100.2	255.255.255.248	ETHERNET
CONTROLADOR PYRETIC	ETH1	192.168.100.4	255.255.255.248	ETHERNET
CONTROLADOR PYRESONANCE	ETH1	192.168.100.5	255.255.255.248	ETHERNET

4.2. Plataforma de hardware

4.2.1. Servidor Controlador

El servidor controlador se implementa en un computador MacBookPro que cuenta con las siguientes características:

- Procesador 2.9 GHz Intel Core i7
- Memoria RAM de 8GB.?
- Disco duro de 750 GB.

- Dos adaptadores de red, uno alámbrico y otro inalámbrico.
- Sistema Operativo OS X 10.9.5 Mavericks

Motor de máquinas virtuales Virtual Box versión 4.3.12 que permite que la máquina física actúe como si se tuviese varios computadores independientes, para ejecutar varios sistemas operativos al mismo tiempo en la misma máquina física, compartiendo recursos como el procesador o la memoria de la máquina física.

El servidor controlador se ejecuta en una máquina virtual que contiene tres controladores instalados POX, Pyretic y PyResonance. Esta máquina virtual tiene tres adaptadores de red, el primero que está configurado con NAT para el acceso a internet el cual es utilizado para actualizaciones de la máquina virtual, el segundo sirve para conectarse con la consola de la máquina real y el tercero está configurado como un adaptador bridge el cual nos permite conectarnos con el switch.

4.2.2. Conmutadores.

En la topología personalizada del presente proyecto se utilizó dos router Mikrotik RB951Ui-2HnD con las siguientes especificaciones técnicas:

Router con un CPU Atheros de nueva generación.

Cuenta con:

- Cinco puertos Ethernet
- Un puerto inalámbrico WIFI de 2,4 GHz de potencia 1 W 802.11b/g/n
- RouterOS como sistema operativo basado en un Kernel de Linux
- 60 MB en disco

- 128 MB de memoria RAM
- La función de entrada/salida PoE en un puerto.

Figura 6. Router Mikrotik RB951Ui-2HnD

El router no cuenta en su configuración de fábrica con el paquete Openflow pero si se lo puede colocar dentro de sus capacidades, por lo que es necesario instalarlo para la realización del presente proyecto de grado (Anexo 2) (InkaLinux, 2015).

4.2.2.1. Configuración de los switches OpenFlows

Para poder tener acceso a la configuración de los conmutadores necesitamos dejar libre una interfaz, la cual vamos a acceder mediante el programa Winbox de la marca Mikrotik (Anexo 1).

Dentro de la interfaz gráfica del programa Winbox conectado al conmutador número uno nos dirigimos al menú principal y escogemos la opción Bridge con la cual vamos a conectar la interfaz Wlan1 con la Ether1 para que a esta se pueda conectar mediante una dirección de red el conmutador 2 con la Ether5 y estos con el controlador.

Después de haber colocado el nombre al nuevo Bridge debemos agregar una dirección IP para la nueva interfaz para que pueda ser parte de la red entre los dos conmutadores y el controlador (InkaLinux, 2015).

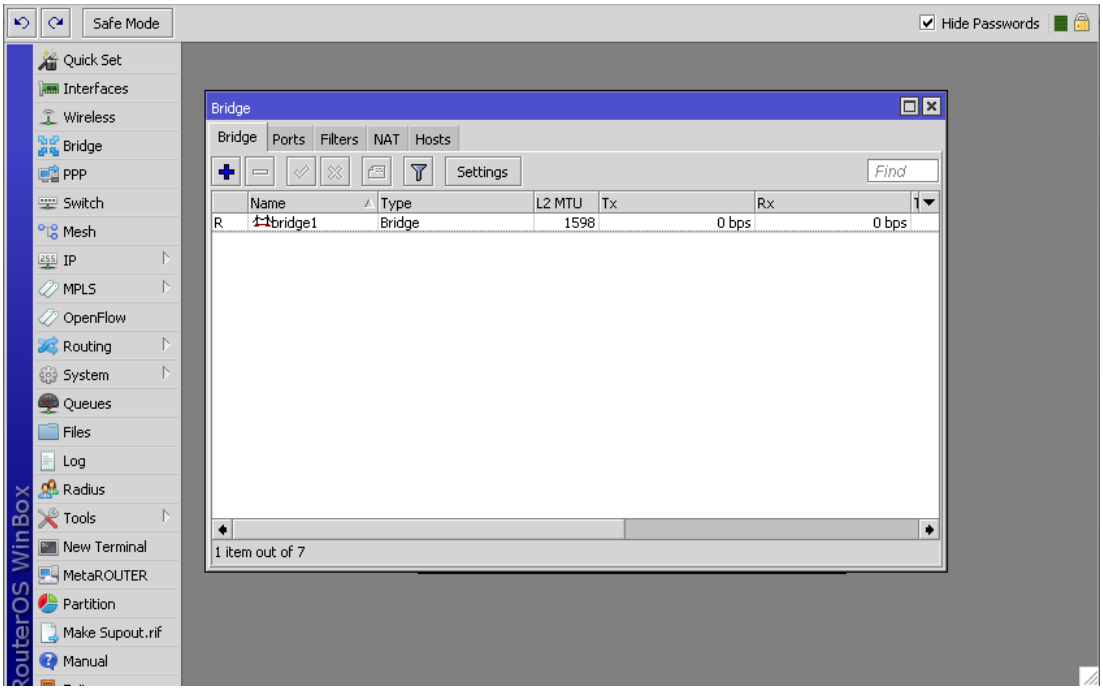


Figura 7. Configuración de Bridge entre conmutadores y Controlador

Se procede a dirigirse al submenú a la opción ports donde mediante la utilización del botón azul con el signo “+”, asignamos los puertos que necesitamos que pertenezcan al bridge, en el caso del primer conmutador es el puerto Wlan1 y Ether1, ya que el controlador estará conectado mediante red inalámbrica al conmutador 1 y en el caso con el conmutador 2 se conectará con la interfaz Ether 5.

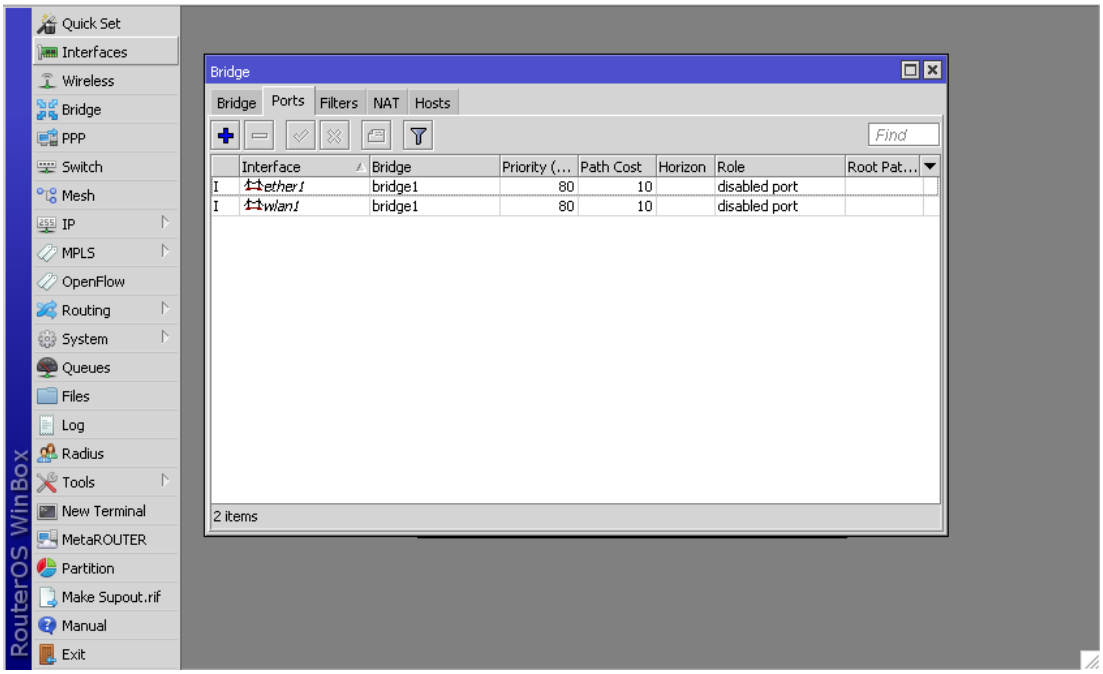


Figura 8. Asignación de puertos pertenecientes al Bridge en el SW1

Para la asignación de direcciones a las distintas interfaces nos dirigimos dentro del menú principal del programa Winbox a la opción IP y del menú que se despliega a continuación seleccionamos Address para poder asignar la dirección IP a la interfaz Bridge1 que se creó con anterioridad, en este caso se coloca la dirección 192.168.100.2/29 perteneciente a la red 192.168.100.0/29 ya que el controlador tiene la dirección 192.168.100.1/29, adicional podremos asignar la dirección 192.168.100.3/29 al puerto Ether5 del conmutador número dos para q pueda comunicarse dentro de esta red.

Posterior a esto se puede colocar algún comentario como guía o aclaración y para aceptar presionamos en la opción Aplicar y OK.

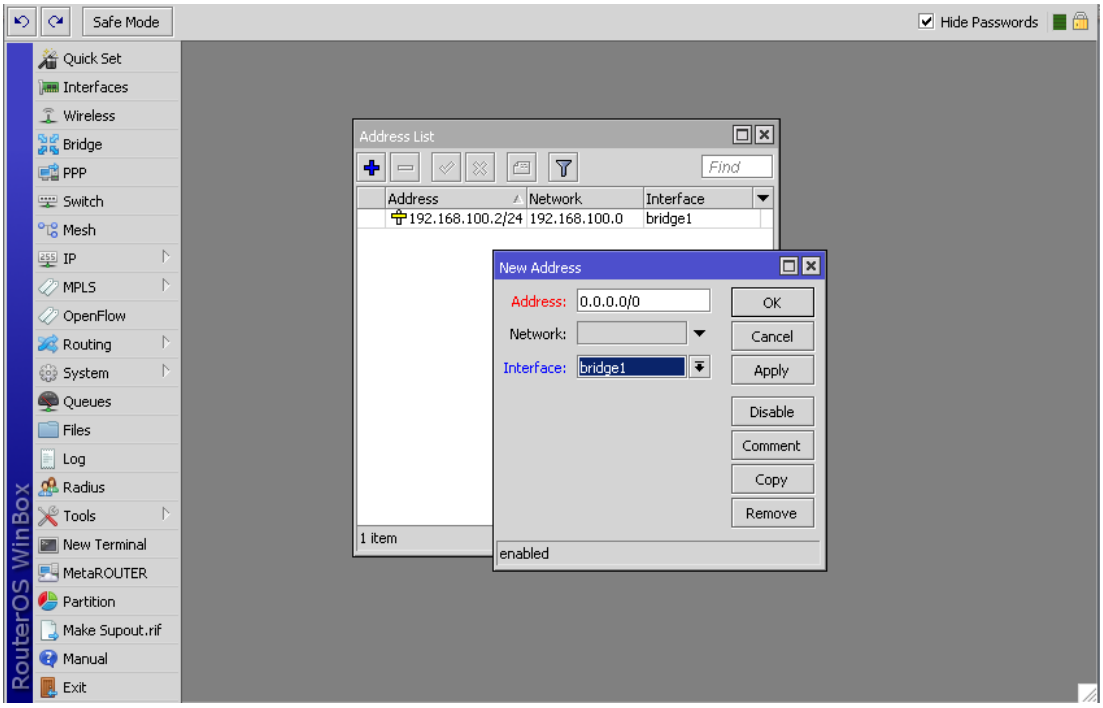


Figura 9. Asignación de la dirección del Bridge

En el menú principal de la interfaz gráfica de Winbox encontramos la opción OpenFlow y al ingresar en él se despliega una pantalla con las opciones de Switches, Flows y Ports.

En la opción Switches mediante el botón azul del signo “+”, procedemos a crear un oflow al que se le asigna un nombre, un puerto de escucha, ID con la dirección MAC e indica la dirección del controlador, y este es el que va a señalar cuales puertos del dispositivo serán OpenFlow.

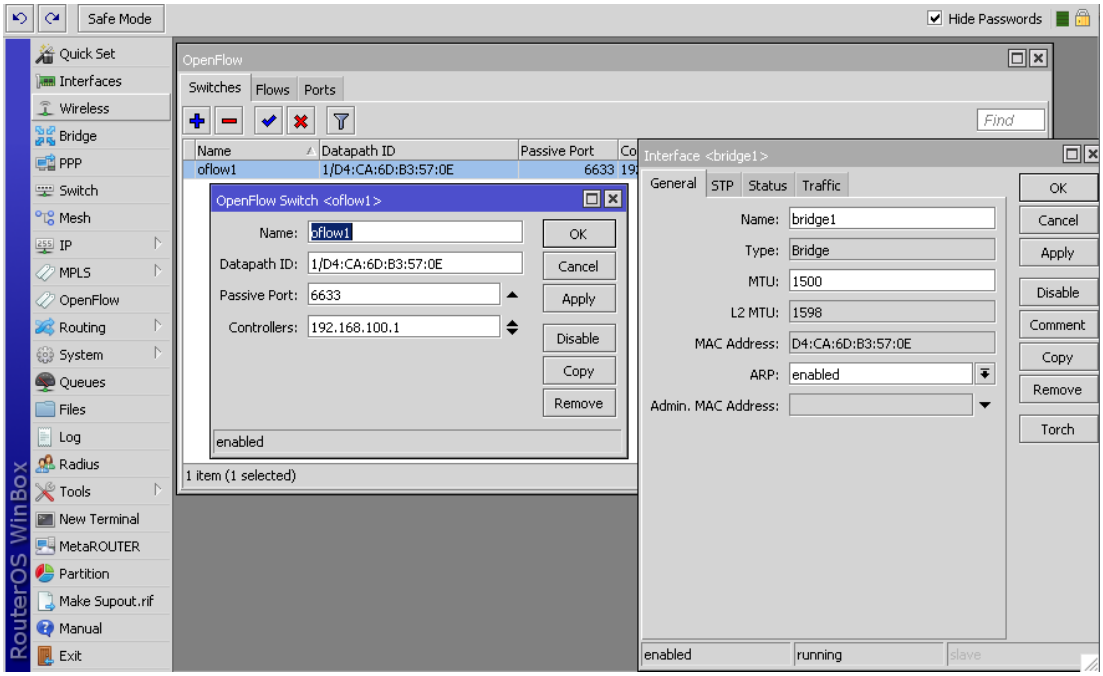


Figura 10. Creación de “oflow1”

Una vez creado el oflow se procede a ir a la opción Ports donde se asignara los puertos que serán OpenFlow, para lo cual hay que indicar el Switch e interfaces comprometidas, como se indicó anteriormente no se tomará en cuenta para esto la interfaz Ether4 del conmutador 1 ni la interfaz Ether 1 del conmutador 2.

En el caso del conmutador 1 se verán comprometidas las interfaces Ether2, Ether3 y Ether5 (InkaLinux, 2015).

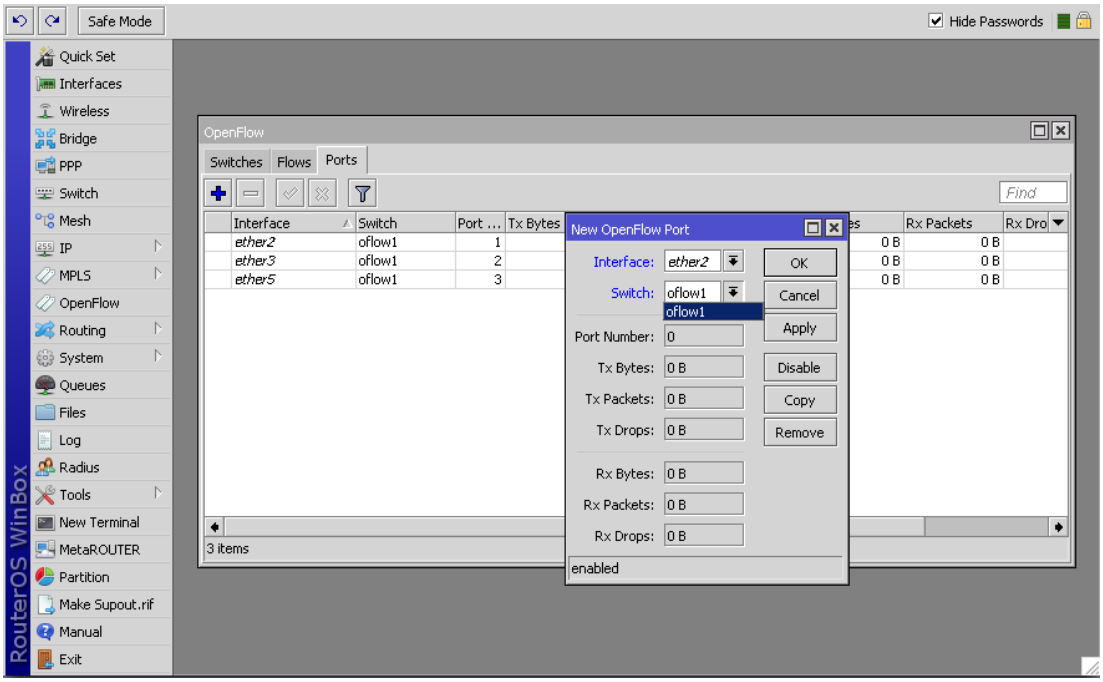


Figura 11. Asignación de los puertos q serán OpenFlow en el dispositivo

4.2.3. Hosts.

La topología de red SDN fue probada con tres hosts conectados a los conmutadores OpenFlow, pero se tomó en cuenta la escalabilidad de la red tanto alámbrica como inalámbricamente por lo que fue diseñada para que en el conmutador 2 se encuentre un AP (Access Point) con el cual se asegura la posibilidad de crecimiento de red.

Los host probados tienen un sistema operativo Windows 7 y estos a su vez poseen máquinas virtuales que emulan más host que se conectan a la red.

4.3. Software del servidor controlador.

Existen varios controladores SDN que se diferencian comúnmente por el lenguaje de programación con el que trabajan, en este proyecto se analizarán tres tipos de controladores que trabajan con python y que se

instalarán en máquinas virtuales con Linux como sistema operativo, los controladores son: POX, Pyretic y PyResonance (Kinetic, en su última versión).

La máquina virtual que se utilizó es la misma tanto para POX como para Pyretic y podemos bajarla de la siguiente página: http://www.frenetic-lang.org/pyretic/. En esta máquina virtual vienen ya preinstalados los dos controladores, y viene lista para ejecutarla en cualquier motor de máquinas virtuales, en este caso VirtualBox. Para el caso de PyResonance o que en su última versión se lo llama Kinetic, podemos bajar la máquina virtual con el controlador instalado de la siguiente página: http://resonance.noise.gatech.edu/. Las máquinas virtuales funcionan mucho mejor cuando se utiliza la versión de la misma plataforma del sistema operativo en el que estemos trabajando, ya sea para 32 o 64 bits (Frenetic - lang, 2011).

4.3.1. POX

Es una plataforma de software escrita en python, que tiene módulos o componentes escritos en el mismo lenguaje, para ejecutar POX basta con tener instalado python e iniciar el componente principal pox.py, el cuál se encargará de reconocer el nombre de todos los módulos a ejecutar y realiza la llamada de cada uno de ellos, para que el controlador POX finalmente se ejecute correctamente.

Los módulos son localizados en todos los directorios que constan en la carpeta en la que se ha instalado POX, por lo tanto si queremos ejecutar un módulo tenemos que especificar a qué carpeta pertenece, y en el caso de que el módulo se encuentre en el directorio raíz de POX, no será necesario especificar dicha carpeta, por ejemplo:

• El módulo ejemplo se encuentra en el directorio raíz de POX, entonces se utilizará el siguiente comando:

$ pox.py ejemplo

• El módulo l2_learning se encuentra en la carpeta forwarding, por lo tanto se utilizará el siguiente comando:

$ pox.py forwarding.l2_learning

Si queremos ejecutar los dos módulos simultáneamente, se establece lo siguiente:

$ pox.py ejemplo forwarding.l2_learning

Además del componente pox.py, existe uno llamado of_01.py el cuál se encuentra en la carpeta OpenFlow. Este componente se encarga de que el controlador se ejecute con la versión 1.0 de OpenFlow, para lograr una comunicación con los dispositivos (switches) con la misma versión, especificando también el puerto de escucha y la dirección IP asociada al controlador (NOXRepo, 2008).

Dependiendo de cómo estén estructurados los módulos, pueden tener ciertas opciones o parámetros y éstos representan valores a ciertas variables que se tenga en el código. Algunos módulos se inician con valores por defecto, pero éstos pueden ser personalizados poniendo la opción que se quiera cambiar después del nombre del módulo que se vaya a ejecutar.

Estas opciones se definen en una función llamada launch o de lanzamiento, la cuál nos permite pasar el valor de las opciones a sus respectivos módulos, e iniciar los mismos. Se puede declarar una función launch sin valores por defecto, en ese caso los valores de las variables se deberían especificar dentro del código o se deberían dar obligatoriamente valores a las opciones de un módulo.

El componente of_01 tiene dos parámetros u opciones importantes y son:

Tabla 2. ***Tabla de Parámetros Importantes***

Opciones	**Valor por defecto**	**Descripción**
--port	6633	Especifica el puerto de escucha del controlador
--address	Todas las direcciones	Especifica las direcciones IP de las interfaces que actúan en el controlador

Por ejemplo si queremos ejecutar el módulo of_01:

- Con valores por defecto, se establece lo siguiente:

$ pox.py openflow.of_01

- Con valores distintos, se añade los parámetros y sus valores después del módulo:

$ pox.py openflow.of_01 --port=6644 --address=192.168.100.1

Para poder realizar un módulo personalizado es importante conocer algunos módulos y APIs preestablecidos, que se utilizan por ejemplo para el manejo de direcciones IPv4, IPv6 y Ethernet, para el envío y recepción de paquetes de los protocolos como DHCP, IPv4, ICMP, TCP, etc., para controlar eventos que existen en una clase o una superclase, para el manejo de los DPIDs, o bien para el manejo de los logs (mensajes de información). Los DPIDs nos identifican a un switch OpenFlow como único en la red SDN.

A más de conocer ciertos módulos que los debemos importar en nuestro código según sea necesario, debemos conocer también funciones importantes como la función launch, que se explicó anteriormente, la función ConnectionUp, que es la que permite enviar mensajes a los swtiches para establecer una conexión con el mismo, y según su uso podemos instalar

paquetes especificando que protocolo se va a utilizar, o la función PacketIn que se activa cuando el controlador recibe un mensaje Openflow desde el switch y maneja ciertos eventos para indicar que un paquete ha llegado o no a un puerto de un switch. En fin existen una gran cantidad de componentes preestablecidos y de APIs que nos ayudarán a crear módulos propios. En la página https://openflow.stanford.edu/display/ONL/POX+Wiki, se encuentra toda la información y algunos ejemplos para realizar un módulo personalizado (McCauley, 2014).

Por ejemplo si queremos realizar un módulo, el cuál permita que un switch de la red sea despreciado y que los demás switches utilicen un cierto módulo, en este caso el módulo l2_learning, el código es el siguiente:

```
from pox.core import core
# Librería que importa el objeto core de POX para establecer una conexión
from pox.lib.util import str_to_dpid
# Librería que permite el manejo de los DPIDs de cada switch
from pox.forwarding.l2_learning import LearningSwitch
# Importamos la clase LearningSwitch del módulo l2_learning
def launch (idswitch):
  """
  Se define la función launch directamente ya que no se usan otras clases
  idswitch es un parámetro al que debemos darle un valor en la línea de comandos
  """
  idswitch = str_to_dpid(idswitch) # Se invoca el dpid de un switch
  def _handle_ConnectionUp (event):
   # Se maneja un evento donde se identifica a todos los switches
    if event.dpid != idswitch:
      LearningSwitch(event.connection, True)
```

```
        # Se establece la conexión con el modulo learning switch
      if event.dpid == idswitch:
        core.getLogger().info("Switch Despreciado: %s" %
        (event.connection,)) # Mensaje de información
    core.openflow.addListenerByName("ConnectionUp",
_handle_ConnectionUp)
```

A este módulo lo llamaremos seleccionar, lo ubicaremos en la carpeta principal de POX, vamos a pasarle la dirección MAC de un switch como parámetro, y el módulo l2_learning sólo funcionará en los demás switches de la red. Para ejecutar éste módulo se usa el siguiente comando:

```
$ pox.py seleccionar --idswitch=00-00-00-00-00-04
```

En la figura 12 se puede apreciar la ejecución de POX con el módulo personalizado seleccionar.

```
mininet@mininet-vm:~$ pox.py seleccionar --idswitch=00-00-00-00-00-04
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[00-00-00-00-00-02 3] connected
INFO:openflow.of_01:[00-00-00-00-00-03 4] connected
INFO:openflow.of_01:[00-00-00-00-00-04 1] connected
INFO:seleccionar:Switch Despreciado: [00-00-00-00-00-04 1]
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
```

Figura 12. Ejecución del controlador POX con el módulo seleccionar

La topología usada en Mininet es una tipo lineal de cuatro switches conectados entre sí, cada uno tiene un host conectado, y se usa a POX como controlador. Esta topología se la puede apreciar en la figura 13. Por otro lado el host del switch seleccionado no podrá tener conectividad con ningún otro host de la red, mientras que los demás hosts si tendrán conectividad, esto lo podemos probar al ejecutar el comando pingall dentro de Mininet (McCauley, 2014).

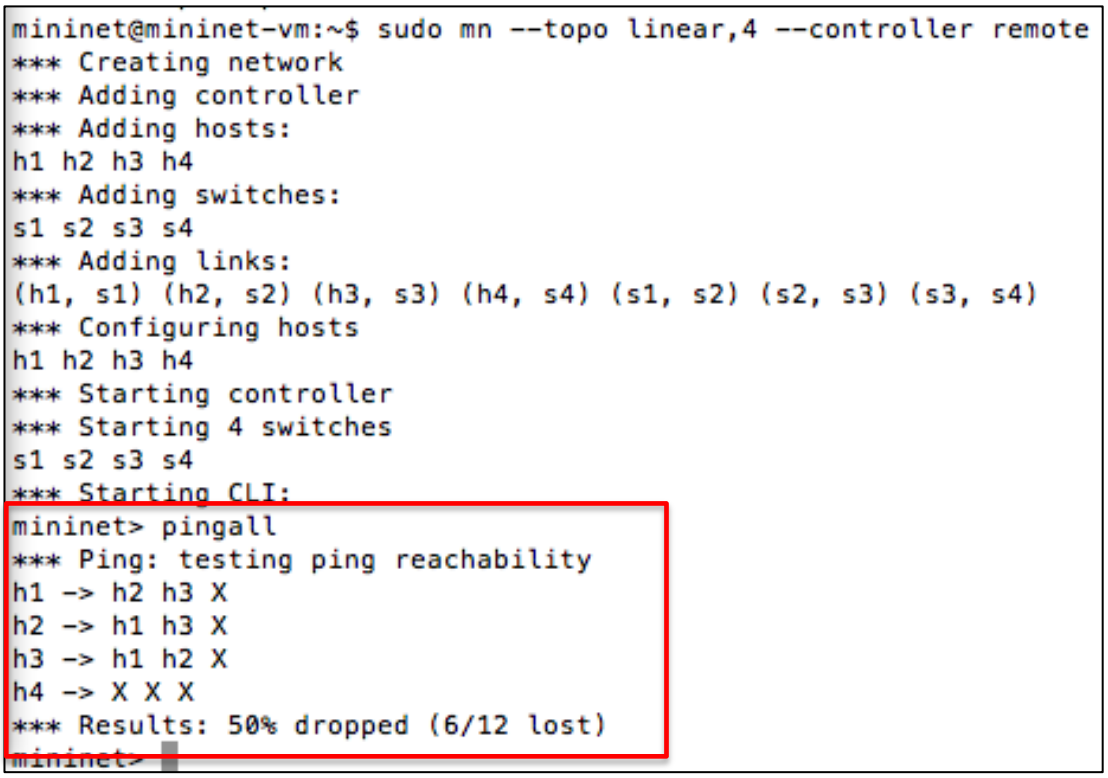


Figura 13. Topología creada en mininet, que usa el controlador POX

4.3.2. Pyretic

Al igual que POX este controlador está implementado en python, tiene módulos escritos en el mismo lenguaje, para ejecutarlo basta con tener instalado python e iniciar el componente principal pyretic.py, el cuál se encargará de reconocer el nombre de todos los módulos a ejecutar y realiza la llamada de cada uno de ellos, para que el controlador Pyretic finalmente se ejecute correctamente.

Los módulos son localizados de la misma forma que en el controlador POX, por lo tanto si queremos ejecutar un módulo tenemos que especificar a qué carpeta de Pyretic pertenece, por ejemplo:

El módulo mac_learner se encuentra en el directorio pyretic/modules, por lo tanto se utilizará el siguiente comando:

$ pyretic.py pyretic.modules.mac_learner

Al igual que en el controlador POX, dependiendo de cómo estén estructurados los módulos, pueden tener ciertas opciones o parámetros. Estos parámetros se definen en una función llamada main dentro del código.

Cada módulo de Pyretic debe tener la función llamada main, la cual se asimila a la función launch en POX. En esta función debemos devolver una instancia de una o varias de las clases existentes en las llamadas políticas de Pyretic (GitHub, 2014).

Una política en Pyretic, es una función que toma un paquete como entrada y devuelve un conjunto de paquetes. Ésta función indica que es lo que un switch debe hacer con los paquetes entrantes. En fin todas las políticas existentes en Pyretic tienen un objetivo específico, por ejemplo el manejo de un switch o de los puertos de un switch, el manejo de direcciones IP y direcciones MAC, el control de flujo entre los switches de la red, etc.

En la tabla 3, se puede apreciar un resumen de las políticas básicas existentes en Pyretic.

Tabla 3. ***Tabla de Políticas básicas en Pyretic***

Política	Sintaxis	Descripción	Ejemplo
match	match(f=v)	Si f es igual a v, se permite el intercambio de paquetes, según se establezca la condición	match(switch=4)
Drop	drop	Retorna un conjunto de paquetes vacíos	drop
identity	identity	Retorna el paquete original	identity
modify	modify(f=v)	Permite el flujo de paquetes donde el valor del campo f se ajusta al valor v	modify(dstip='2 .2.2.8')
forward	fwd(a)	Permite el flujo de paquetes donde el valor de un puerto de salida es a. Equivale a establecer: modify(port=a)	fwd(1) modify(port=1)
flood	flood()	Se permite el flujo de paquetes por cada puerto local en el switch	flood()
Composición Paralela	A + B	Retorna la unión de la salida de A con la salida de B	fwd(1) + fwd(2)
Composición Secuencial	A >> B	Retorna la salida de B donde la salida de A es la entrada de B.	fwd(1) >> fwd(2)
Negación	~ A	Retorna la negación lógica de una política establecida en este caso A	~match(switch=1)

Para realizar un módulo personalizado, se debe por lo menos conocer el funcionamiento de las políticas básicas de Pyretic, anteriormente mencionadas. Estas políticas nos simplifican el código, dependiendo de lo que queramos realizar, por ejemplo: en el controlador POX se necesita programar muchas líneas de código, para realizar un módulo que funcione como un componente hub, a diferencia que en Pyretic para realizar el mismo componente simplemente se agrega la política flood, permitiendo el flujo de paquetes en todos los puertos, tal y como funciona un hub, así:

```
from pyretic.lib.corelib import * # Importamos la librería principal
def main():

 return flood() # Se utiliza la política flood
```

Por ejemplo si ahora queremos crear un módulo parecido al que realizamos con el controlador POX, el cual permita seleccionar un switch y despreciarlo, podemos escribir el siguiente código:

```
from pyretic.lib.corelib import * # Importamos la librería principal
from pyretic.lib.std import * # Librería para el manejo de políticas
from pyretic.modules.mac_learner import mac_learner
# Importamos la función mac_learner del módulo mac_learner
def main (idswitch):
 print "Switch OpenFlow %s Despreciado" % idswitch # Mensaje
 '''
```

Utilizamos la política match negada para que solo en el switch seleccionado no se permita el flujo de paquetes y se devuelva paquetes nulos

```
 '''
 return ~match(switch=int(str(idswitch))) >> mac_learner()
```

A este módulo lo llamaremos “seleccionar”, lo ubicaremos en el directorio pyretic/pyretic, vamos a pasarle el identificador de un switch como parámetro, y el módulo mac_learner sólo funcionará en los demás switches de la red y no en el seleccionado. Para ejecutar éste módulo se usa el siguiente comando:

$ piretic.py pyretic.seleccionar --idswitch=4

En la figura 14 se puede apreciar la ejecución de Pyretic con el módulo personalizado seleccionar.

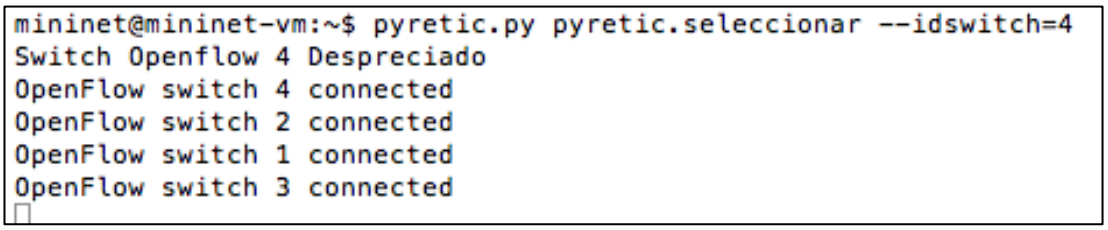


Figura 14. Ejecución del controlador Pyretic con el módulo seleccionar

La topología usada es la misma que se utilizó anteriormente con el controlador POX, pero en este caso se usa a Pyretic como controlador y se la puede apreciar en la figura 15. El host del switch seleccionado no podrá tener conectividad con ningún otro host de la red, mientras que los demás hosts si tendrán conectividad.

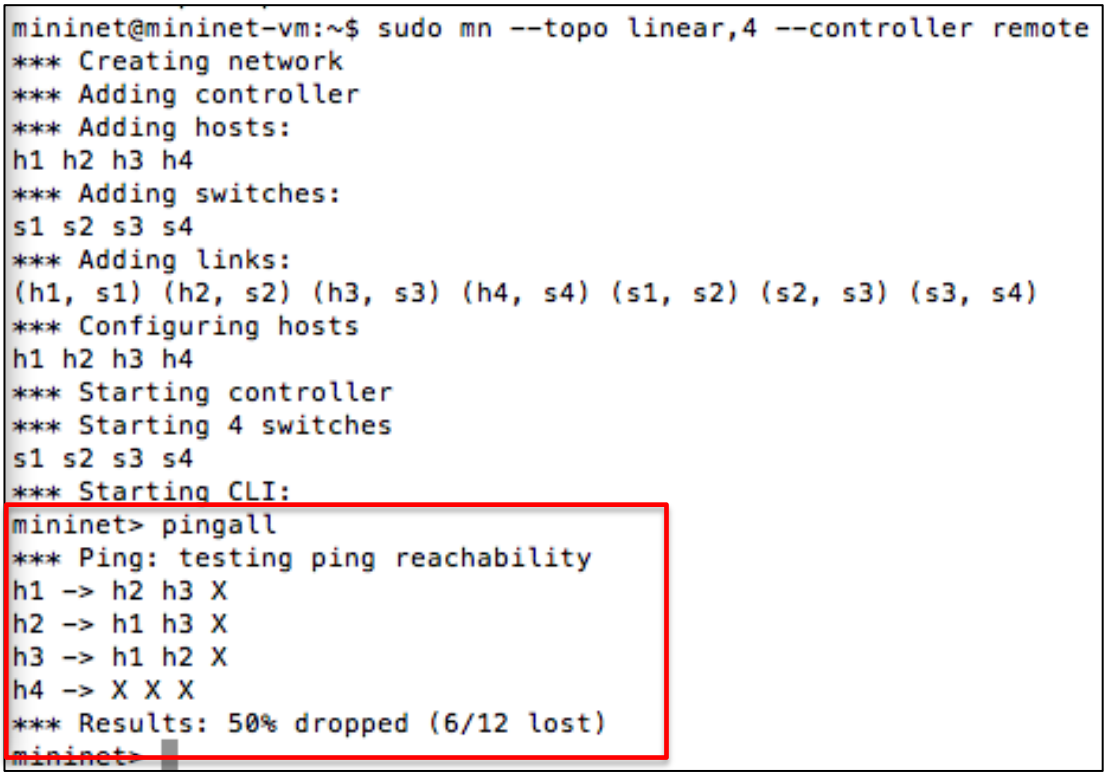


Figura 15. Topología creada en Mininet, que usa el controlador Pyretic

4.3.3. PyResonance

Este controlador esta implementado con Pyretic, por lo tanto para ser ejecutado tenemos que tener instalado tanto el controlador Pyretic como python, y lo iniciamos con el componente pyretic.py, que es el mismo que se utiliza para el controlador Pyretic.

Tal como se mencionó anteriormente se puede descargar la máquina virtual que viene incluida con PyResonance, o también puede ser instalado en la misma máquina virtual de Pyretic.

Los módulos son localizados de la misma forma que en POX y en Pyretic, por lo tanto si queremos ejecutar un módulo tenemos que especificar a qué carpeta pertenece, por ejemplo:

El módulo ids se encuentra en el directorio pyretic/pyresonance/apps, por lo tanto para ejecutar el módulo se utiliza el siguiente comando:

$ pyretic.py pyretic.pyresonance.apps.ids

Al igual que en POX y Pyretic, dependiendo de cómo estén estructurados los módulos, pueden tener ciertas opciones o parámetros. Estos parámetros se definen en la función main al igual que en Pyretic.

PyResonance permite un control dinámico en la red SDN por eventos, es decir este controlador reacciona a diferentes tipos de acontecimientos que se presentan en la red, cambiando la política de red aplicada dinámicamente. Los acontecimientos pueden ser por ejemplo, la detección de intrusos, cuando un host alcanza un límite de ancho de banda, cuando se autentica un host en la red, etc. (GitHub, 2013).

Un módulo personalizado de PyResonance se implementa igual que un módulo de Pyretic, donde se puede construir múltiples políticas de red y componerlas entre sí secuencial o paralelamente, expresando una política general para la red SDN. En los módulos de PyResonance se establece un conjunto de estados y un conjunto de transiciones entre ellos, los mismos que se asignan a algún comportamiento de la red que es reconocido mediante una política de Pyretic.

Un módulo en PyResonance debe tener la siguiente estructura:

• La función LPEC, donde un LPEC se refiere a un conjunto de paquetes que se quieran recibir o enviar. En el código se puede representar un LPEC usando cualquier filtro Pyretic. Por ejemplo, el LPEC cuya fuente de la dirección IP sea la 10.0.0.1 se puede expresar simplemente como:

match(srcip = IPAddr ('10 .0.0.1 '))

Donde la función LPEC para el ejemplo se programaría de la siguiente manera:

```
def lpec(f):
return match(srcip=f['srcip'])
```

• Las funciones de transición que son aquellas que asignan un valor que una variable debe tomar cuando llega un evento en particular en el controlador. Por ejemplo, si tenemos una función de transición llamada infected que codifica un solo caso que es cuando se produce un evento en el cual un host está infectado, el nuevo valor de este evento es tomado por la variable de infected. La función de transición infected estaría programada de la siguiente manera:

```
def infected(event):
return event
```

• La máquina de estados finita (FSM) que es la que asocia las funciones de transición que definimos con las variables de estado correspondientes. En la FSM se definen los valores de cada variable, cada definición de una variable simplemente especifica el tipo de la variable (es decir, un conjunto de valores permitidos), el valor inicial, y las funciones de transición asociadas. Por ejemplo la variable de la función de transición infected tiene un valor lógico booleano que es falso inicialmente, el cual representa el supuesto caso de que todos los host en la red no están infectados. La programación para la FSM sería la siguiente:

```
def __init__(self):
        self.fsm_description = { 'infected' : (bool, False, NextFns(event_fn=infected))}
```

• Las políticas y los eventos. Pyretic automáticamente dirige cada evento de entrada a la adecuada LPEC FSM, donde estará a cargo de la

función de transición que se especifica en la definición de la FSM (por ejemplo, fsm_desccription) (GitHub, 2013).

Con la estructura anteriormente mencionada, podemos escribir el código para un módulo de PyResonance, donde un switch de la red SDN pase a un estado de infectado y por lo tanto el controlador no permita el flujo de paquetes en ese switch en específico, es decir lo desprecie porque está infectado.

El código escrito para el controlador PyResonance es el siguiente:

```
from pyretic.lib.corelib import * # Importamos librerías de Pyretic
from pyretic.lib.std import *
from pyretic.pyresonance.fsm_policy import * # Importamos la política FSM
from pyretic.pyresonance.drivers.json_event import JSONEvent
# Importamos librerías JSON
from pyretic.modules.mac_learner import mac_learner
# Importamos la función mac_learner del módulo mac_learner
class seleccionar (DynamicPolicy):
    def __init__(self):
     # se define la función lpec
      def lpec(f):
        return match(switch=f['switch']) # Se escoge un switch
     # instalamos las funciones de transición
      def infected(event):
        return event
      def policy(state):
          if state['infected']:
              return drop # No se permite el flujo de paquetes
          else:
              return identity # Se permiten los paquetes originales
     # definimos la fsm donde se analiza si está o no infectado un switch
```

```
        self.fsm_description = {'infected' : (bool, False, NextFns(event_fn=infected)), 'policy' : ([drop,identity], identity, NextFns(state_fn=policy)) }
        # se define la política fsm y se reciben valores de los estados
        fsm_pol = FSMPolicy(lpec,self.fsm_description)
        json_event = JSONEvent()
        json_event.register_callback(fsm_pol.event_handler)
        super(seleccionar,self).__init__(fsm_pol)
    def main():
      return seleccionar() >> mac_learner()
```

A este módulo lo llamaremos seleccionar_resonance, lo ubicaremos en el directorio pyretic/pyretic/pyresonance/apps y para ejecutar éste módulo se usa el siguiente comando:

```
$ pyretic.py pyretic.pyresonance.apps.seleccionar_resonance
```

En la figura 16 se puede apreciar la ejecución de PyResonance con el módulo personalizado seleccionar_resonance.

```
mininet@mininet-vm:~$ pyretic.py pyretic.pyresonance.apps.seleccionar_resonance
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
Connected to pyretic frontend.
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[00-00-00-00-00-04 1] connected
INFO:openflow.of_01:[00-00-00-00-00-03 4] connected
INFO:openflow.of_01:[00-00-00-00-00-01 3] connected
INFO:openflow.of_01:[00-00-00-00-00-02 2] connected
Received connection from ('127.0.0.1', 59959)
```

Figura 16. Ejecución del controlador PyResonance con el módulo seleccionar_resonance

Para enviar los eventos al controlador utilizaremos un cliente JSON (JavaScript Object Notation), que lo podemos encontrar en el directorio pyretic/pyretic/pyresonance de la máquina virtual y se llama json_sender.py.

En la Figura 17 se aprecia la ejecución del comando para que el cliente JSON pase los valores de los eventos al controlador, en este caso el switch 4 es el que está infectado. En el comando también se debe especificar la dirección del controlador y el puerto de escucha.

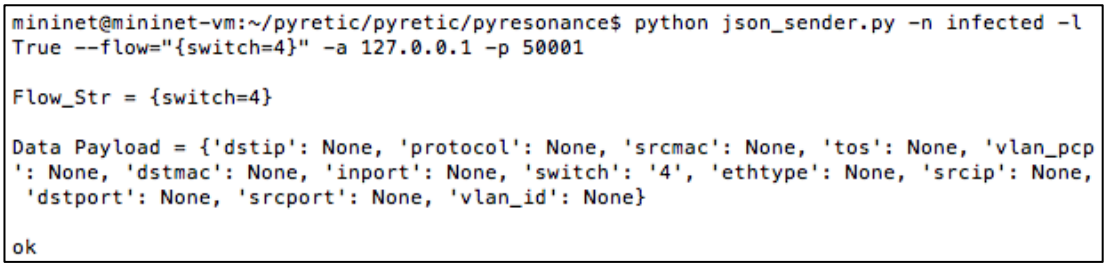


Figura 17. Ejecución del comando para que el cliente JSON pase los valores de los eventos al controlador

La topología usada para este controlador se la puede apreciar en la Figura 5 y es la misma que se utilizó anteriormente con POX y Pyretic. El host del switch infectado no podrá tener conectividad con ningún otro host de la red, mientras que los demás hosts si tendrán conectividad, esto lo podemos probar al ejecutar el comando pingall dentro de Mininet.

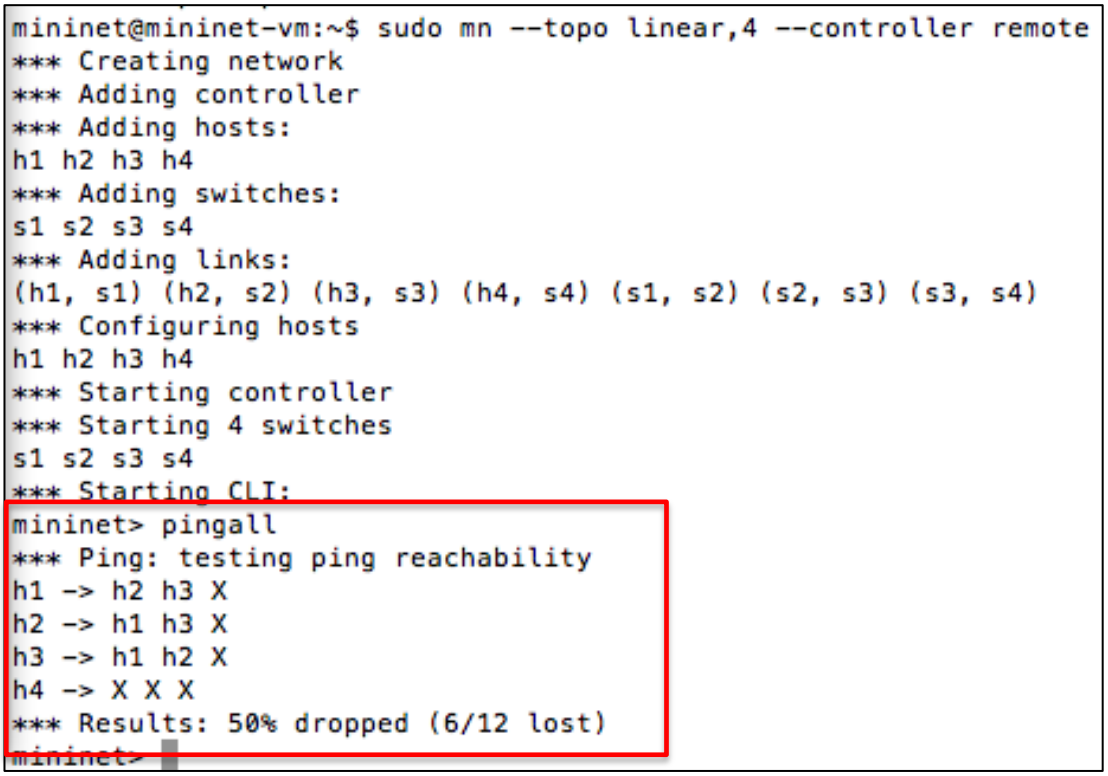


Figura 18. Topología creada en Mininet que usa el controlador PyResonance

CAPITULO 5: EVALUACIÓN DEL PROTOTIPO

5.1. Evaluación de Resultados

5.1.1. Evaluación de resultados en la simulación de la red definida por software (SDN)

5.1.1.1. Controlador POX

Iniciamos la topología personalizada en Mininet, figura 19, como se observó en el Capítulo 3, con el controlador POX, el módulo dhcpserv (Anexo 7), y el módulo personalizado ***seleccionar***.

```
mininet@mininet-vm:~/mininet/custom$ sudo python topologiaSDN.py
Unable to contact the remote controller at 127.0.0.1:6633
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
*** Starting 2 switches
s1 s2
Registro de las conexiones de los hosts
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s2-eth1
h4 h4-eth0:s2-eth2
*** Starting CLI:
mininet>
```

Figura 19. Topología creada en Mininet que usa el controlador POX

Como se aprecia en la figura 20 el módulo dhcpserv asigna direcciones a los host de toda la red, mientras que el módulo seleccionar no permite el flujo de paquetes en el switch seleccionado por lo tanto los host conectados al mismo no podrán tener conectividad ni entre ellos ni con los host del otro switch.

```
mininet@mininet-vm:~$ pox.py dhcpserv seleccionar --idswitch='00-00-00-00-00-01'
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
       SERVIDOR DHCP
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[00-00-00-00-00-02 1] connected
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:seleccionar:Switch Despreciado: [00-00-00-00-00-01 2]
INFO:dhcpserv:Asignada 192.168.10.8 a b2:3f:94:6a:81:fd
INFO:dhcpserv:Asignada 192.168.10.3 a fe:d9:05:94:6b:e4
INFO:packet:(dhcp parse) warning DHCP packet data too short to parse header: dat
a len 86
INFO:packet:(dhcp parse) warning DHCP packet data too short to parse header: dat
a len 86
INFO:dhcpserv:Asignada 192.168.10.9 a 4e:ad:fd:25:7b:5f
INFO:dhcpserv:Asignada 192.168.10.9 a 4e:ad:fd:25:7b:5f
INFO:packet:(dhcp parse) warning DHCP packet data too short to parse header: dat
a len 86
INFO:packet:(dhcp parse) warning DHCP packet data too short to parse header: dat
a len 86
INFO:dhcpserv:Asignada 192.168.10.10 a be:35:6e:0b:9e:c5
INFO:dhcpserv:Asignada 192.168.10.10 a be:35:6e:0b:9e:c5
```

Figura 20. Controlador POX con sus respectivos módulos

En la figura 21 se verifica que los host han obtenido direcciones IP después del requerimiento realizado al controlador mediante su módulo dhcpserv.

```
mininet> h1 ifconfig h1-eth0
h1-eth0   Link encap:Ethernet  HWaddr b2:3f:94:6a:81:fd
          inet addr:192.168.10.8  Bcast:192.168.10.255  Mask:255.255.255.0
          inet6 addr: fe80::b03f:94ff:fe6a:81fd/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1102 (1.1 KB)  TX bytes:1836 (1.8 KB)

mininet> h2 ifconfig h2-eth0
h2-eth0   Link encap:Ethernet  HWaddr fe:d9:05:94:6b:e4
          inet addr:192.168.10.3  Bcast:192.168.10.255  Mask:255.255.255.0
          inet6 addr: fe80::fcd9:5ff:fe94:6be4/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1102 (1.1 KB)  TX bytes:1152 (1.1 KB)

mininet> h3 ifconfig h3-eth0
h3-eth0   Link encap:Ethernet  HWaddr 4e:ad:fd:25:7b:5f
          inet addr:192.168.10.9  Bcast:192.168.10.255  Mask:255.255.255.0
          inet6 addr: fe80::4cad:fdff:fe25:7b5f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:16 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2740 (2.7 KB)  TX bytes:1152 (1.1 KB)

mininet> h4 ifconfig h4-eth0
h4-eth0   Link encap:Ethernet  HWaddr be:35:6e:0b:9e:c5
          inet addr:192.168.10.10  Bcast:192.168.10.255  Mask:255.255.255.0
          inet6 addr: fe80::bc35:6eff:fe0b:9ec5/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:15 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2650 (2.6 KB)  TX bytes:1152 (1.1 KB)

mininet>
```

Figura 21. Asignación de las direcciones IP a los Host por el módulo dhcpserv

Como el host 1 (h1) y host 2 (h2) se encuentran conectados al switch seleccionado por lo tanto no existirá forma que puedan tener flujo de paquetes de datos ni entre ellos ni con los host 3 (h3) ni 4 (h4) como se indicó anteriormente. Como se puede comprobar mediante la ejecución del comando ping entre ellos mostrado en la figura 22.

La latencia máxima de respuesta del comando ping es de 39.7 ms y la menor es de 0.043 ms, estos datos servirán para poder comparar el tiempo de respuesta que existe para cada controlador.

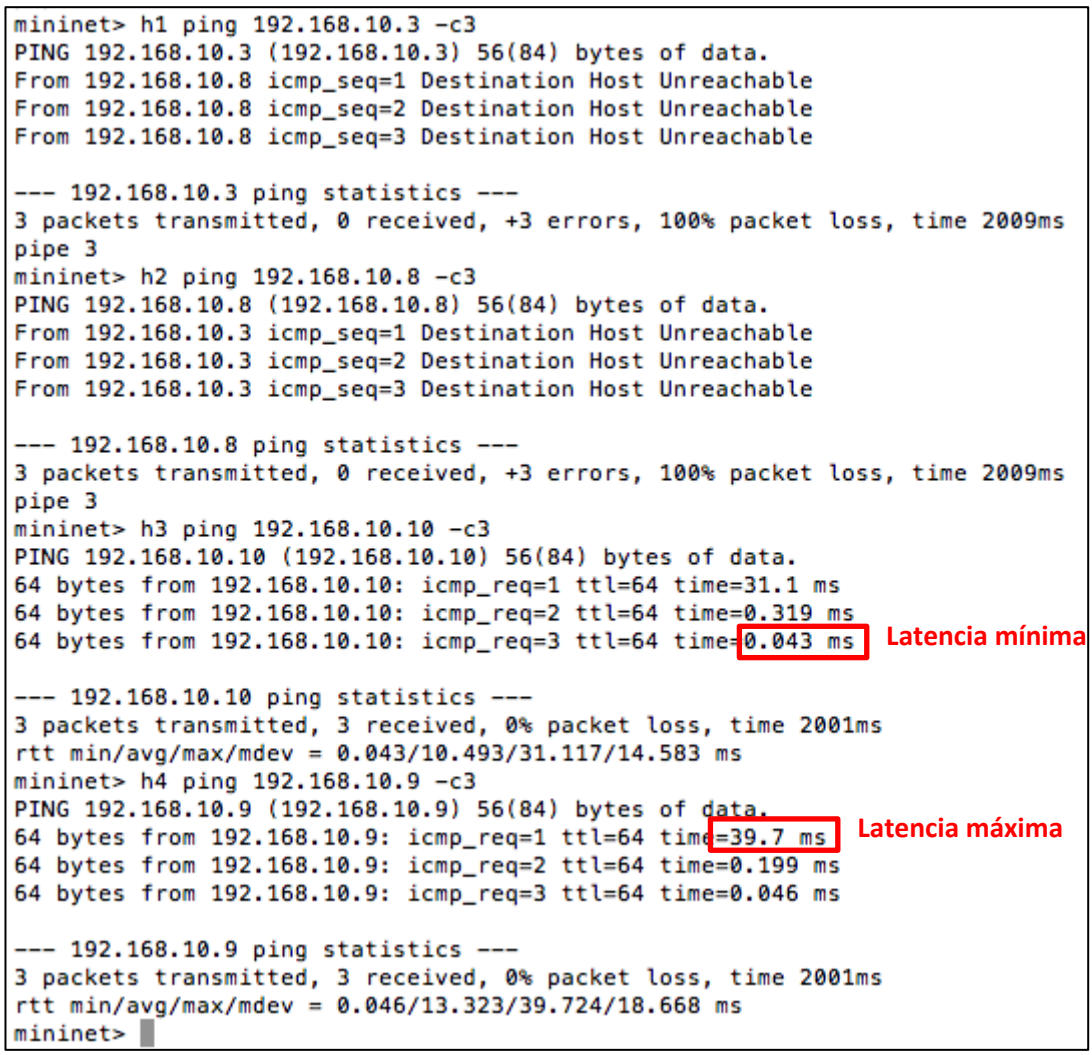


Figura 22. Resultados en el controlador POX

Cabe recalcar que el módulo desarrollado con el nombre seleccionar, contiene en su código al módulo l2_learning y es por esto que se puede permitir o negar el flujo de paquetes como se explicó anteriormente en el capítulo 4.

5.1.1.2. Controlador Pyretic

Modificamos la topología personalizada en Mininet que se observó en el Capítulo 3, comentando el código que se indica en la figura 23, para crear

una topología en la cual los host no realicen peticiones DHCP, ya que en Pyretic no se ha desarrollado un módulo que asigne direcciones IP.

Los host de esta topología tienen una dirección IP asignada por defecto por Mininet.

```
# Configuracion de clientes dhcp
'''
h1 = net.get('h1')
h1.cmd('dhclient -r')
h1.cmd('dhclient h1-eth0')
h2 = net.get('h2')
h2.cmd('dhclient -r')
h2.cmd('dhclient h2-eth0')
h3 = net.get('h3')
h3.cmd('dhclient -r')
h3.cmd('dhclient h3-eth0')
h4 = net.get('h4')
h4.cmd('dhclient -r')
h4.cmd('dhclient h4-eth0')
'''
```

Figura 23. Código comentado para modificación de topología de Pyretic

La topología usada para el controlador Pyretic la podemos ejecutar con el comando:

$ sudo python topologiaSDN_Pyretic.py

```
mininet@mininet-vm:~/mininet/custom$ sudo python topologiaSDN_Pyretic.py
Unable to contact the remote controller at 127.0.0.1:6633
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
*** Starting 2 switches
s1 s2
Registro de las conexiones de los hosts
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s2-eth1
h4 h4-eth0:s2-eth2
*** Starting CLI:
mininet>
```

Figura 24. Topología creada en Mininet que usa el controlador Pyretic

En la figura 25 se observa la ejecución del controlador Pyretic con el módulo seleccionar que se programó anteriormente en el capítulo 4 y tiene el mismo funcionamiento que el módulo seleccionar del controlador POX.

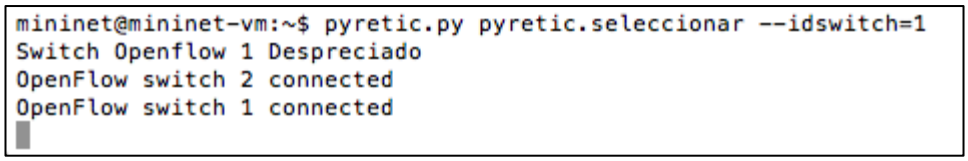


Figura 25 Controlador Pyretic con el módulo seleccionar.

En la figura 26 se observa que Mininet se encarga de asignar direcciones IP a los host por defecto sin necesidad de que exista un módulo DHCP para Pyretic.

```
mininet> h1 ifconfig h1-eth0
h1-eth0   Link encap:Ethernet  HWaddr 86:29:87:31:e5:18
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:28 errors:0 dropped:2 overruns:0 frame:0
          TX packets:44 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2006 (2.0 KB)  TX bytes:2848 (2.8 KB)

mininet> h2 ifconfig h2-eth0
h2-eth0   Link encap:Ethernet  HWaddr b6:3c:3b:1f:da:7b
          inet addr:10.0.0.2  Bcast:10.255.255.255  Mask:255.0.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:28 errors:0 dropped:2 overruns:0 frame:0
          TX packets:38 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2006 (2.0 KB)  TX bytes:2596 (2.5 KB)

mininet> h3 ifconfig h3-eth0
h3-eth0   Link encap:Ethernet  HWaddr 2e:25:73:37:9a:ec
          inet addr:10.0.0.3  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::2c25:73ff:fe37:9aec/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:42 errors:0 dropped:2 overruns:0 frame:0
          TX packets:37 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2874 (2.8 KB)  TX bytes:2666 (2.6 KB)

mininet> h4 ifconfig h4-eth0
h4-eth0   Link encap:Ethernet  HWaddr a6:94:99:9c:a0:9c
          inet addr:10.0.0.4  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::a494:99ff:fe9c:a09c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:42 errors:0 dropped:2 overruns:0 frame:0
          TX packets:37 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2874 (2.8 KB)  TX bytes:2666 (2.6 KB)

mininet>
```

Figura 26. Asignación de las direcciones IP a los Host por Mininet

Los resultados son similares al del controlador POX y podemos observarlos en la figura 27 en donde el host 1 (h1) y host 2 (h2) se encuentran conectados al switch seleccionado por lo tanto no existirá forma que puedan tener flujo de paquetes de datos ni entre ellos ni con los hosts del otro switch. Como se puede comprobar mediante la ejecución del comando ping.

La latencia máxima de respuesta del comando ping es de 70.8 ms y la menor es de 0.031 ms, estos datos servirán para poder comparar el tiempo de respuesta que existe para cada controlador.

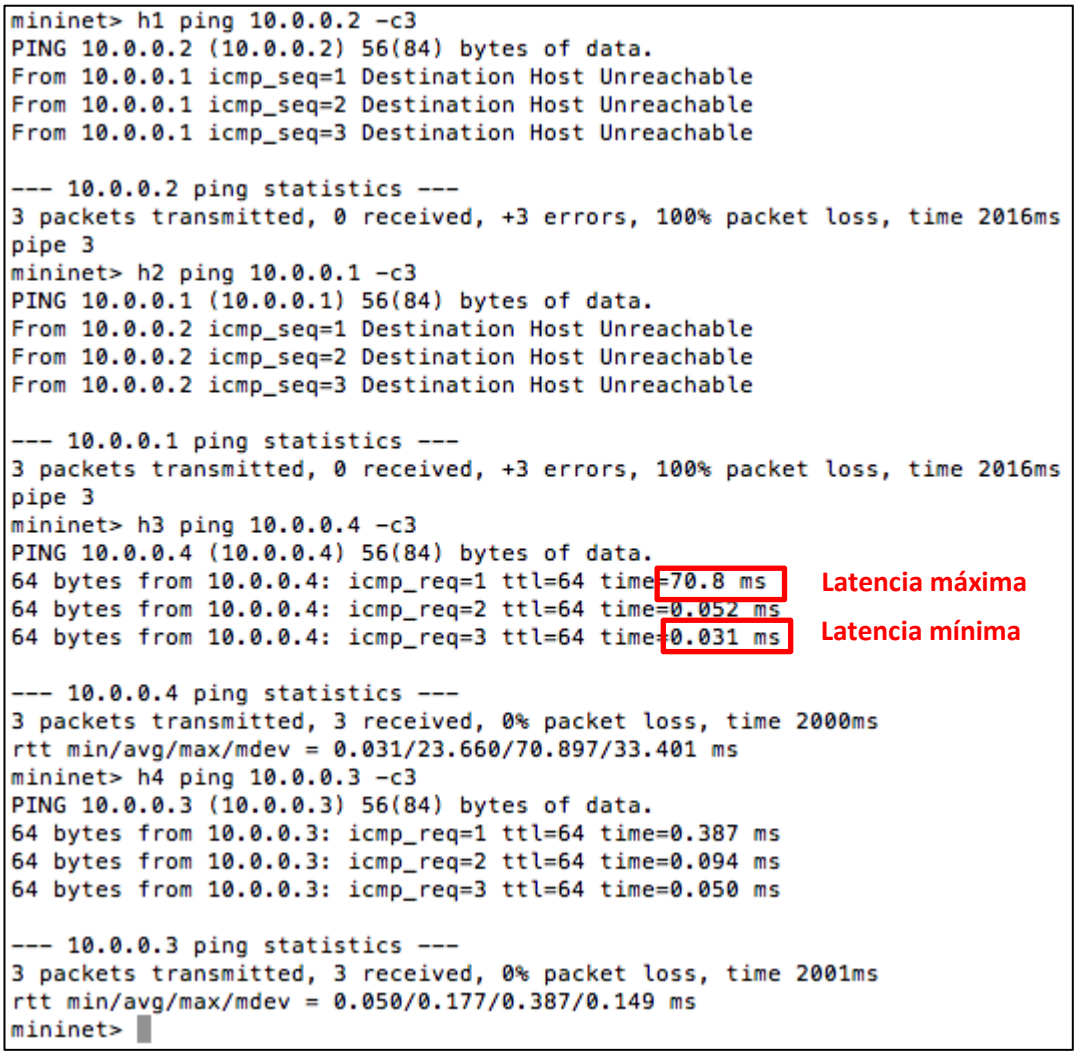


Figura 27 Resultados en el controldor Pyretic.

El módulo desarrollado con el nombre “seleccionar”, contiene en su código al módulo mac_learner y es por esto que se puede permitir o negar el flujo de paquetes como se explicó anteriormente en el capítulo 4.

5.1.1.3. Controlador PyResonance

Inicializamos la topología personalizada en Mininet, que es similar a la utilizada con el controlador Pyretic, se ejecuta PyResonance con el módulo seleccionar_resonance creado anteriormente en el capítulo 4, y en este caso se utiliza un cliente JSON para especificar el switch infectado como se señaló en la figura 17.

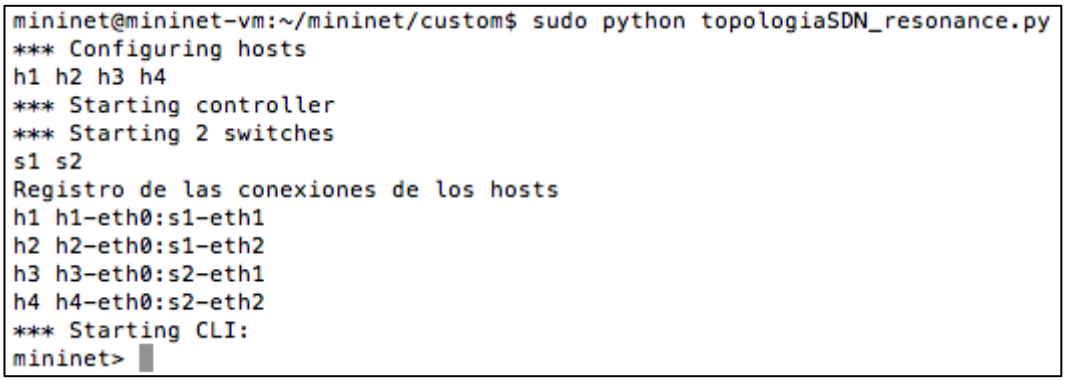


Figura 28 Topología creada en Mininet que usa el controlador PyResonance.

En la figura 29 se observa la ejecución del controlador PyResonance con el módulo seleccionar que se programó anteriormente en el capítulo 4.

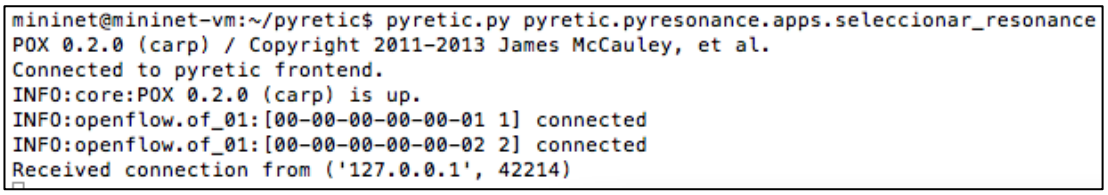


Figura 29 Controlador PyResonance con el módulo seleccionar_resonance.

En la figura 30 se observa que la latencia máxima de respuesta del comando ping es de 101 ms y la menor es de 0.047 ms, y comparando los tres controladores se puede constatar que el tiempo de respuesta del primer

paquete enviado con el comando ping se demora más que los paquetes subsiguientes donde se evidencia que existe un tiempo de retardo hasta que se instale por primera vez una dirección o ruta en la tabla flujos.

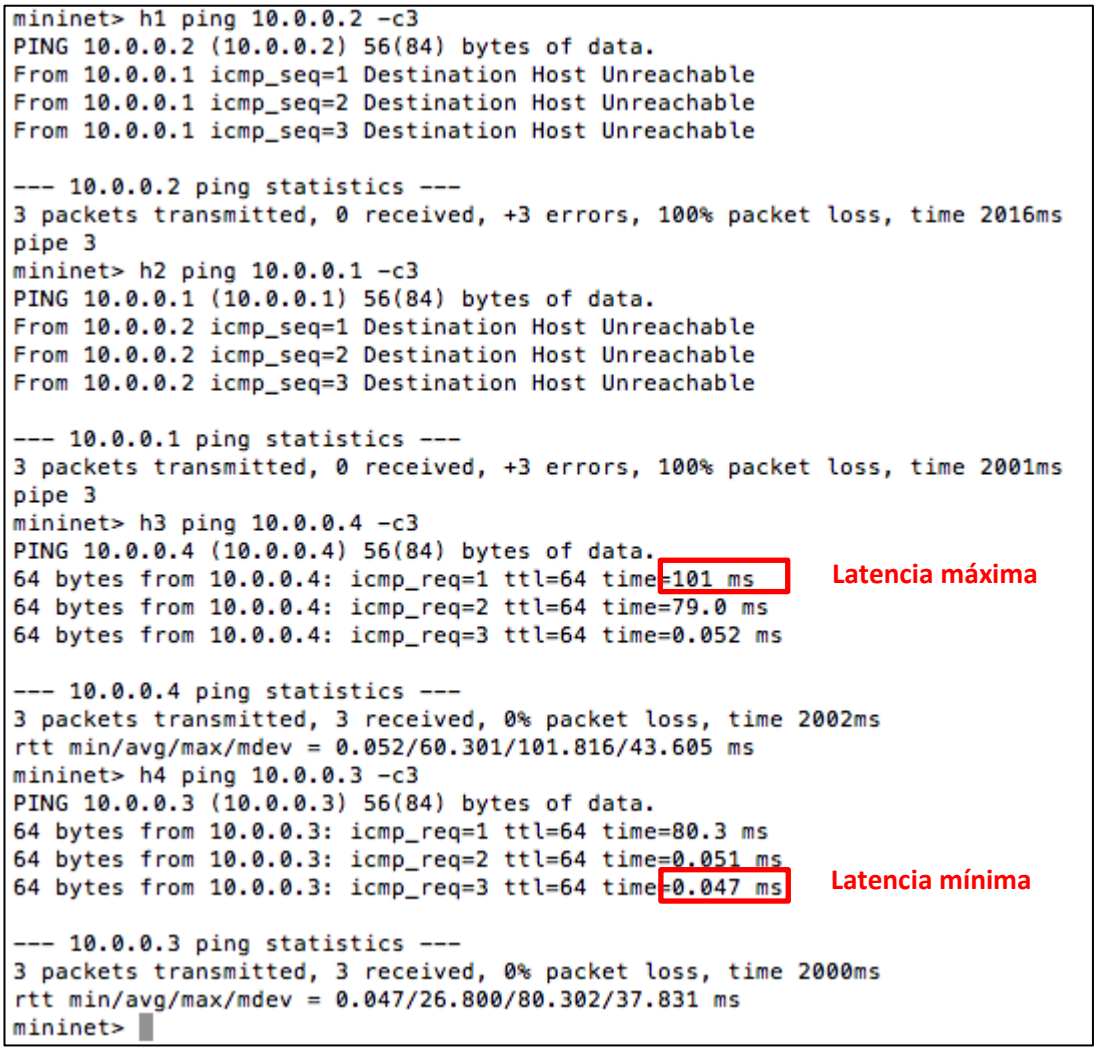


Figura 30. Resultados en el controldor PyResonance

5.1.2. Evaluación de resultados en la implementación de la red definida por software (SDN)

5.1.2.1. Controlador POX

Iniciamos el controlador POX en la topología física, con el módulo dhcpserv y seleccionar, para esto ejecutamos el comando que se muestra en la figura 31.

```
mininet@mininet-vm:~$ pox.py dhcpserv seleccionar --idswitch='d4-ca-6d-b3-57-0e|1'
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
       SERVIDOR DHCP
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[d4-ca-6d-af-a8-34|2 1] connected
INFO:openflow.of_01:[d4-ca-6d-b3-57-0e|1 2] connected
INFO:seleccionar:Switch Despreciado: [d4-ca-6d-b3-57-0e|1 2]
INFO:dhcpserv:Asignada 192.168.10.2 a 18:67:b0:a0:26:06
INFO:dhcpserv:Asignada 192.168.10.2 a 18:67:b0:a0:26:06
INFO:dhcpserv:Asignada 192.168.10.3 a 00:1c:c0:fd:83:3e
INFO:dhcpserv:Asignada 192.168.10.3 a 00:1c:c0:fd:83:3e
INFO:dhcpserv:Asignada 192.168.10.6 a 40:6c:8f:57:96:66
INFO:dhcpserv:Asignada 192.168.10.4 a 08:00:27:0a:dc:f0
```

Figura 31 Controlador POX con sus respectivos módulos.

En este caso se desprecia el switch 1 (SW1), esto quiere decir que solo los host del switch 2 (SW2) tienen conectividad lógica, por lo tanto nos enfocaremos en ellos para obtener la latencia máxima y la mínima.

```
C:\Users\PABLO>ping 192.168.10.3

Haciendo ping a 192.168.10.3 con 32 bytes de datos:
Respuesta desde 192.168.10.3: bytes=32 tiempo=510ms TTL=128
Respuesta desde 192.168.10.3: bytes=32 tiempo=2ms TTL=128
Respuesta desde 192.168.10.3: bytes=32 tiempo=3ms TTL=128
Respuesta desde 192.168.10.3: bytes=32 tiempo=2ms TTL=128

Estadísticas de ping para 192.168.10.3:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
    (0% perdidos),
Tiempos aproximados de ida y vuelta en milisegundos:
    Mínimo = 2ms, Máximo = 510ms, Media = 129ms

C:\Users\PABLO>_
```

Figura 32. Resultados con el controlador POX desde PC_PABLO

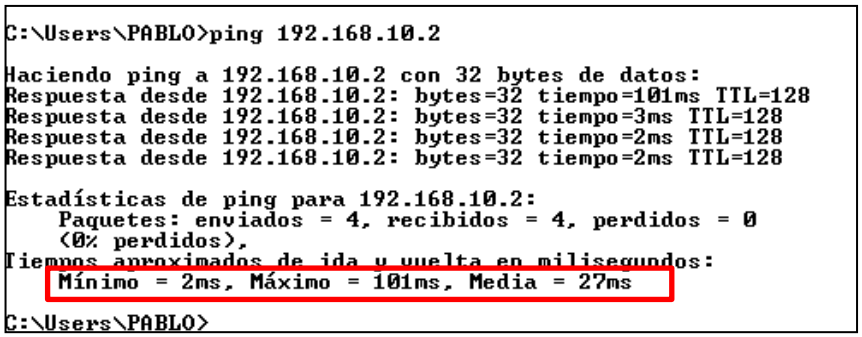


Figura 33. Resultados con el controlador POX desde PC_PABLO 1

Usando el controlador POX, con los módulos dhcpserv y seleccionar, los únicos host que poseen conectividad lógica son PC_PABLO y PC_PABLO1 los cuales pertenecen al SW2, mientras que los otros hosts pertenecientes al SW1 no poseen conectividad lógica con ningún host de la red.

Analizando las figuras 32 y 33 se puede determinar que la latencia mínima es de 2 (ms) mientras que la latencia máxima es de 510 (ms).

5.1.2.2. Controlador Pyretic

Iniciamos el controlador Pyretic en la topología física, con el módulo seleccionar, para esto ejecutamos el comando que se muestra en la figura 34. En Pyretic no tenemos un módulo que funcione como DHCP, por lo tanto pondremos direcciones estáticas en los host físicos para realizar la prueba de conectividad.

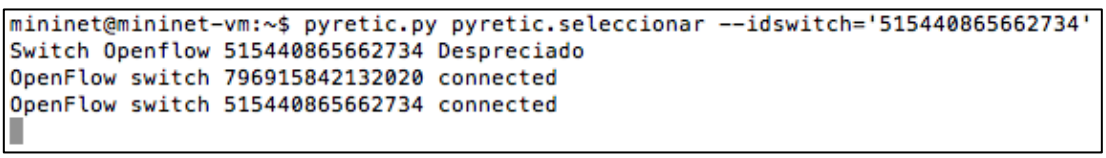


Figura 34 Controlador Pyretic con sus respectivos módulos.

Se desprecia el switch 1 (SW1), lo que quiere decir que solo los host del switch 2 (SW2) tienen conectividad lógica, por lo tanto nos enfocaremos en ellos para obtener la latencia máxima y la mínima.

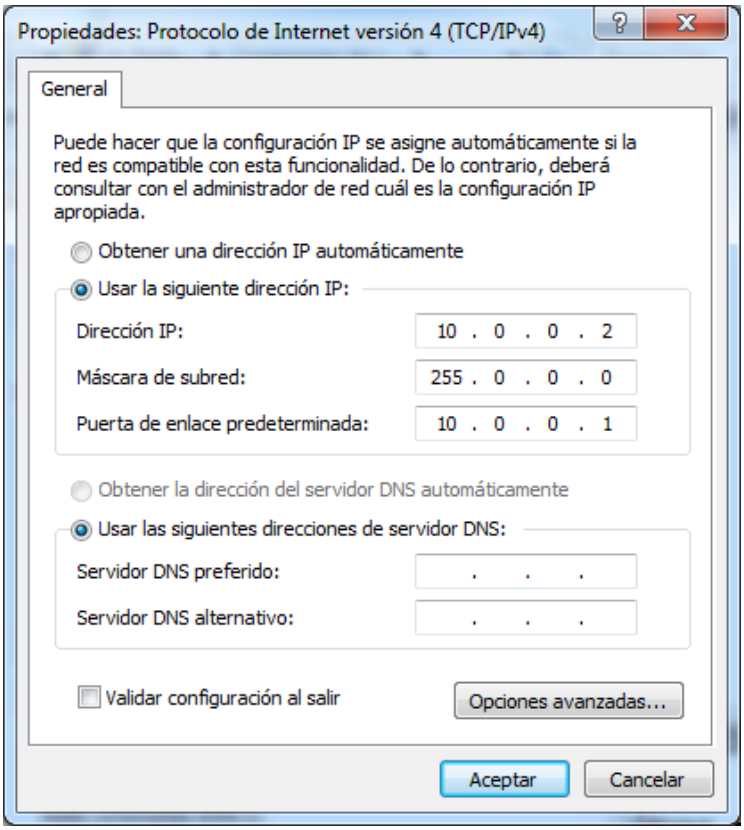


Figura 35. Configuración de dirección estática en PC_PABLO

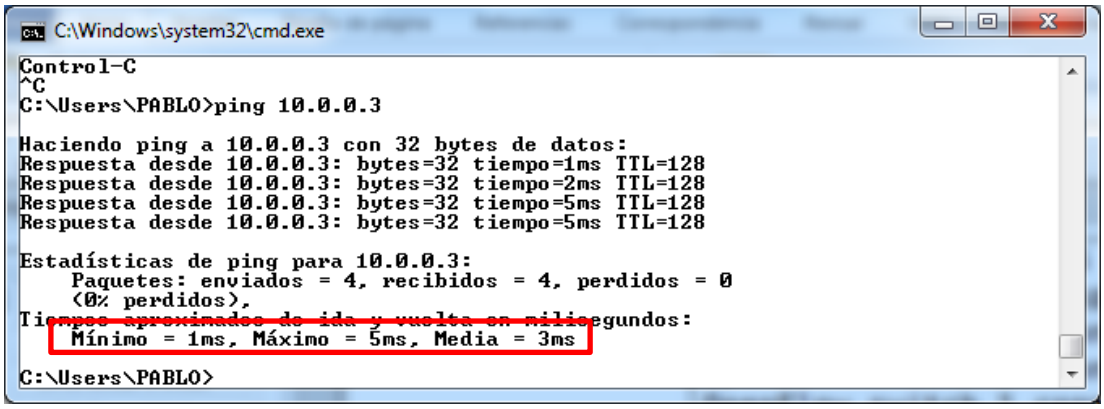


Figura 36. Resultados con el controlador Pyretic desde PC_PABLO

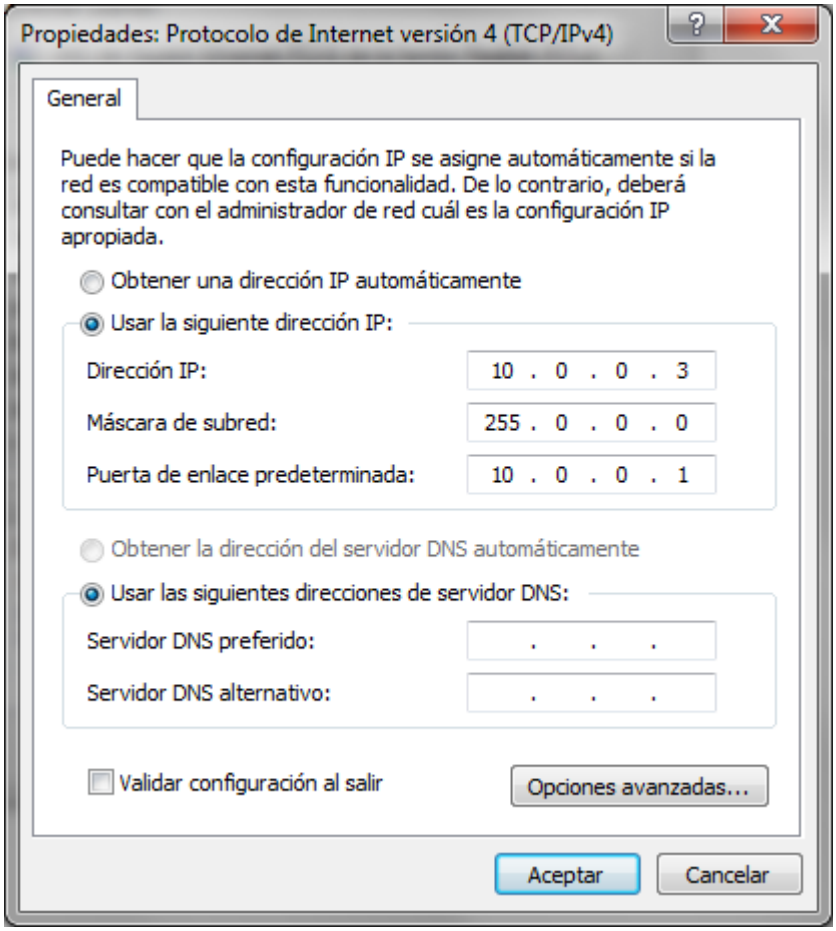


Figura 37. Configuración de dirección estática en PC_PABLO 1

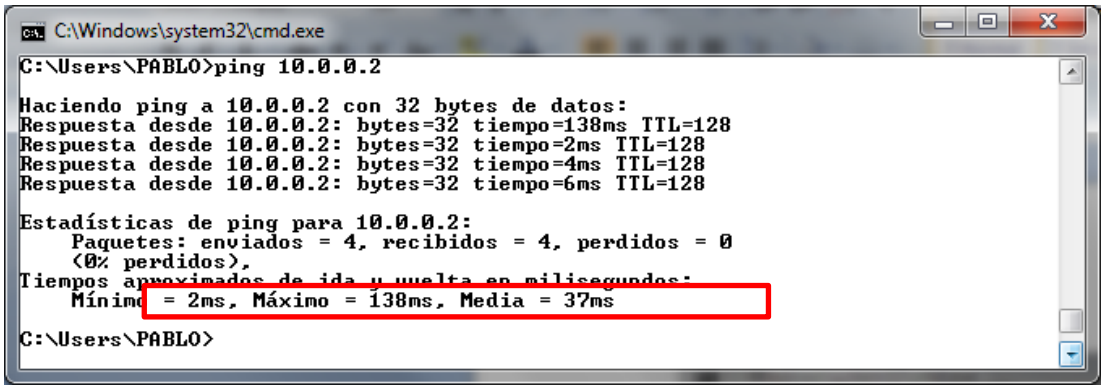


Figura 38. Resultados con el controlador Pyretic desde PC_PABLO 1

Usando el controlador Pyretic, con su módulo seleccionar, los únicos host que poseen conectividad lógica son el PC_PABLO y PC_PABLO1 los cuales pertenecen al SW2, mientras que los otros hosts que pertenecen al SW1 no poseen conectividad lógica con ningún host de la red.

Analizando las figuras 36 y 38 se puede determinar que la latencia mínima es de 1 (ms), mientras que la latencia máxima es de 138 (ms).

5.1.2.3. Controlador PyResonance

Iniciamos el controlador PyResonance en la topología física, con el módulo seleccionar_resonance, para esto ejecutamos el comando que se muestra en la figura 39. En PyResonance tampoco tenemos un módulo que funcione como DHCP, por lo tanto se pondrá direcciones estáticas en los host físicos al igual que en Pyretic.

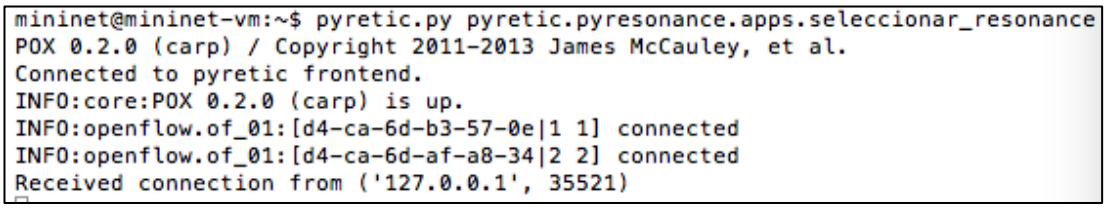


Figura 39. Controlador PyResonance con el módulo seleccionar_resonance

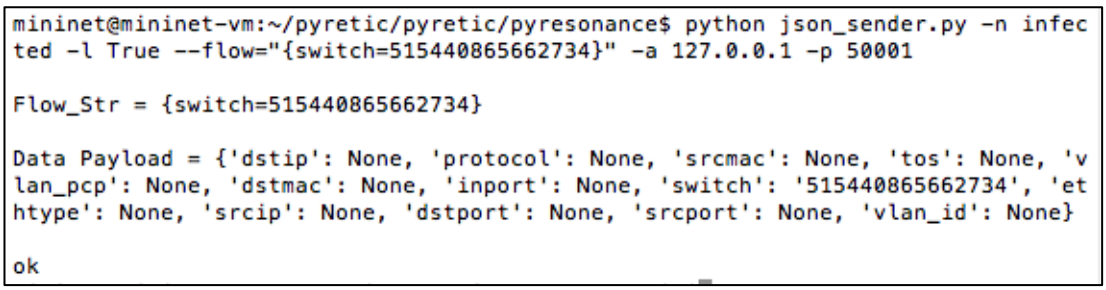


Figura 40. Controlador PyResonance con el módulo seleccionar

Se infecta el switch 1 (SW1) mediante el cliente JSON, lo que quiere decir que solo los host del switch 2 (SW2) tienen conectividad lógica, por lo tanto nos enfocaremos en ellos para obtener la latencia máxima y la mínima.

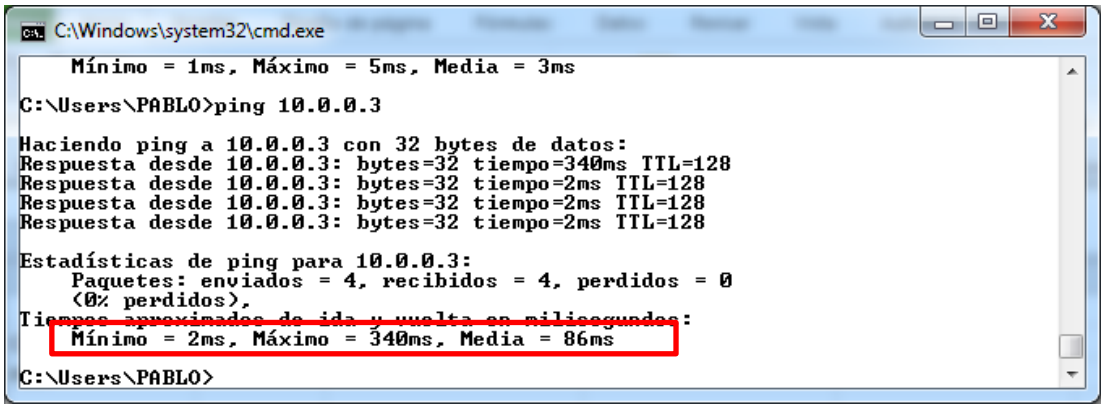


Figura 41. Resultados con el controlador PyResonance desde PC_PABLO

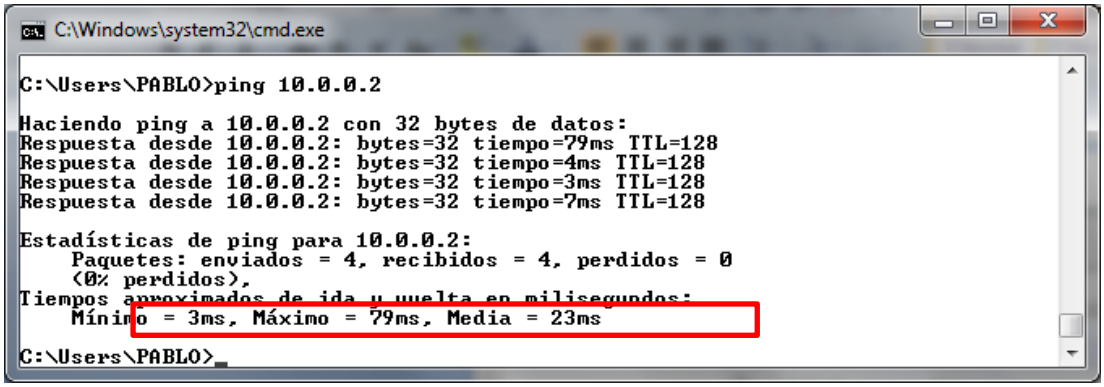


Figura 42. Resultados con el controlador PyResonance desde PC_PABLO 1

Usando el controlador PyResonance, con su módulo seleccionar_resonance, los únicos host que poseen conectividad lógica son el PC_PABLO y PC_PABLO1 los cuales pertenecen al SW2, mientras que los otros hosts que pertenecen al SW1, que se encuentra infectado, no poseen conectividad lógica con ningún host de la red.

Analizando las figuras 41 y 42 se puede determinar que la latencia mínima es de 2 (ms), mientras que la latencia máxima es de 340 (ms).

5.1.2.4. Controladores POX y Pyretic

Tratando de emular la realidad en la implementación del prototipo de red SDN se hizo coexistir a dos controladores a la vez, en el primer caso se lo realizó entre los controladores POX y Pyretic con los módulos dhcpserv y mac_learner respectivamente. Cabe recalcar que cada controlador estaría ejecutándose en una máquina virtual distinta.

En la figura 43 se observa el funcionamiento del módulo dhcpserv que asigna direcciones IP a los host en la red física. En las figuras 45, 46, 47 y 48 se verifica las direcciones asignadas por este módulo.

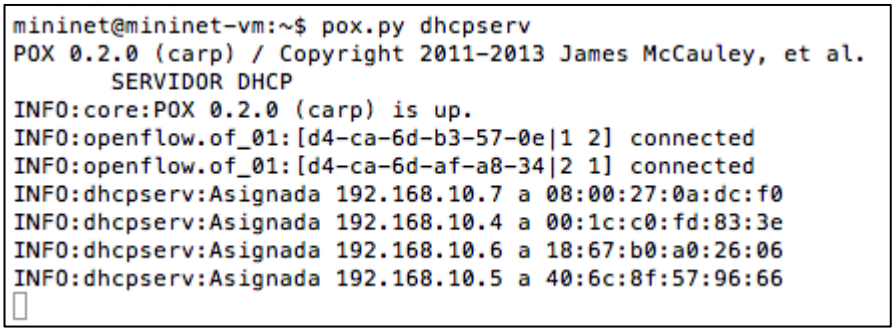


Figura 43. Controlador POX con el módulo dhcpserv

En la figura 44 se observa el funcionamiento del módulo mac_learner del controlador Pyretic y se constata que el manejo de los identificadores para los conmutadores es distinto al del controlador POX.

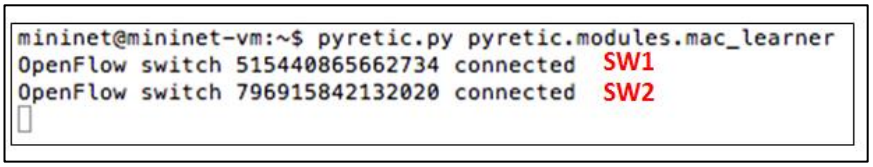


Figura 44. Controlador Pyretic con el módulo mac_learner

La dirección IP asignada por el módulo dhcpserv del controlador POX para el host PC_DARIO es 192.168.10.5/24 y se encuentra en el SW1.

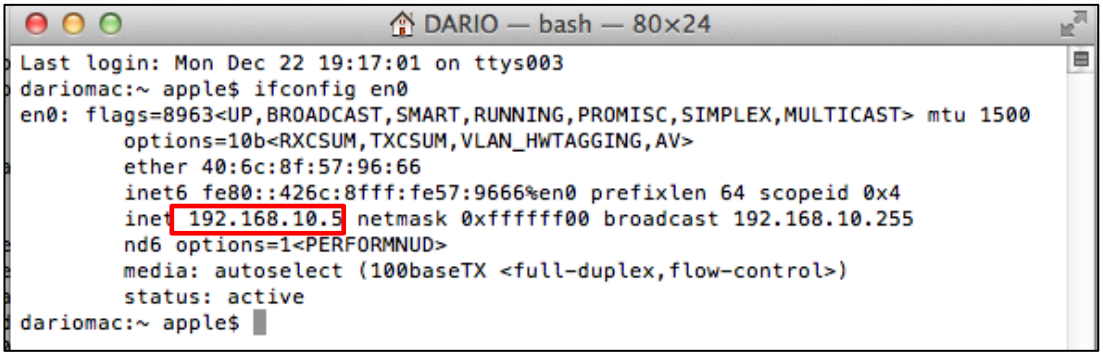


Figura 45. Verificación de la dirección IP asignada a PC_DARIO

La dirección IP asignada por el módulo dhcpserv del controlador POX para el host PC_DARIOMAC es 192.168.10.7/24 y se encuentra en el SW1.

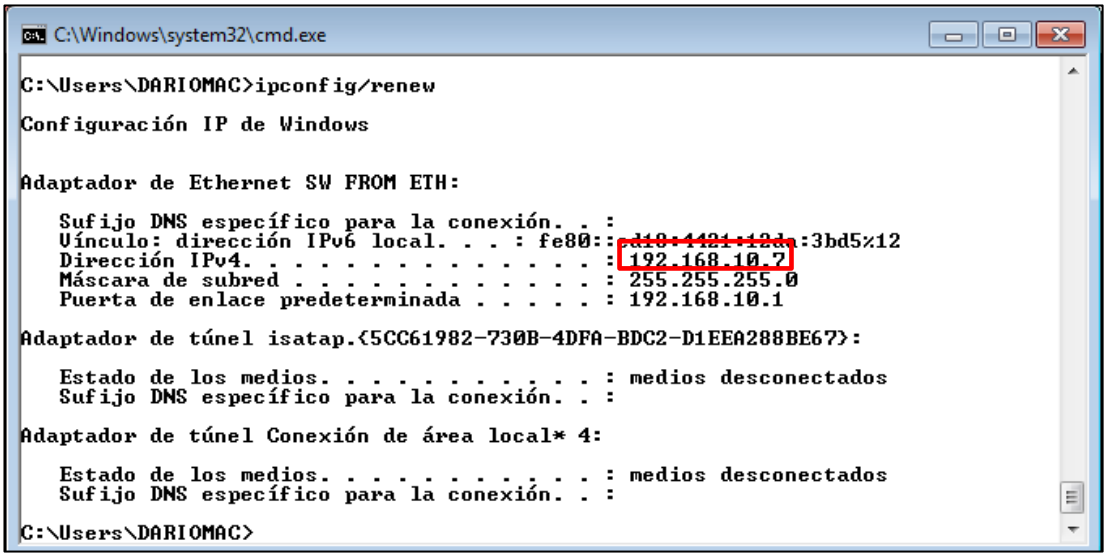


Figura 46. Asignación de dirección PC_DARIOMAC

La dirección IP asignada por el módulo dhcpserv del controlador POX para el host PC_PABLO es 192.168.10.6/24 y se encuentra en el SW2.

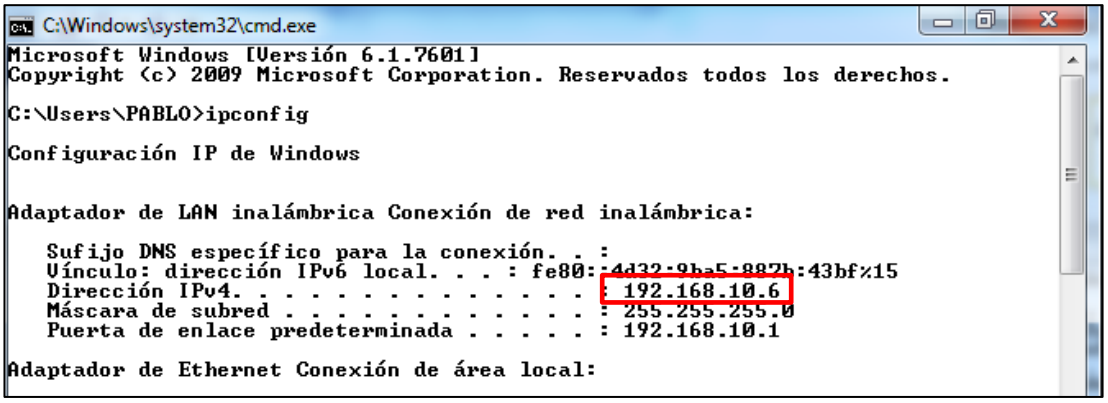


Figura 47. Asignación de dirección PC_PABLO

La dirección IP asignada por el módulo dhcpserv del controlador POX para el host PC_PABLO1 es 192.168.10.4/24 y se encuentra en el SW2.

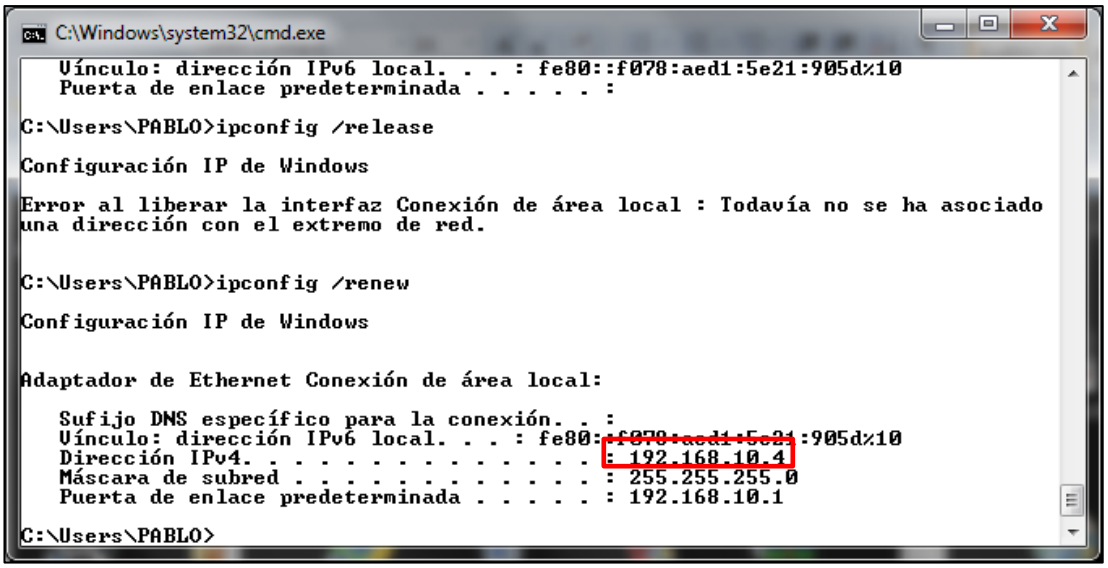


Figura 48. Asignación de dirección PC_PABLO1

En las siguientes figuras de la 49 a la 52 se constata la conectividad entre todos los host de la red sin importar si están conectados al SW1 o al SW2, esto indica que los módulos tanto del controlador POX como del controlador Pyretic están funcionando correctamente.

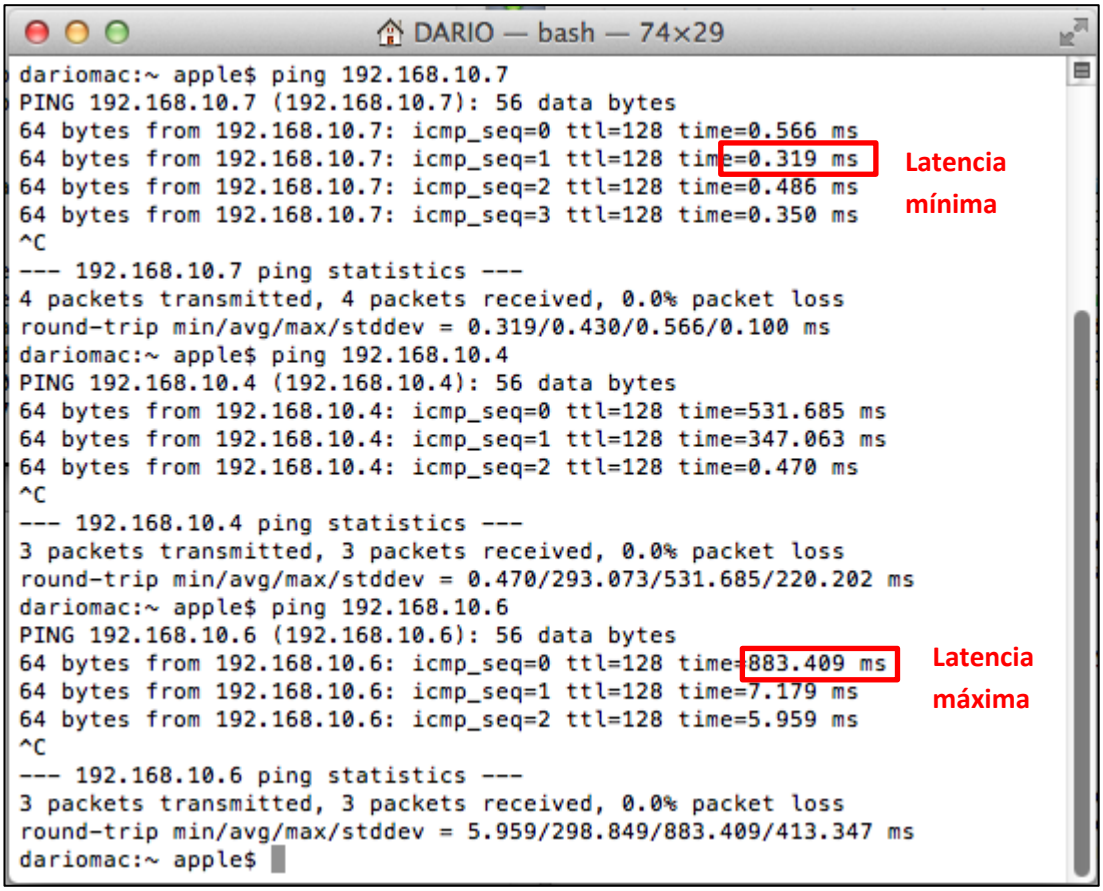


Figura 49. Resultado de la ejecución de los controladores POX y Pyretic desde PC_DARIO

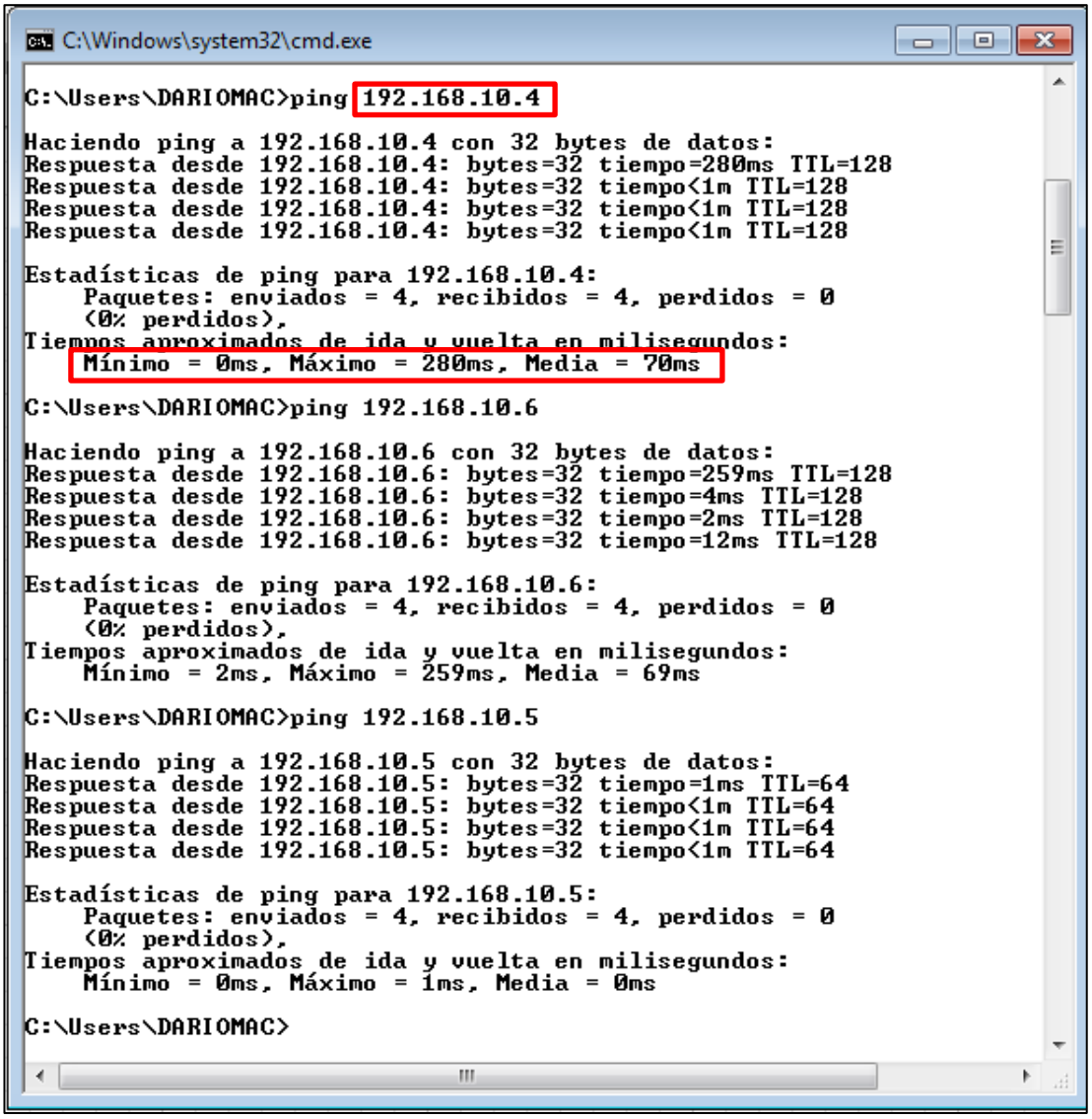


Figura 50. Resultado de la ejecución de los controladores POX y Pyretic desde PC_DARIOMAC

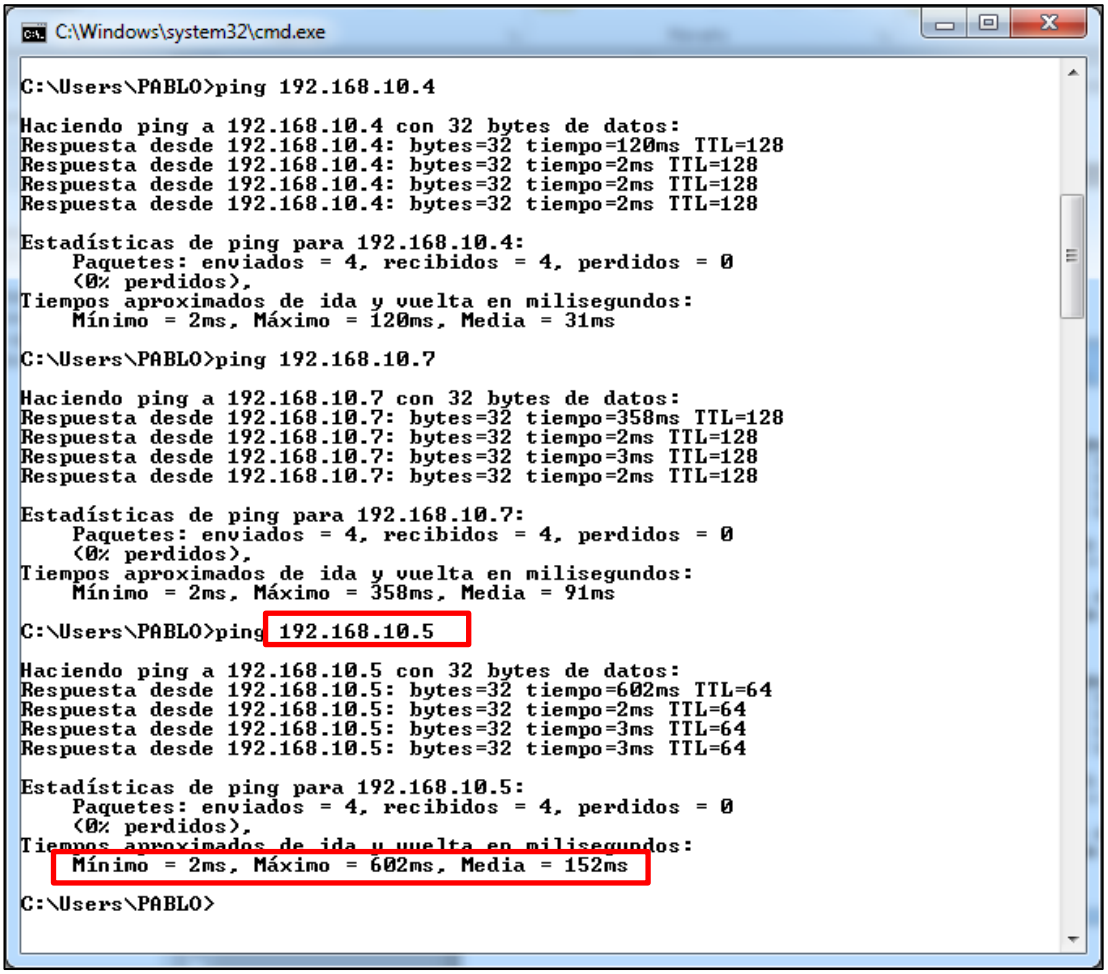


Figura 51. Resultado de la ejecución de los controladores POX y Pyretic desde PC_PABLO

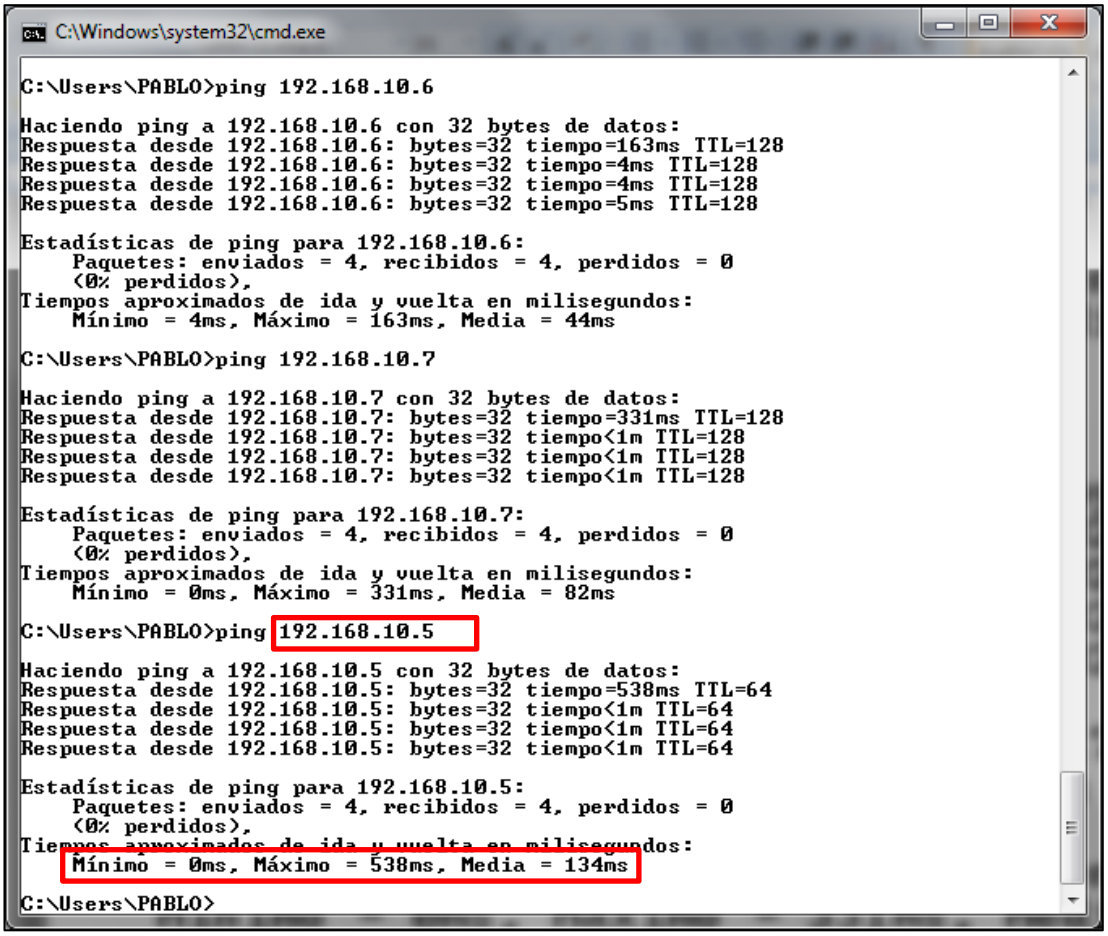


Figura 52. Resultado de la ejecución de los controladores POX y Pyretic desde PC_PABLO1

Usando los controladores POX y Pyretic a la vez, con los módulos dhcpserv y seleccionar respectivamente, todos los host poseen conectividad física y lógica sin importar si pertenecen al SW1 o al SW2. Analizando los resultados se obtuvo una latencia mínima de 0 (ms) entre PC_DARIOMAC y PC_DARIO y una latencia máxima de 883,409 (ms) entre PC_DARIO y PC_PABLO.

5.1.2.5. Controladores POX y PyResonance

En el segundo caso se hizo coexistir a los controladores POX y PyResonance con los módulos dhcpserv y seleccionar_resonance

respectivamente. El funcionamiento del controlador POX es el mismo en los dos casos por lo tanto nos referiremos al funcionamiento del controlador PyResonance visto en la figura 53.

```
mininet@mininet-vm:~$ pyretic.py pyretic.pyresonance.apps.seleccionar_resonance
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
Connected to pyretic frontend.
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[d4-ca-6d-af-a8-34|2 2] connected
INFO:openflow.of_01:[d4-ca-6d-b3-57-0e|1 1] connected
Received connection from ('127.0.0.1', 42259)
Received connection from ('127.0.0.1', 42260)
Received connection from ('127.0.0.1', 42261)
Received connection from ('127.0.0.1', 42262)
```

Figura 53. Funcionamiento del controlador PyResonance con el módulo seleccionar_resonance

5.1.2.5.1. Controladores POX y PyResonance (TRUE)

Para enviar los valores de los estados al controlador Pyretic se usa un cliente JSON, tal como se mencionó en el capítulo 4, en la figura 54 se observa que se envía el tipo de dato lógico TRUE al estado infected, el filtro Pyretic en el cual se especifica el identificador del SW2, la dirección IP del controlador y el puerto de escucha. Esto quiere decir que en la máquina de estados finita se condiciona a que “no” exista flujo de paquetes en SW2, ya que en un inicio existe conectividad lógica entre toda la red SDN.

```
mininet@mininet-vm:~/pyretic/pyretic/pyresonance$ python json_sender.py -n infected -l
True --flow="{switch=796915842132020}" -a 127.0.0.1 -p 50001

Flow_Str = {switch=796915842132020}

Data Payload = {'dstip': None, 'protocol': None, 'srcmac': None, 'tos': None, 'vlan_pcp
': None, 'dstmac': None, 'inport': None, 'switch': '796915842132020', 'ethtype': None,
'srcip': None, 'dstport': None, 'srcport': None, 'vlan_id': None}

ok
```

Figura 54. Envio del valor True al estado infected del controlador Pyretic mediante el cliente JSON

En las figuras 55, 56, 57 y 58 se observa los resultados de las pruebas al denegar el flujo de paquetes en SW2, es decir que todos los host conectados a este switch no tendrán conectividad lógica entre ellos ni con los host de SW1, por otro lado los host de SW1 tendrán conectividad lógica entre ellos pero no con los host de SW2.

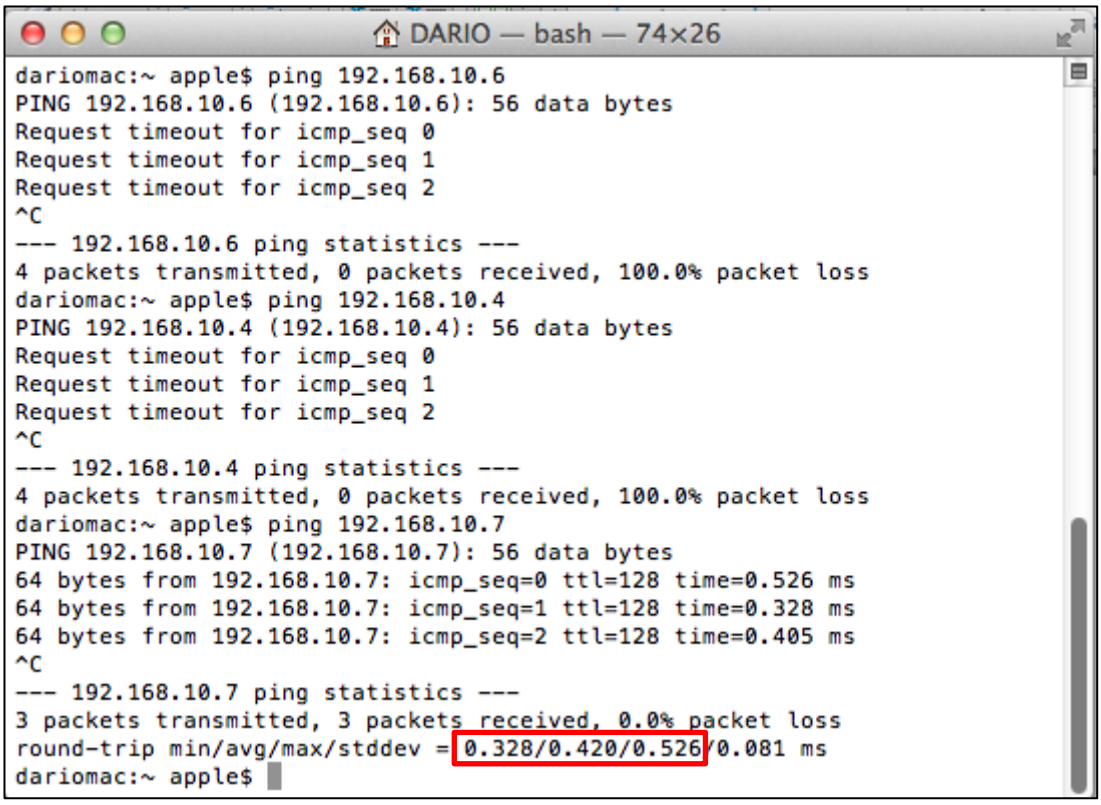


Figura 55. Resultado de la ejecución de los controladores POX y PyResonance desde PC_DARIO con el estado infected (TRUE)

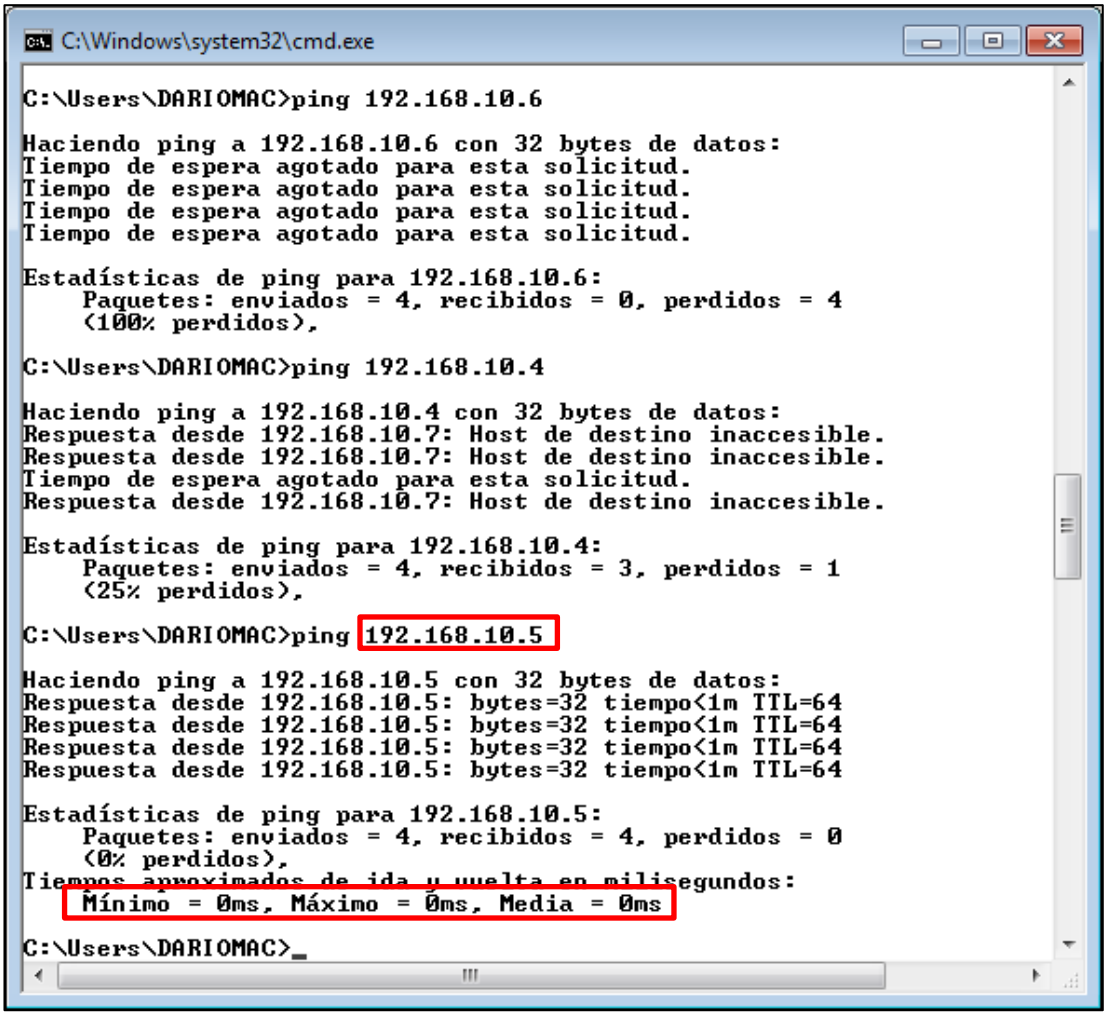


Figura 56. Resultado de la ejecución de los controladores POX y PyResonance desde PC_DARIOMAC con el estado infected (TRUE)

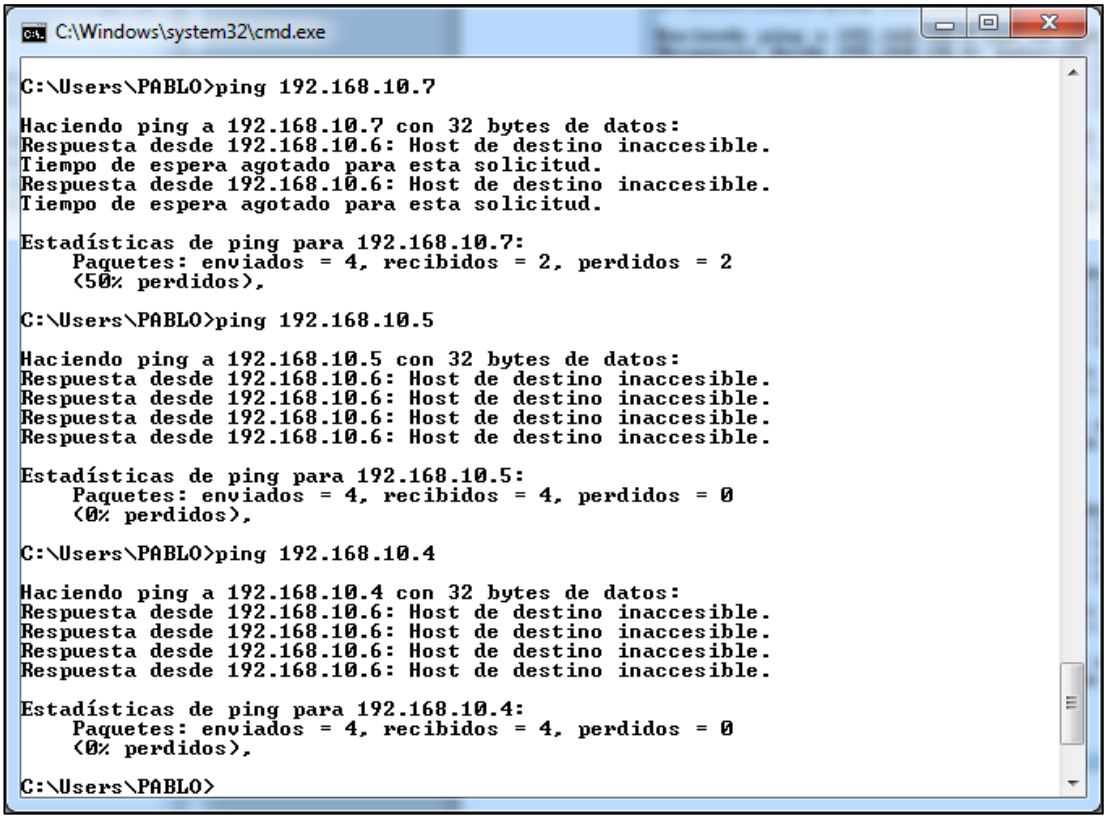


Figura 57. Resultado de la ejecución de los controladores POX y PyResonance desde PC_PABLO con el estado infected (TRUE)

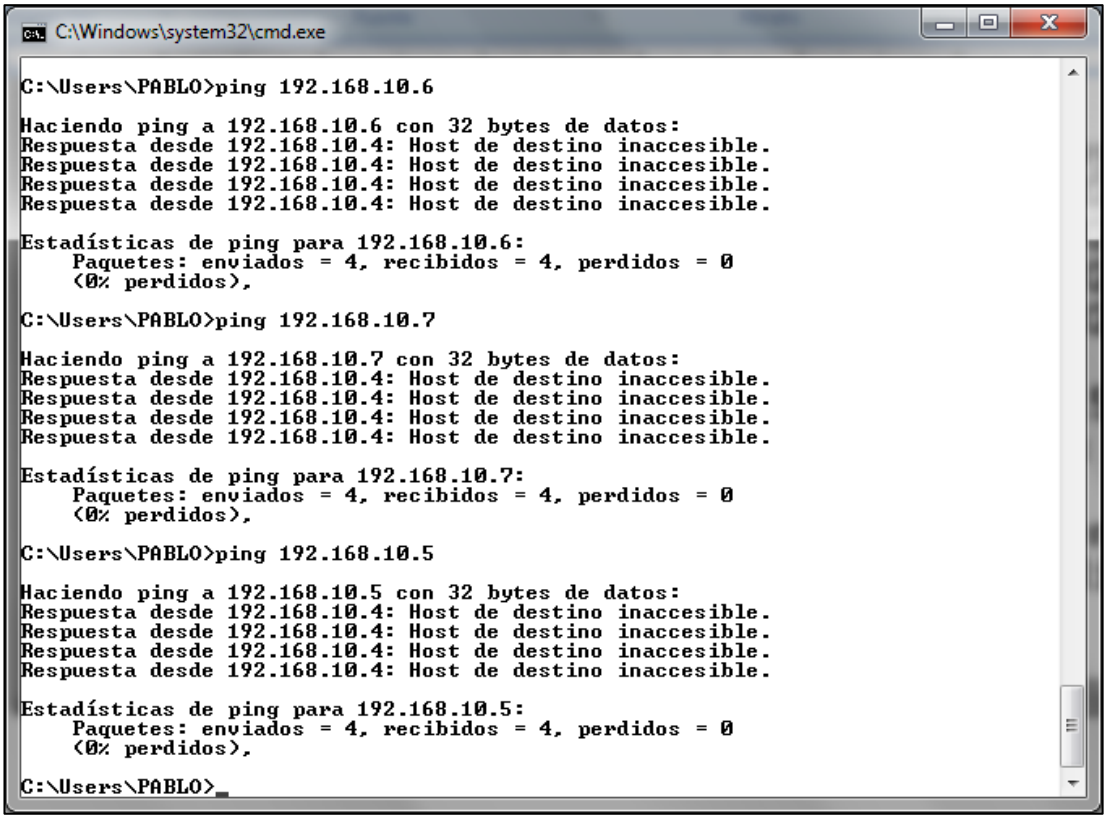


Figura 58. Resultado de la ejecución de los controladores POX y PyResonance desde PC_PABLO1 con el estado infected (TRUE)

En la coexistencia entre los controladores POX y PyResonance, con sus respectivos módulos dhcpserv y seleccionar_resonance, existen dos posibilidades de estados, con el estado TRUE, los únicos host que poseen conectividad lógica son el PC_DARIO y PC_DARIOMAC los cuales pertenecen al SW1, mientras que los hosts PC_PABLO y PC_PABLO1 que pertenecen al SW2 no poseen conectividad lógica entre ellos ni con los hosts del SW1. Se obtuvieron como resultados la latencia mínima de 0 (ms) y una latencia máxima de 0,526 (ms).

5.1.2.5.2. Controladores POX y PyResonance (FALSE)

En la figura 59 se observa que se envía el tipo de dato lógico FALSE al estado infected, y los demás datos son los mismos. Esto quiere decir que en la máquina de estados finita se condiciona a que retorne el flujo de paquetes

en SW2 por lo tanto existirá conectividad lógica entre todos los host ya sean del SW1 o del SW2.

```
mininet@mininet-vm:~/pyretic/pyretic/pyresonance$ python json_sender.py -n infected -l
False --flow="{switch=796915842132020}" -a 127.0.0.1 -p 50001

Flow_Str = {switch=796915842132020}

Data Payload = {'dstip': None, 'protocol': None, 'srcmac': None, 'tos': None, 'vlan_pcp
': None, 'dstmac': None, 'inport': None, 'switch': '796915842132020', 'ethtype': None,
'srcip': None, 'dstport': None, 'srcport': None, 'vlan_id': None}

ok
```

Figura 59. Envio del valor False al estado infected del controlador Pyretic mediante el cliente JSON

En las figuras 60, 61, 62 y 63 se observa los resultados de las pruebas al retomar el flujo de paquetes en SW2, es decir que todos los host de la red SDN volverán a tener conectividad lógica.

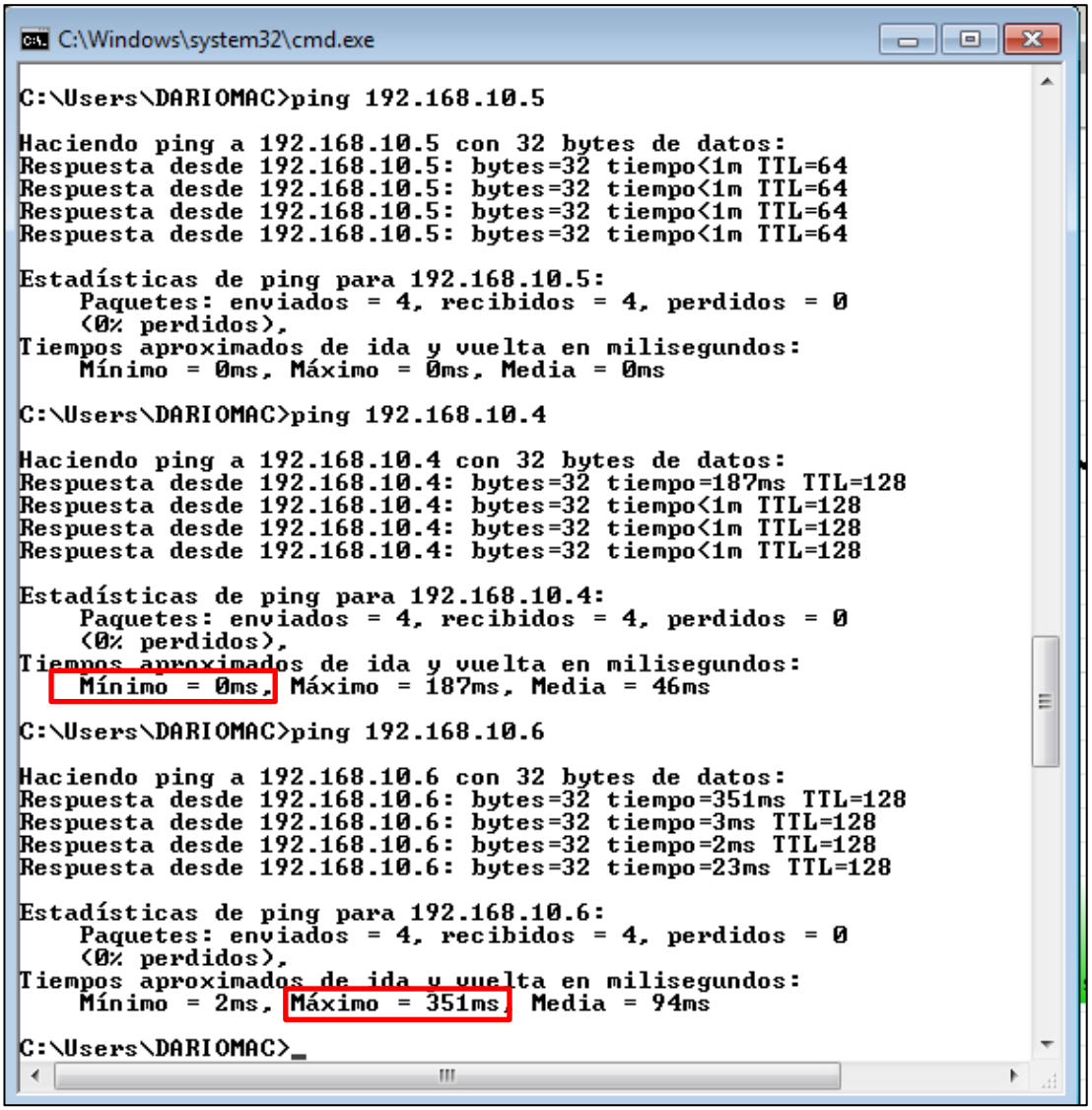


Figura 60. Resultado de la ejecución de los controladores POX y PyResonance desde PC_DARIOMAC con el estado infected (FALSE)

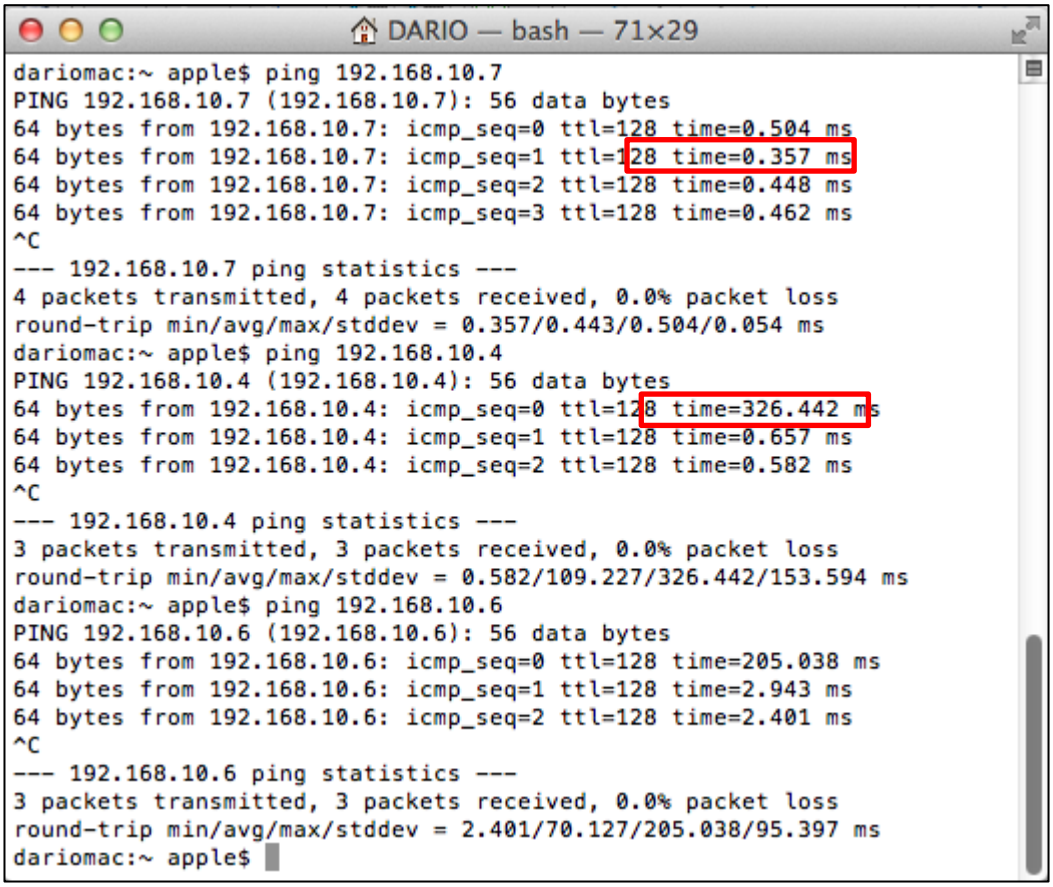


Figura 61. Resultado de la ejecución de los controladores POX y PyResonance desde PC_DARIO con el estado infected (FALSE)

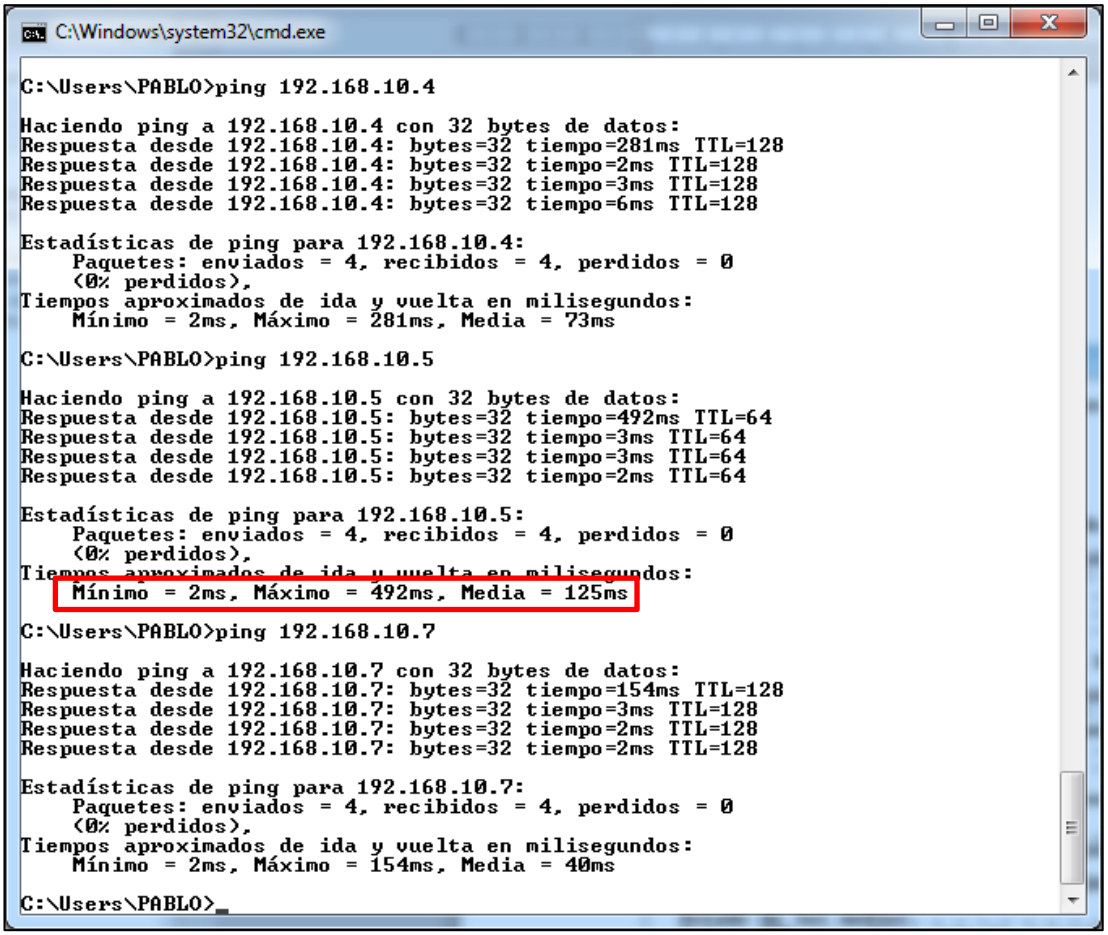


Figura 62. Resultado de la ejecución de los controladores POX y PyResonance desde PC_PABLO con el estado infected (FALSE)

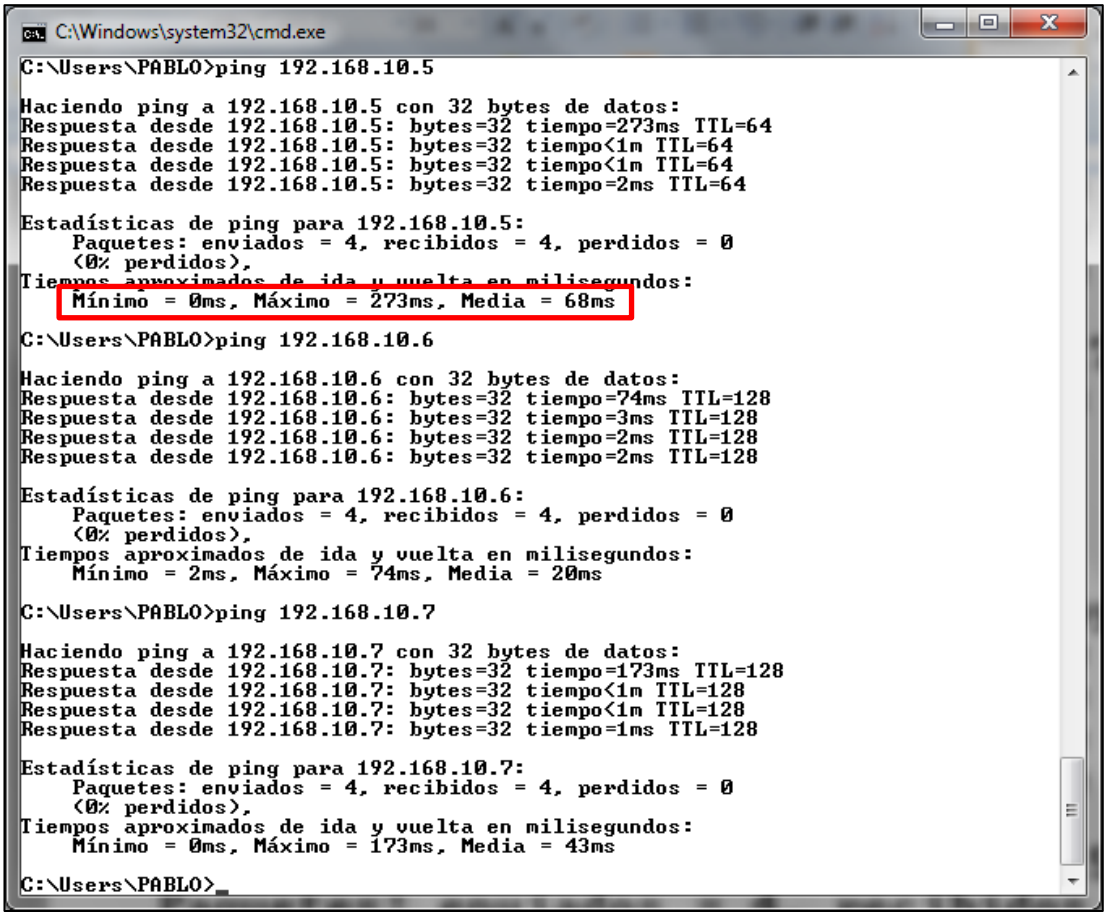


Figura 63. Resultado de la ejecución de los controladores POX y PyResonance desde PC_PABLO1 con el estado infected (FALSE)

En la coexistencia entre los controladores POX y PyResonance, con sus respectivos módulos dhcpserv y seleccionar_resonance, además con el estado FALSE, todos los host poseen conectividad física y lógica sin importar si pertenecen al SW1 o al SW2. Se obtuvieron como resultados la latencia mínima de 0 (ms) y una latencia máxima de 492 (ms).

5.2. Tabulación de resultados

5.2.1. Tabulación de resultados en la simulación de la red SDN

Tabla 4. ***Comparación de resultados en la simulación de la red SDN***

CONTROLADOR	MÓDULO	SWITCH (SW1/SW2)	HOST ORIGEN	HOST DESTINO	LATENCIA MÍNIMA (ms)	LATENCIA MEDIA (ms)	LATENCIA MÁXIMA (ms)	PING (SI/NO)
POX	dhcpserv, seleccionar	SW1	H1	H2	-	-	-	NO
		SW2	H3	H4	0,043	10,493	31,117	SI
		SW2	H4	H3	0,046	13,323	39,724	SI
		SW1 / SW2	H1	H3	-	-	-	NO
		SW1 / SW2	H2	H3	-	-	-	NO
		SW1 / SW2	H1	H4	-	-	-	NO
		SW1 / SW2	H2	H4	-	-	-	NO
Pyretic	seleccionar	SW1	H1	H2	-	-	-	NO
		SW2	H3	H4	0,031	23,66	70,897	SI
		SW2	H4	H3	0,05	0,177	0,387	SI
		SW1 / SW2	H1	H3	-	-	-	NO
		SW1 / SW2	H2	H3	-	-	-	NO
		SW1 / SW2	H1	H4	-	-	-	NO

Continua

		SW1 / SW2	H2	H4	-	-	-	NO
PyResonance	seleccionar_resonance	SW1	H1	H2	-	-	-	NO
		SW2	H3	H4	0,052	60,301	101,816	SI
		SW2	H4	H3	0,047	26,8	80,302	SI
		SW1 / SW2	H1	H3	-	-	-	NO
		SW1 / SW2	H2	H3	-	-	-	NO
		SW1 / SW2	H1	H4	-	-	-	NO
		SW1 / SW2	H2	H4	-	-	-	NO

LEYENDA	
LATENCIA MÍNIMA	LATENCIA MÁXIMA

5.2.2. Tabulación de resultados en la implementación de la red SDN

Tabla 5. ***Comparación de resultados en la implementación de la red SDN***

CONTROLADOR	MÓDULO	SWITCH (SW1/SW2)	HOST ORIGEN	HOST DESTINO	LATENCIA MÍNIMA (ms)	LATENCIA MEDIA (ms)	LATENCIA MÁXIMA (ms)	PING (SI/NO)
POX	dhcpserv, seleccionar	SW1	PC_DARIO	PC_DARIOMAC	-	-	-	NO
		SW2	PC_PABLO	PC_PABLO1	2	129	510	SI
		SW2	PC_PABLO1	PC_PABLO	2	27	101	SI
		SW1 / SW2	PC_DARIO	PC_PABLO	-	-	-	NO
		SW1 / SW2	PC_DARIOMAC	PC_PABLO	-	-	-	NO
		SW1 / SW2	PC_DARIO	PC_PABLO1	-	-	-	NO
		SW1 / SW2	PC_DARIOMAC	PC_PABLO1	-	-	-	NO
Pyretic	seleccionar	SW1	PC_DARIO	PC_DARIOMAC	-	-	-	NO
		SW2	PC_PABLO	PC_PABLO1	1	3	5	SI
		SW2	PC_PABLO1	PC_PABLO	2	37	138	SI
		SW1 / SW2	PC_DARIO	PC_PABLO	-	-	-	NO
		SW1 / SW2	PC_DARIOMAC	PC_PABLO	-	-	-	NO
		SW1 / SW2	PC_DARIO	PC_PABLO1	-	-	-	NO

Continua

		SW1 / SW2	PC_DARIOMAC	PC_PABLO1	-	-	-	NO
POX y Pyretic	dhcpserv (POX), seleccionar (Pyretic)	SW1	PC_DARIO	PC_DARIOMAC	0.319	0,43	0,566	SI
		SW1	PC_DARIOMAC	PC_DARIO	0	0	1	SI
		SW2	PC_PABLO	PC_PABLO1	2	31	120	SI
		SW2	PC_PABLO1	PC_PABLO	4	44	163	SI
		SW1 / SW2	PC_DARIO	PC_PABLO	5,959	298,849	883,409	SI
		SW2 / SW1	PC_PABLO	PC_DARIO	2	152	602	SI
		SW2 / SW1	PC_PABLO	PC_DARIOMAC	2	91	358	SI
		SW1 / SW2	PC_DARIOMAC	PC_PABLO	2	69	259	SI
		SW1 / SW2	PC_DARIO	PC_PABLO1	0.470	293,073	531,685	SI
		SW2 / SW1	PC_PABLO1	PC_DARIO	0	134	538	SI
		SW2 / SW1	PC_PABLO1	PC_DARIOMAC	0	82	331	SI
		SW1 / SW2	PC_DARIOMAC	PC_PABLO1	0	70	280	SI
PyResonance	seleccionar_resonance	SW1	PC_DARIO	PC_DARIOMAC	-	-	-	NO
		SW2	PC_PABLO	PC_PABLO1	2	86	340	SI
		SW2	PC_PABLO1	PC_PABLO	3	23	79	SI
		SW1 / SW2	PC_DARIO	PC_PABLO	-	-	-	NO
		SW1 / SW2	PC_DARIOMAC	PC_PABLO	-	-	-	NO
		SW1 / SW2	PC_DARIO	PC_PABLO1	-	-	-	NO
		SW1 / SW2	PC_DARIOMAC	PC_PABLO1	-	-	-	NO
POX y PyResonance (TRUE)	dhcpserv (POX), seleccionar_resonance (PyResonance)	SW1	PC_DARIO	PC_DARIOMAC	0,328	0,42	0,526	SI
		SW1	PC_DARIOMAC	PC_DARIO	0	0	0	SI
		SW2	PC_PABLO	PC_PABLO1	-	-	-	NO

Continua

		SW1 / SW2	PC_DARIO	PC_PABLO	-	-	-	NO
		SW1 / SW2	PC_DARIOMAC	PC_PABLO	-	-	-	NO
		SW1 / SW2	PC_DARIO	PC_PABLO1	-	-	-	NO
		SW1 / SW2	PC_DARIOMAC	PC_PABLO1	-	-	-	NO
POX y PyResonance (FALSE)	dhcpserv (POX), seleccionar_resonance (PyResonance)	SW1	PC_DARIO	PC_DARIOMAC	0,357	0,443	0,504	SI
		SW1	PC_DARIOMAC	PC_DARIO	0	0	0	SI
		SW2	PC_PABLO	PC_PABLO1	2	73	281	SI
		SW2	PC_PABLO1	PC_PABLO	2	20	74	SI
		SW1 / SW2	PC_DARIO	PC_PABLO	2,401	70,127	205,038	SI
		SW2 / SW1	PC_PABLO	PC_DARIO	2	125	492	SI
		SW2 / SW1	PC_PABLO	PC_DARIOMAC	2	40	154	SI
		SW1 / SW2	PC_DARIOMAC	PC_PABLO	2	94	351	SI
		SW1 / SW2	PC_DARIO	PC_PABLO1	0,582	109,227	326,444	SI
		SW2 / SW1	PC_PABLO1	PC_DARIO	0	68	273	SI
		SW2 / SW1	PC_PABLO1	PC_DARIOMAC	0	43	173	SI
		SW1 / SW2	PC_DARIOMAC	PC_PABLO1	0	46	187	SI

LEYENDA	
LATENCIA MÍNIMA	LATENCIA MÁXIMA

5.3. Análisis de resultados

Tabla 6. ***Comparación de resultados entre la simulación e implementación de la red SDN***

CONTROLADOR	RED SDN	LATENCIA MIN	LATENCIA MAX	LATENCIA PROM
POX	SIMULADA	0,043	39,724	19,8835
Pyretic		0,031	70,897	35,464
PyResonance		0,047	101,816	50,9315
POX	FÍSICA	2	510	256
Pyretic		1	138	69,5
POX y Pyretic		0	883,409	441,7045
PyResonance		2	340	171
POX y PyResonance		0	492	246

Según los resultados podemos deducir que en la simulación al trabajar con el controlador POX cuando se instala por primera vez una regla en la tabla de flujos existe una respuesta más rápida que con los controladores Pyretic y PyResonance, mientras que cuando una regla ya se encuentra instalada en la tabla de flujos el controlador Pyretic tiene una respuesta más eficiente que con los otros dos controladores. Por lo tanto trabajar con el controlador Pyretic resulta más eficiente a pesar que la latencia promedio del controlador POX sea más baja.

En la implementación al igual que en la simulación se obtiene un mejor rendimiento de la red utilizando el controlador Pyretic, ya que el nivel de procesamiento al instalar o actualizar una regla en la tabla de flujos es más rápido.

Cuando se hace coexistir dos o más controladores notoriamente el nivel de procesamiento es mayor, pero solo cuando se instala por primera vez una regla en la tabla de flujos, posterior a esto se puede observar que el tiempo de respuesta es prácticamente instantáneo.

5.4. Presupuesto referencial del prototipo

El presupuesto referencial tomando en cuenta los equipos detallados anteriormente se indican a continuación:

• Dos computadores con las características de hardware incluidas anteriormente, en uno de ellos se encontrará el servidor controlador como máquina virtual y las dos máquinas físicas serán para trabajar como hosts. Cada computador cuesta 1138 dólares americanos con cincuenta centavos, lo que da un total de 2277 dólares americanos incluido impuestos.

• Dos Router Mikrotik RB951Ui-2HnD que tienen las mismas características que otros equipos de distinta marca y su precio es más conveniente. Cada Router Mikrotik de esta serie cuesta 89 dólares americanos con cincuenta centavos, lo que da un total de 179 dólares americanos incluido impuestos.

• Rubros por instalación y configuración de los Router Mikrotik y el controlador: 20 dólares americanos la hora. El tiempo de capacitación, implementación y pruebas es de 200 horas. Lo que da un total de 4000 dólares americanos. Tomando en cuenta estas consideraciones, el presupuesto referencial asciende a 6456 dólares americanos. En el Anexo 6 se detallan las proformas en las cuales se ha basado este presupuesto referencial.

CAPITULO 6: CONCLUSIONES Y RECOMENDACIONES

6.1. Conclusiones

• Al finalizar el presente Proyecto se dispone del diseño e implementación de un prototipo de una red definida por software (SDN) para la Universidad de Fuerzas Armadas ESPE, compuesta por un servidor controlador, dos conmutadores y seis hosts.

• Las SDN como tecnología nueva incorporan una etapa en el desarrollo de las redes, permitiendo mejorar significativamente tanto la capacidad de gestión, como la escalabilidad y agilidad dentro de la red al emplear controladores.

• En el presente proyecto se proporciona una introducción a las redes definidas por software (SDN) y el protocolo Openflow, sus características generales, su arquitectura, ventajas y desventajas, sobresaliendo tendencias como la movilidad del usuario, la virtualización de servidores, y la necesidad para responder a las cambiantes condiciones en la red, que significan demandas sobre las redes.

• Los elementos a tener en cuenta para la selección de controladores SDN son: soporte OpenFlow, virtualización de red, funcionalidad de la red, escalabilidad, rendimiento, programación de red, confiabilidad, seguridad de la red, monitorización centralizada y visualización, fabricantes de controladores SDN, soporte de plataformas y procesamiento.

• En el proyecto se diseñó y programó los elementos de la Red mediante la simulación en el programa Mininet que permite simular redes SDN en un solo host o máquina virtual como si fuera una red física.

• Se implementó, controló y gestionó de forma centralizada los dispositivos que comprenden la topología planteada en el Proyecto ya que la arquitectura de las SDN tiene un servidor o grupo de servidores

controladores donde se centraliza la toma de decisiones, al contrario que en una red tradicional, lo que conlleva a políticas inconsistentes. Se puede tener un conjunto de servidores que actúen de manera coordinada como una entidad controladora, esto para evitar centralizar toda la toma de decisiones en un solo equipo físico, en base a esto las políticas de seguridad se puede controlar de manera más eficiente la red en caso de incursiones o ataques malintencionados porque existe una menor probabilidad de inconsistencia en las reglas de flujos.

- En el proyecto se realizó la implementación de la SDN para la ESPE, para la cual se utilizó dos conmutadores Mikrotik RB951Ui-2HnD, que fueron adaptados para que funcionen con el protocolo OpenFlow. Estos equipos de fábrica no vienen diseñados para soportar este protocolo pero si es posible actualizarlos.

- El controlador Pyretic es muy similar POX, se podría decir que es su evolución ya que tiene estructuras y métodos más complejos. Tanto en el controlador Pyretic como en POX se puede programar parámetros para tomarlos en cuenta o ignorarlos, esto hará posible un mayor control para un administrador y se evitaran inconsistencias.

- El controlador PyResonance, que tiene una gran aceptación a nivel comercial debido a su gran facilidad de uso, puede definir reglas de manera mucho más sencilla que el resto de controladores por lo que es considerado de alto nivel, es similar a los controladores POX y Pyretic pero se lo considera la evolución de ellos.

6.2. Recomendaciones

- Se recomienda tener conocimientos previos sobre el lenguaje de programación Python, ya que esto nos permite crear los métodos necesarios y utilizar de manera óptima los APIs existentes para cada controlador y así poder generar nuevos módulos.

- Se recomienda monitorear la aplicación, se lo puede hacer agregando una base de datos en la cual se almacenen las notificaciones de eventos ocurridos en cada equipo de la red, estos datos estén disponibles y puedan ser utilizados en un análisis posterior.

- Para evitar problemas de compatibilidad es necesario verificar que el software del servidor controlador sea compatible con los conmutadores elegidos para realizar la implementación. Esto es vital ya que caso contrario la SDN no podrá ser implementada.

- Se recomienda empezar usando la herramienta Mininet para estudiar las SDN. En esta herramienta se puede simular redes SDN y es posible familiarizarse con conceptos y procedimientos propios de OpenFlow y de las SDN. Es una herramienta muy práctica, ya que incluye todo lo necesario para trabajar con SDN en un nivel básico. No posee una interfaz gráfica y se sugiere previamente tener conocimientos básicos del sistema operativo Linux y de Python para la creación de topologías o componentes personalizados.

Bibliografía

Azodolmolky, S. (2013). *Software Defined Networking with OpenFlow.* Birmingham: PACKT.

CEPRA, CEDIA. (2014). *Proyecto: Implementación de un testbed para una SDN empleando la infraestructura de CEDIA.* Retrieved from http://cepra.cedia.org.ec/index.php/cepra-vii?id=127

Doxygen. (2015, Febrero 3). *Mininet Python API Reference Manual*. Retrieved from http://mininet.org/api/annotated.html

Ferro, G. (2014, Febrero 11). *OpenFlow and Software Defined Networking.* Retrieved from http://content.ipspace.net/get/Software%20Defined%20Networking.pdf.

Frenetic - lang. (2011, Diciembre). *Pyretic*. Retrieved from http://www.frenetic-lang.org/pyretic/

GitHub. (2013, Noviembre). *Pyresonance*. Retrieved from https://github.com/Resonance-SDN/pyresonance/wiki

GitHub. (2013, Septiembre 27). *PyResonance Tutorial*. Retrieved from https://github.com/Resonance-SDN/pyresonance/wiki/How-to-write-a-PyResonance-Task:-Step-by-step-tutorial

GitHub. (2014, Diciembre). *Tutorial Pyretic*. Retrieved from https://github.com/frenetic-lang/pyretic/wiki

InkaLinux. (2015). *Entendiendo al Winbox por dentro - Opciones Generales*. Retrieved from http://www.inkalinux.com/foros/showthread.php?107-4-Entendiendo-al-Winbox-por-dentro-Opciones-General

Kumar, R. (2013). *Software Defined Networking - a definitive guide .* Smashwords.

Lopez, D. R. (2012, Noviembre 9). *SDN (Software Defined Networking): cambiando de paradigma en la red*. Retrieved from http://blogthinkbig.com/sdn-software-defined-networking-cambiando-de-paradigma-en-la-red/

McCauley, M. (2014, Diciembre 26). *POX Wiki*. Retrieved from Open Networking Lab: https://openflow.stanford.edu/display/ONL/POX+Wiki

Mejía, D. (2014, Enero 13). Redes definidas por software. *Introducción a redes definidas por software*. Quito, Pichincha, Ecuador.

Nadeau, T., & Gray, K. (2013, Agosto). *Software Defined Networks.* O´REILLY.

NOXRepo. (2008). *NOX*. Retrieved from About POX: http://www.noxrepo.org/pox/about-pox/

Principia technologica. (2014, marzo). *Investigación de redes definidas por software.* Retrieved from https://principiatechnologica.files.wordpress.com/2014/03/sdn-principia-technologica-2.pdf

Principia technologica. (2014, Marzo). *Investigación de redes definidas por software.* Retrieved from https://principiatechnologica.files.wordpress.com/2014/03/sdn-principia-technologica-3.pdf

Python Software Foundation. (2015). *Python*. Retrieved from https://docs.python.org/3/

Redclara. (n.d.). *Indico*. Retrieved from http://eventos.redclara.net/indico/getFile.py/access?contribId=0&resId=0&materialId=slides&confId=197

Tiwari, V. (2013). *SDN and OpenFlow for Beginners.* Northville, MI.