ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

FACULTAD DE INFORMÁTICA Y ELECTRÓNICA

ESCUELA DE INGENIERÍA EN SISTEMAS

IMPLEMENTACIÓN DE UN PROTOTIPO DE RED DEFINIDA POR SOFTWARE PARA EL HOTSPOT-ESPOCH MEDIANTE UN CONTROLADOR BASADO EN OPENFLOW

Trabajo de titulación presentado para optar al grado académico de:

INGENIERO EN SISTEMAS INFORMÁTICOS

AUTORES: CATHERINE ELEANA YÁNEZ CARRERA

FABIÁN GUSTAVO GALLEGOS PILLAJO

TUTOR: ING. Msc. PATRICIO RENÉ MORENO

RIOBAMBA – ECUADOR

2015

AGRADECIMIENTO

Agradecemos a la Escuela Superior Politécnica de Chimborazo en especial a la Facultad de Informática y Electrónica con sus docentes quienes aportaron significativamente a nuestra formación profesional.

A las personas que de alguna u otra manera aportaron para llevar a cabo este trabajo de investigación: Ing. Patricio Moreno, Director de tesis por ser guía en el desarrollo de investigación, Ramón Fontes (Brasil) por su notable colaboración en nuestra investigación y a Juan Carlos Chico por compartir sus conocimientos.

A nuestras familias por el apoyo incondicional y por ser el motor fundamental.

FABIÁN GUSTAVO GALLEGOS PILLAJO

CATHERINE ELEANA YÁNEZ CARRERA

DEDICATORIA

El presente trabajo de investigación lo dedico a Dios por haber iluminado mi camino en los momentos más sinuosos de mi vida y a mis padres por creer en mí y ser mi apoyo incondicional.

CATHERINE ELEANA YÁNEZ CARRERA

Dedico el esfuerzo la dedicación y la perseverancia que me tomo realizar este trabajo a mi familia; el pilar fundamental que motivó cada momento para cumplir mis metas, a mi enamorada Paola por brindarme ese apoyo incondicional en los momentos adversos y cuando más la necesitaba; gracias amor. A mis amigos, colegas y esas personas que estuvieron día a día con esas palabras y la compañía necesarias para no decaer.

FABIÁN GUSTAVO GALLEGOS PILLAJO

FIRMAS RESPONSABLES Y NOTAS

NOMBRE	FIRMA	FECHA
ING. GONZALO NICOLAY SAMANIEGO **DECANO DE LA FACULTAD DE INFORMÁTICA Y ELECTRÓNICA**		
DR. JULIO SANTILLÁN CASTILLO **DIRECTOR DE LA ESCUELA DE INGENIERÍA EN SISTEMAS**		
ING. PATRICIO MORENO **DIRECTOR DE TESIS**		
ING.GERMANIA VELOZ **MIEMBRO DEL TRIBUNAL**		
COORDINADOR SISBIB - ESPOCH		

NOTA: _____

RESPONSABILIDAD DE LOS AUTORES

"Nosotros, Catherine Eleana Yánez Carrera y Fabián Gustavo Gallegos Pillajo, somos responsables de las ideas, doctrinas y los resultados expuestos en esta tesis, y el patrimonio intelectual de la Tesis de Grado pertenece a la Escuela Superior Politécnica de Chimborazo".

Fabián Gustavo Gallegos Pillajo

Catherine Eleana Yánez Carrera

ÍNDICE GENERAL

ÍNDICE DE ABREVIATURAS

API	Application Programming Interface
ARP	Address Resolution Protocol
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IP	Internet Protocol
LAN	Local Area Network
LLDP	Link Layer Discover Protocol
NAT	Network Address Translation
ONF	Open Networking Foundation
RTT	Round-Trip Time
SDN	Software Defined Network
TCP	Transmission Control Protocol
UDP	User Data Protocol
VLAN	Virtual Local Area Network

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ANEXOS

INTRODUCCIÓN

El presente trabajo de investigación se centra en el estudio de la tecnología SDN, sus elementos y funciones, se comparan los controladores basados en Openflow, con el fin de obtener uno, el más óptimo, e implementar la propuesta planteada.

En el Capítulo I Marco referencial, se detalla la problematización, la justificación teórica y práctica de la investigación, así como también los objetivos a alcanzarse y la hipótesis planteada.

En el Capítulo II Marco Teórico, contiene la teoría en la que se basa la investigación, se estudia el lenguaje de programación python, las redes SDN con su arquitectura y características, las redes virtuales Mininet, el Protocolo Openflow, los controladores y tipos de controladores y un estudio de los controladores a comparar.

En el Capítulo III Comparación de controladores en escenarios de prueba, se describe el escenario, ejecución de los controladores en la red virtual, comparación de los controladores a través de indicadores para obtener el controlador más óptimo.

En el Capítulo IV Creación del prototipo SDN, con el controlador más óptimo se realiza un prototipo SDN, se explica su programación, características y funciones y se realizan pruebas para la demostración de la hipótesis.

CAPÍTULO I

1. MARCO REFERENCIAL

Capítulo donde se explica la situación actual y real, el problema y parte de la solución para el caso HOTSPOT-ESPOCH. Se determina un cateo con distintas fuentes de información como el Director de la Dirección de Tecnologías de la Información y Comunicación y Técnicos de la misma donde se establece concretamente el problema que la red tiene, dado esto se plantean los objetivos de la investigación de donde se parte para obtener un estudio más detallada y aplicar una solución al problema, se determina el campo práctico y teórico, y por último se define una hipótesis a comprobar.

1.1 PROBLEMATIZACIÓN

La Escuela Superior Politécnica de Chimborazo es una institución educativa de categoría 'B' que cuenta con alrededor de 9000 estudiantes y aproximadamente 2000 entre docentes y personal administrativo. Los mismos que tienen acceso a la red inalámbrica de la ESPOCH a través de un usuario y contraseña individual ya registrados.

La infraestructura tecnológica es administrada por la Dirección de Tecnologías de la Información y Comunicación (DTIC) denominada de esta manera desde Julio del 2013, anteriormente conocida como Departamento de Sistemas y Telecomunicaciones (DESITEL), la información que se genera aquí se ha administrado a través de políticas de redes, programación y web.

La Escuela Superior Politécnica de Chimborazo cuenta con dos redes inalámbricas: ESPOCH y ESPOCH-WIFI. Actualmente el hotspot ESPOCH es gestionado por un controlador Wireless Lan Controller CISCO que posee licencias de la misma compañía.

El controlador del Hotspot ESPOCH cuenta con una licencia de funcionamiento CISCO perteneciente al año 2009 y que además posee un límite de conexiones de puntos de acceso. Esto significa que si se desea actualizar o adquirir más conexiones de puntos de acceso es necesario costear las licencias que comprenden los años subsiguientes hasta el actual [1].

En el caso de adquirir nuevos puntos de acceso para el controlador CISCO existente, estos deberán ser del mismo fabricante; por lo que es una demanda económica considerable razón por la cual no ha existido cambio alguno desde el año 2009.

La red inalámbrica ESPOCH cuenta con un número determinado de puntos de acceso que permiten la conexión de distintos dispositivos móviles (laptops, celulares, tablets, iPod, etc.), conociendo el tamaño de la población politécnica que aproximadamente cubren las 10000 personas y que únicamente 1/8 del ancho de banda de la red de la ESPOCH es destinado a la red inalámbrica, dichos puntos de acceso no soportan el gran número de dispositivos conectados simultáneamente por lo que genera congestión en la conexión [1].

Esta investigación no tiene antecedentes de haber sido realizada, comprobada o implementada anteriormente en la Escuela Superior Politécnica de Chimborazo; conclusión obtenida una vez revisada tanto la Biblioteca General y virtual de la institución. En investigaciones ajenas a la institución se corrobora que si existe información y un único precedente de haber sido consultada como en el caso de la Escuela Politécnica Nacional donde existe un proyecto de tesis que utiliza una SDN para un hardware físico [7].

El objeto de estudio se limita a la Escuela Superior Politécnica de Chimborazo a la red inalámbrica ESPOCH, el cual será el punto de inicio para desarrollar un prototipo SDN.

1.2 OBJETIVOS

OBJETIVO GENERAL

Implementar un prototipo de red definida por software para el sistema HOTSPOT-ESPOCH mediante un controlador basado en Openflow.

OBJETIVOS ESPECÍFICOS

- Estudiar características de los controladores POX, RYU y PYRETIC para el desarrollo de redes virtuales definidas por software.

- Comparar los controladores POX, RYU y PYRETIC para determinar el controlador que ofrezca mejores características en latencia, líneas de código y rendimiento de la red, usando software iperf y wireshark.

- Desarrollar el prototipo SDN para el sistema HOTSPOT - ESPOCH usando el controlador con las mejores características.

- Analizar y determinar a través de pruebas de rendimiento y latencia características del prototipo del HOTSPOT – ESPOCH actual y el prototipo SDN.

1.3 JUSTIFICACIÓN

Para el desarrollo de la presente investigación se realizará una justificación teórica y práctica del tema propuesto.

1.3.1 JUSTIFICACIÓN TEÓRICA

La red virtual generada por SDN (Software-Defined Networking) es la opción más recomendada para adaptar el aumento imparable de tráfico de datos y que hace que la red sea más programable; el crear redes virtuales con el uso de controladores puede balancear la carga, priorizar paquetes, administrar flujos, manejar tablas de enrutamiento y conmutación, etc.

La interfaz de programación se desarrolla en diferentes lenguajes que permiten la construcción de controladores; la mayoría de los controladores están basados en el lenguaje de programación PYTHON razón por la cual el análisis se enfocara a tres de ellos que son: POX, RYU y PYRETIC todos ellos bajo la directiva de OpenFlow.

Los controladores mencionados fueron escogidos por la facilidad y diseño que ofrece el lenguaje Python y además porque posee una librería propia y específica (pylibopenflow) para el manejo de Openflow [17].

PYTHON es un lenguaje de programación usado mayoritariamente en el campo investigativo, que ofrece programas más compactos y mucho más cortos a diferencia de otros lenguajes, razón por la cual es considerado como un lenguaje de muy alto nivel. Está basado en conjuntos de instrucciones que son ejecutados paso a paso y es de código abierto [13].

POX, RYU y PYRETIC son controladores para SDN basados en Python que esta soportado por Openflow que funciona sobre el sistema operativo Linux. Los controladores poseen API's (Interfaz de Programación de Aplicaciones) y un buen apoyo para virtualización de redes.

Estos controladores poseen varias y distintas configuraciones pero con la misma finalidad que es el virtualizar una red SDN, por lo que es necesario conocer el controlador con las características más apropiadas para gestionar y manejar la red inalámbrica HOTSPOT-ESPOCH.

1.3.2 JUSTIFICACIÓN PRÁCTICA

Para la práctica del desarrollo de esta investigación es necesario especificar una comparación de las características de los controladores POX, RYU y PYRETIC todos bajo la directiva Openflow; a través de un escenario de prueba donde los parámetros de evaluación se centren en latencia, líneas de código utilizado y rendimiento de red; y de esta manera conocer el más adecuado, que administre el desarrollo de un PROTOTIPO SDN de la red inalámbrica HOTSPOT-ESPOCH que contenga los dispositivos necesarios. El controlador está programado en Python y será configurado de acuerdo a las necesidades de la red inalámbrica ESPOCH.

A través del prototipo SDN se balancea la carga aprovechando el ancho de banda no utilizado y se destina a una actividad que requiera de más recursos en un momento dado; con el prototipo SDN se puede reemplazar las funciones de la red física actual destinada al sistema HOTSPOT-ESPOCH y de esta manera corroborar si la factibilidad de crear una red SDN es la mejor solución para el actual problema con respecto a la congestión de dispositivos móviles en la red inalámbrica ESPOCH.

1.4 HIPÓTESIS

La implementación de un prototipo de red definida por software (SDN) permitirá mejorar el rendimiento y gestión de dispositivos móviles del HOTSPOT-ESPOCH a través de un controlador basado en Openflow.

CAPÍTULO II

2. MARCO TEÓRICO

Se establece los temas de investigación que sirven de base para obtener la solución práctica, se investiga varias fuentes de información relacionadas con el tema propuesto, se busca en páginas oficiales, documentos web, articulo científicos, monografías, etc, de donde se obtiene solo la información más relevante sobre el tema. Cabe recalcar que de toda la información obtenida se usa únicamente datos que se acoplen a la situación de la solución propuesta.

2.1 LENGUAJE DE PROGRAMACIÓN PYTHON

Python es un lenguaje de programación, de alto nivel y de propósito general orientado a objetos con características que lo hacen fácil de leer y simple para implementar. Es de código abierto, por lo que es libre de usar hasta para aplicaciones comerciales. Python es ejecutable en varios sistemas operativos: Mac, Windows y sistemas Unix asi como también ha sido usado para Java y .NET y mininet [22].

Python tiene una librería pura para OpenFlow llamada: **pylibopenflow**. La biblioteca realiza las siguientes tareas [19]:

- Leer un archivo de cabecera OpenFlow y producir una cadena para el embalaje y desembalaje de mensajes OpenFlow.
- Generar código de contenedor con valores predeterminados para los mensajes OpenFlow.

- Simular un switch OpenFlow (en términos de mensajería)
- Hacer un controlador muy simple Openflow

2.1.1 CARACTERÍSTICAS DE PYTHON

- El código de Python es claramente definido y visible por lo que es fácil de leer.
- Tiene una amplia biblioteca estándar, portable y multiplataforma compatible con UNIX, Windows y Mac.
- Python se puede ejecutar en una amplia variedad de plataformas de hadware con una misma interfaz en todas las plataformas.
- Un archivo de python se procesa en tiempo de ejecución por el intérprete, no es necesario compilar el programa antes de ejecutarlo.
- Python admite estilo orientado a objetos o una técnica de programación que encapsula el código dentro de los objetos.
- Python permite añadir módulos de bajo nivel para el intérprete, estos módulos ayudan a que sus herramientas sean más eficientes.

Los Scripts escritos en Python tienen la extensión .**py** y se pueden analizar y ejecutar inmediatamente. También se pueden guardar como programas compilados con la extensión **.pyc**, que se utilizan a menudo en forma de módulos de programación que pueden ser referenciados por otros programas Python [24].

2.1.2 ELEMENTOS PRINCIPALES DE PYTHON

Python tiene muchas similitudes con Perl, C y Java. Sin embargo, hay algunas diferencias definidas entre los lenguajes, los elementos principales de python son los siguientes [23]:

- **Identificadores:** Nombre usado para identificar una variable, función, clase, módulo u otro objeto, python no admite signos de puntuación en los identificadores.
- **Palabras Reservadas:** Al igual que en otros lenguajes de programación python tiene palabras reservadas que cumplen una función específica, no se pueden utilizar como constante o variable o cualquier otros nombres de identificador. Todas las palabras clave de Python contienen sólo letras minúsculas: and, class, def, del, else, is, in, etc.
- **Líneas y Sangría:** Python no permite llaves para indicar bloques de código para las definiciones de clase, de función o control de flujo. Los bloques de código se indican mediante sangría de línea, misma que se aplica de manera rígida.
- **Declaraciones de líneas múltiples**: Las declaraciones en Python suelen terminar con una nueva línea. Python, sin embargo , permite el uso del carácter de continuación de línea (\ ), que indica que la línea debe continuar:
- **Comentarios en Python**: El signo de número (#) que no está dentro de un literal de cadena comienza un comentario. Todos los elementos después del # y hasta el final de la línea física son parte del comentario y el intérprete de Python los ignora.
- **Líneas en blanco**: En una sesión de intérprete interactivo, es necesario introducir una línea física vacía para terminar una sentencia de múltiples líneas.
- **Varias sentencias en una sola línea:** El punto y coma (;) permite que varias instrucciones en una sola línea.
- **Suites de python:** Las suites en python es un grupo de declaraciones individuales, que hacen un solo bloque de código, una suite tiene líneas de cabecera que comienzan la declaración (con la palabra clave) y terminan con dos puntos (:).

2.2 REDES DEFINIDAS POR SOFTWARE

Las Arquitecturas de red tradicionales están mal adaptadas para satisfacer las necesidades de las empresas de hoy en día, gracias a la industria encabezada por la Fundación de red

abierta (ONF) , las redes definidas por Software (SDN) son la transformación de las redes de arquitectura.

En la arquitectura de SDN, los planos de control y de datos están separados para convertirse en una red totalmente programable, la inteligencia de red y el estado de la misma son lógicamente centralizados, y la infraestructura de red se simplifica en el uso de las aplicaciones. Como resultado, las empresas y las compañías ganan programación, automatización y control de la red, lo que les permite construir redes altamente escalables y flexibles que se adaptan fácilmente a las necesidades del negocio [20].

Las SDN están diseñadas para combinar la red en la nube. Esta evolución de la red moderna permite hacer una red más elástica, ágil y dinámica, para mantener el ritmo cambiante de hoy en día [11].

La ONF es un consorcio industrial sin ánimo de lucro que está liderando el avance de la SDN y estandarización de los elementos críticos de la arquitectura SDN como el protocolo OpenFlow. OpenFlow es la primera interfaz estándar diseñada específicamente para SDN, proporcionando alto rendimiento y un control de tráfico eficiente a través de dispositivos de red de múltiples proveedores [20].

2.2.1 ARQUITECTURA DE LAS SDN

La base de la arquitectura SDN es un controlador el mismo que a través de su programación permite administrar la red y manipular los dispositivos de interconectividad.

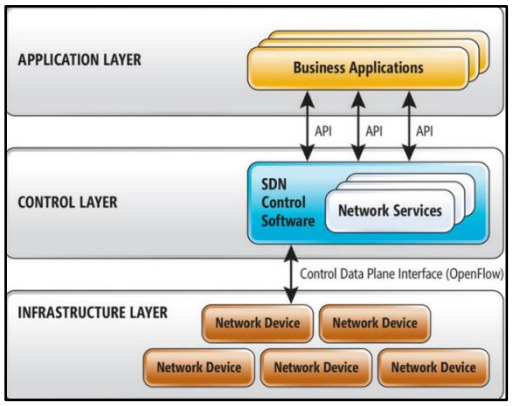
La arquitectura SDN consta de tres capas distintas:

- ✓ **La capa de aplicación**: Consta de las aplicaciones empresariales finales que consumen los servicios de comunicaciones SDN.

- ✓ **La capa de control**: Proporciona la función de control que supervisa el comportamiento de reenvío de red a través de una interfaz abierta, desde aquí opera el controlador.
- ✓ **La capa de infraestructura**: Consta de los elementos de red y los dispositivos que proporcionan la conmutación de paquetes y de reenvío.

En la **FIGURA N° 1** se muestra la arquitectura SDN donde la capa de control es la más importante ya que desde ahí se administra y se comunica con los dispositivos de conectividad a través del uso del protocolo Openflow.

FIGURA N° 1: ARQUITECTURA SDN



Fuente: http://www.ramonmillan.com/tutoriales/sdnredesinteligentes.php

La **FIGURA N° 1** representa una vista lógica de la arquitectura SDN. La inteligencia de red es centralizada en el controlador SDN basados en software, que mantienen una visión global de la red. Como resultado, la red conjuga a las aplicaciones y motores de políticas como un solo interruptor lógico. Con las SDN, se simplifica en gran medida el diseño de la red y la operación. SDN también simplifica el trabajo de los dispositivos de red, pues ya no tienen que entender y procesar miles de normas, sino simplemente aceptar instrucciones del controlador SDN [21].

2.2.2 CARACTERÍSTICAS Y VENTAJAS DE LA SDN

Las características principales de las redes definidas por software son:

- ✓ Directamente programables: Se puede programar directamente el controlador desde el cual se manipula toda la red.
- ✓ Ágil: Permite a los administradores de red ajustar dinámicamente el flujo de tráfico de toda la red para satisfacer las necesidades cambiantes.
- ✓ Gestiona de manera centralizada: La inteligencia de la red es centralizada en software basado en controladores SDN que mantienen una visión global de la red.
- ✓ Programación configurada: SDN permite a los administradores de red configurar, administrar, asegurar y optimizar los recursos de red muy rápidamente a través de las aplicaciones SDN, que son escritas por los propios programadores que utilizan lenguajes de programación comunes.
- ✓ Basados en estándares abiertos y de proveedor neutral: Cuando se implementa a través de estándares abiertos, SDN simplifica el diseño de la red y la operación porque las instrucciones son proporcionados por los controladores SDN en lugar de múltiples dispositivos y protocolos específicos del fabricante [14].

2.2.3 OBJETIVO PRINCIPAL DE LA SDN

A través de las SDN los operadores y los administradores pueden configurar la red mediante programación esta abstracción simplifica la red, en lugar de tener que codificar manualmente decenas de miles de líneas de configuración dispersos entre miles de dispositivos. Además, aprovechando la inteligencia centralizada del controlador SDN, se puede alterar el comportamiento de la red en tiempo real y desplegar nuevas aplicaciones y servicios de red en cuestión de horas o días, en lugar de las semanas o meses necesarios hoy en día. Al centralizar el estado de la red en la capa de control, la SDN ofrece a los

administradores de red la flexibilidad para configurar, administrar, proteger y optimizar los recursos de red a través de programas automatizados SDN dinámicas.

Además de la abstracción de la red, las arquitecturas SDN admiten un conjunto de APIs que hacen posible la implementación de servicios de red comunes, incluyendo enrutamiento, seguridad, control de acceso, la ingeniería de tráfico, calidad de servicio, el procesador y la optimización del almacenamiento, la energía de uso, y todas las formas de gestión de políticas para satisfacer los objetivos del negocio [9].

2.3 MININET

Es un simulador de red virtual. Permite la ejecución de switches, routers y enlaces en un solo núcleo de Linux. Utiliza una virtualización ligera para hacer un solo sistema parecido a una red completa. Un anfitrión MiniNet se comporta como una máquina real.

Los programas que la MiniNet ejecuta, pueden enviar paquetes por lo que parece una verdadera interfaz Ethernet, con una velocidad y demora de enlace determinada. Los paquetes se procesan por lo que parece un verdadero switch de Ethernet, un enrutador con una determinada cantidad de colas. Cuando dos programas, como un cliente y un servidor iperf, se comunican a través de la MiniNet, el rendimiento medido debe coincidir con el de dos máquinas reales.

En síntesis una MiniNet ejecuta: conmutadores, enlaces, y controladores como reales, los mismos que se crean utilizando software en lugar de hardware y en mayor parte su comportamiento es similar al de los elementos de hardware.

2.3.1 CARACTERÍSTICAS DE LA MININET

- ✓ Es rápida: La puesta en marcha de una red simple tarda sólo unos segundos.
- ✓ Puede crear topologías personalizadas: Permite crear las topologías de red de acuerdo a las necesidades y requerimientos, con un solo switch, grandes topologías tipo Internet, un centro de datos o cualquier otra cosa.
- ✓ Ejecuta programas reales: cualquier cosa que se ejecuta en Linux está disponible para que ejecute una MiniNet, desde los servidores de Internet se puede hacer uso de herramientas de monitoreo como: Wireshark.
- ✓ Permite personalizar el reenvío de paquetes: Los switch de MiniNet son programables usando el protocolo OpenFlow. Los diseños de SDN que se ejecutan en una MiniNet se pueden transferir fácilmente a los switches OpenFlow para el envío de paquetes a la velocidad actual.
- ✓ Permite compartir y replicar los resultados: cualquier persona con una computadora puede ejecutar el código una vez que haya sido empaquetado.
- ✓ Se puede utilizar fácilmente: Se puede crear y ejecutar experimentos MiniNet escribiendo simples scripts de Python.
- ✓ Es un proyecto de código abierto: Permite examinar, modificar el código fuente, corregir los errores, problemas de archivos, etc [5].

2.3.2 LIMITACIONES DE LA MININET

- ✓ MiniNet destina un solo núcleo de Linux para los hosts virtuales que está usando.
- ✓ MiniNet usa un solo sistema por lo que existen limitaciones de recursos la capacidad del CPU tienen que ser equilibrados y compartidos entre los hosts virtuales y los switch.
- ✓ Los hosts de MiniNet tiene un mismo sistema de archivos y el espacio PID lo que significa que se debe tener cuidado cuando se ejecuten demonios y evitar matar a los procesos erróneos por error.

- ✓ MinINet no hace NAT fuera de la caja por lo que sus anfitriones no pueden hablar directamente a Internet, solo puede hacerlo si se proporciona un medio para que lo hagan.
- ✓ MiniNet no escribe el controlador Openflow, si se necesita enrutamiento personalizado se debe encontrar o desarrollar un controlador con las características deseadas [25].

2.3.3 INTRODUCCIÓN AL USO DE MININET

Para la ejecución de la MiniNet es un requerimiento la instalación de un X-Manager (solución de conectividad), existe un X-Manager para cada sistema operativo:

- ✓ Linux: XQuartz
- ✓ Windows: Xming

Para el uso de un cliente ssh (Putty) es necesario activar el acceso para la sesión ssh en el X-Manager que se esté utilizando: X11.

Existen varias asignaciones de comandos (palabras de referencia) dentro de algunas terminales que usa una MiniNet:

Palabras de referencia:

- ✓ Hx: Host
- ✓ S: Switch
- ✓ Cx: Controlador

Terminales:

- ✓ $: comando en shell
- ✓ #:comando como root
- ✓ Mininet>: comandos dentro de mininet

Una MiniNet se crea y manipula a través de comandos en los diferentes terminales:

- ✓ $Sudo mn: inicia la mininet con una topología, por default se crea i switch, t controlador y 2 host

MiniNet a través de un solo comando permite ejecutar la creación de una red como se muestra en la **FIGURA N° 2**

FIGURA N° 2: COMANDO MININET

Fuente: http://mininet.org

Dentro de la MiniNet:

- ✓ Nodes y net: Para ver los nodos y sus enlaces.
- ✓ Dump: información sobre los nodos.

Sintaxis:

- ✓ mininet>[nodo]comando: Si se coloca un nodo frente a un comando, se indica que el comando se ejecuta a ese nodo, se puede sustituir el nombre del nodo por su IP.
- ✓ mininet>h2 ping –c3 h1: Hace ping entre dos host indicando el número de paquetes.

Comandos dentro de la mininet:

- ✓ mininet>exit: Salir de mininet
- ✓ mininet>xterm [nodo]: Abre un terminal para el nodo.
- ✓ mininet>link [node1][node2][up or down]: Crea o elimina un link entre dos nodos.
- ✓ mininet>pingall: Prueba conectividad entre dos nodos
- ✓ mininet>help: Ayuda, muestra lista de comandos de mininet

- ✓ mn –c: Limpia la topología. [6]

2.3.4 CREACIÓN DE UNA TOPOLOGÍA DE UNA MININET

En MiniNet se puede crear varias topologías entre las más usadas están:

- ✓ single
- ✓ linear
- ✓ tree
- ✓ custom.

La forma de crear cada topología es la siguiente:

- ✓ $sudo mn --topo single,N: Un switch conectado a N hosts
- ✓ $sudo mn --topo linear,N: Cada switch se conecta con otro de forma linear, y cada switch tiene un host conectado.
- ✓ $sudo mn --topo tree,depth=n,fanout=m: Crea una topología de árbol con Profundidad N y anchura M.
- ✓ $sudo mn --custom mytopo.py --topo mytopo: Para topologías generales es necesario crear un archivo en python con su topología.

Para usar programas dentro de MiniNet:

- ✓ $mininet>iperf: Llama al programa iperf que hace pruebas con redes informáticas, funciona después de hacer ping entre los host de una red.
- ✓ $ sudo wireshark &: Llama a wireshark, el mismo que permite analizar los protocolos usados [6].

2.4 OPENFLOW

Openflow provee un protocolo abierto para ejecutar, programar o controlar el comportamiento de envío en los diferentes switches y routers que intervienen en una red definida por software (SDN). Openflow permite también programar tablas de flujos para poder ser enviados a los switches o routers sin la necesidad de que existan personas que se encarguen de explicar el funcionamiento del trabajo interno de los dispositivos presentes en una red. Con Openflow el ancho de banda, latencia y los pasos largos (saltos) entre switches y routers; se puede diseñar "networking" independiente y además programable sin tener en cuenta políticas de creación de fabricantes.

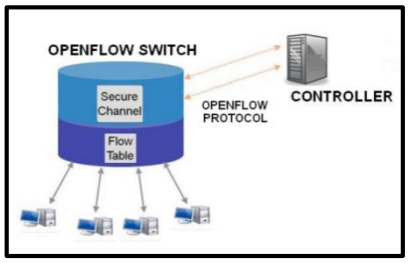
Openflow permite adaptar a la red a tal punto del no tan solo tener el plano de control y la decisión de envío en un solo dispositivo (comúnmente declarado en un switch), más bien, define un estándar sobre los cuales estas dos definiciones pueden ser abstractas. Esto permite el control lógico de los datos, los cuales son enviados a un "controlador" externo que se comunica con el uso de un protocolo Openflow (Openflow Protocol) para manejar la red de acuerdo a situaciones específicas.

2.4.1 ARQUITECTURA OPENFLOW

La arquitectura de Openflow consta de tres partes importantes: El controlador Openflow, el dispositvio Openflow (Switch) y el Protocolo Openflow. El enfoque Openflow considera a un controlador centralizado como aquel que configura todos los dispositivos.

En la **FIGURA N° 3** se aprecia la arquitectura de como Openflow se adapta y funciona a través de una red virtual con el Controlador, Switch Openflow, y el propio Protocolo Openflow, presentando así la arquitectura general de cómo funciona una red SDN con una directiva Openflow.

FIGURA N° 3: ARQUITECTURA OPENFLOW



Fuente: http://electivaredesavanzadas.blogspot.com/2015/02/openflow.html

Openflow entre otras y muchas cosas permite:

- Más flexibilidad y control de simulación y software.
- Más velocidad, escalabilidad, rentabilidad y confiabilidad.
- Vendedores no necesitan explicar implementaciones.
- Tiene una visión más global de la red.

2.4.2 FUNCIONALIDAD DE OPENFLOW

La tecnología de Openflow funciona a partir de tres partes iniciales: Una tabla de flujo que está dentro o instalado en el switch, un controlador y un Protocolo Openflow que se comunica , opta por políticas en los flujos y habla correctamente con dicho dispositivo.

El switch Openflow consiste en una tabla de flujo que contiene las entradas de flujo, para desempeñar una búsqueda de paquetes y así realizar un envío por un canal seguro hacia el controlador. Los mensajes Openflow son los que se intercambian entre el switch y el controlador [4].

Para obtener una búsqueda segura el switch es capaz de realizar decisiones de búsqueda para los paquetes que están por entrar accionando una búsqueda simple en las tablas de flujo de entrada.

Para cada paquete entrante, el switch va a través de la tabla de flujo para encontrar una entrada coincidente. Si en verdad encuentra esa entrada coincidente, el switch envía el paquete basado en la acción asociada con esa entrada de flujo en particular.

Para entender un poco mejor de lo que se trata estas tablas de flujo y sus entradas las explicaremos a continuación:

1. Campos de Cabecera: Estos campos contienen diez tuplas o filas que identifican el flujo **TABLA N° 1.**

TABLA N° 1: CAMPOS DE CABECERA OPENFLOW

Ingress Port	Ether Source	Ether Dst	Ether Type	VLAN Id	IP Source	IP Dst	IP Proto	Src Port	Dst Port

Fuente: http://searchdatacenter.techtarget.com/es/definicion/OpenFlow

- Ingress Port: Puerto de Ingreso hace referencia al puerto de ingreso comienza en 1 sea un puerto físico o virtual de 32 bits.
- Ether Source: Dirección MAC de partida de 48 bits.
- Ether Dst: Dirección MAC de destino de 48 bits.
- Ether Type: Tipo de mensaje del paquete Openflow que se envía de 16 bits, por ejemplo: ICMP, IP, HTTP.
- VLAN Id: Identificador de VLAN de 12 bits
- IP Source: IPV4 de Origen de 32 bits.
- IP Dst: IPV4 de Destino de 32 bits.
- IP Proto: Protocolo IP de 8 bits, todos los IPV4 y los paquetes ARP
- Src Port: Puerto de Origen TCP de 16 bits
- Dst Port: Puerto de Destino TCP de 16 bits por ejemplo: puerto 80(http), 22 (ssh).

2. Contadores: Estos son usados para propósitos estadísticos, para guardar la trayectoria del número de paquetes y bytes por cada flujo y el tiempo que trascurre desde el inicio del flujo
3. Acciones: Especifica la manera en el cual los paquetes son y serán procesados. Una acción puede constar de dos partes :
    - Enviar un paquete o paquetes dados, para después llenar opcionalmente sus campos de la cabecera.
    - Soltar el paquete
    - Enviar el paquete al controlador

Un controlador es una unidad central o núcleo, que mueve la funcionalidad del plano de control de la red Openflow. El controlador otorga una interfaz para crear, modificar y controlar los flujos de tablas de un switch. Corre típicamente sobre un servidor en red y que puede ser parte de un conjunto entero de switches Openflow en la red, uno para cada switch o uno por cada conjunto de switches.

Por tanto la funcionalidad de control de la red puede estar completamente o localmente centralizada de acuerdo a como se lleve a cabo la delegación de la gestión del switch para controladores [4].

Lo que se requiere, sin embargo, es que si existe más de un controlador procesando, de igual manera éstos tengan una vista de la topología de la red en cualquier momento dado. La vista de la red incluye entre otras cosas el nivel de la topología del switch, la ubicación de usuarios, hosts, middleboxes y otros elementos de y servicios de la red.

2.4.3 PROTOCOLO OPENFLOW

Es extremadamente necesario que un controlador implemente un protocolo Openflow para su comunicación, y por medio de mensajes, para su posterior envió en la red. Los mensajes

de un controlador puede que necesiten que sean respondidos por los switches que intervienen en la red. Los cuales pueden clasificarse de la siguiente forma:

- Mensajes de Características: Comúnmente desarrollada después de que exista un canal de establecimiento entre el controlador y switch Openflow, el controlador requiere a un switch un pedido de conexión para observar las características y capacidades del mismo y así el controlador pueda gestionar de mejor manera la red.
- Mensajes de Configuración: Un controlador es apto y capaz de establecer y consultar parámetros de configuración al switch. El switch responde a la solicitud propuesta por el controlador.
- Mensajes de Modificación de Estados: Son considerados mensajes enviados por el controlador para gestionar el estado del switch. Su función consiste en agregar o suprimir entradas en la tabla de flujo de los switches, es decir, añadir o eliminar reglas.
- Mensajes de Lectura de Estado: Son mensajes enviados por el controlador para leer el estado y recolectar datos estadísticos del controlador.
- Mensajes Packet-out: Esos mensajes son usados para enviar paquetes al switch. Al no existir una coincidencia entre el paquete y alguna entrada de la tabla de flujo del switch, ese paquete es reenviado por completo al controlador (Packet-in) y éste a su vez envía paquetes de salida usando Packet-out con el conjunto de acciones respectivos.
- Mensajes de Barrera: Estos mensajes son utilizados por el controlador para garantizar si las relaciones de mensajes fueron cumplidas y así recibir notificaciones de operaciones concluidas.

El canal seguro es la interfaz que conecta cada switch Openflow al controlador. A través de esta interfaz el controlador intercambia mensajes con los switches para configurar y administrar dichos dispositivos [15].

2.4.4 TIPOS DE COMUNICACIÓN OPENFLOW

Openflow provee de un protocolo de comunicación entre el proceso del controlador y los Switches Openflow. Existen tres tipos de comunicación soportados por el Protocolo Openflow. La comunicación Controlador-a-Switch, la comunicación AsíncronA y la comunicación Sincrónica.

- Comunicación Controlador-a-Switch: Son iniciados por el controlador como origen y no siempre espera o requiere una respuesta por el switch. A través de esta comunicación el controlador configura el switch, gestiona la tabla de flujo y obtiene información acerca del estado de dicha tabla o de las capacidades del switch a cualquier momento dado.
- Comunicación Asíncrona: Estos son enviados sin ser solicitados desde el switch hacia el controlador y denota un cambio en el switch o en el estado de la red. Este cambio también es llamado Evento.

  Uno de los más comunes es el evento packet-in (paquete adentro) el cual se dispara siempre y cuando un paquete no tiene una coincidencia de entrada en el switch. Si esto es lo que sucede el evento o mensaje packet-in es direccionado hacia el controlador, que contiene una parte o el paquete completo y así el controlador es el que inspecciona y decide qué tipo de flujo se le debería optar o establecer.
- Comunicación Sincrónica: Estos son enviados sin ser solicitados desde cualquiera de los dos partes (Switch o Controlador). Esta comunicación puede ser utilizada para diagnosticar problemas ocurrentes en la conexión que se realiza entre Switch y Controlador [3].

2.4.5 COMPORTAMIENTO DEL CONTROLADOR

Los controladores Openflow ocasionalmente en el caso de enviar paquetes al switch, que con el surge y siempre existe comunicación, puede comportarse dos maneras. De manera Reactiva y Proactiva.

- Comportamiento Proactivo: Es donde el controlador obtiene entradas de flujos dentro del switch basados en la configuración del usuario y topologías de red. El controlador realiza un pre-llenado a la tabla de flujos en cada switch. Este comportamiento tiene un tiempo adicional de cero porque la regla de envío ya está establecida. Ahora, si el switch pierde la conexión con el controlador, no es posible que se interrumpa el tráfico.
- Comportamiento Reactivo: El primer paquete de flujo recibido por el switch, dispara al controlador para insertar las entradas de flujo en cada switch Openflow de la red. Este comportamiento tiene el más eficiente uso en la tabla existente de memoria de flujo, pero cada nuevo flujo causa un pequeño tiempo adicional. Finalmente, con una dependencia fuerte del controlador, si el switch pierde la conexión, pues no se puede enviar el paquete [12].

2.5 TIPOS DE CONTROLADORES BASADOS EN OPENFLOW

Un controlador es el responsable de tomar las decisiones adicionando o removiendo entradas en una tabla de flujo de acuerdo a reglas o políticas de encaminamiento. La inteligencia de la red es la que se destina a este componente de la arquitectura SDN y que ejecuta un software que se encarga de llevar a cabo todas sus actividades.

Algunos controladores se basan o están programados en algunos o varios lenguajes de programación siendo C, Python, Java, Ruby, los que demandan mayor posicionamiento en

el mercado al momento de querer hablar sobre desarrollo de Controladores basados en Openflow.

Existen varios controladores en diferentes lenguajes de programación los mismos que se listan a continuación:

- Beacon: Basado en el Lenguaje Java, se puede implementar en Linux, Windows, Android y Mac.
- NOX: Programado en C y en python, implementado únicamente en la pltaforma Linux, controlador actualmente obsoleto.
- Maestro: Programado en Java, soportado en plataforma Windows, Mac y Linux
- Trema: Programado en Ruby y C, soportado en plataforma Linux, incluye un emulador para pruebas
- RouteFlow: Soportado en plataforma Linux, tiene licencia Apache.
- Floodligth: Controlador con licencia Apache y programado en Java.

La mayoría de controladores definidos en el mercado actual están basados en un lenguaje de programación apto y adecuado para administrar redes, es este el caso de Python, que ha obtenido una fuerte acogida al momento de programar aplicaciones en red y no dado de menos en lo que a la Arquitectura SDN se tratase.

Los controladores más conocidos en el ámbito de este lenguaje de programación Python son: POX, RYU, PYRETIC. Dado que los controladores manejan el plano de control, el uso de un controlador específico en una red influye en el rendimiento es decir en la variación de velocidad con que se transmite cierto número de paquetes e un período de tiempo, de la misma manera influye en la latencia de la red, el uso de un controlador cambia el total de retardos temporales que se producen en una red, el tiempo que le toma a un bit en viajar de un host a otro, es decir varia al tiempo que se demora en la transmisión de paquetes, estos tres controladores permiten el uso de herramientas de monitoreo de red como: iperf, dpctl, etc.

2.5.1 CONTROLADOR POX

Como se citó anteriormente existen varios tipos de controladores que manejan el funcionamiento de una red. Una manera de como desenvolver el paradigma de SDN depende del lenguaje de programación en la que dicho controlador haya sido creado.

El controlador POX es una alternativa que surgió para sustituir o mejorar un controlador ya existente, se lo conoce como NOX y se lo llama también como el hermano mayor de POX; éste controlador NOX fue desarrollado en su mayoría en C++. POX a diferencia de su otro acompañante NOX es desarrollado en Python y que a ésta además se le puede entregar acciones al controlador para que ejecute actividades de acuerdo a la situación necesaria en la red.

El controlador POX es uno de los controladores más utilizados desde la creación de ONF ya que fue la primera alternativa que surgió para las redes SDN. Como primera opción se trató de mejorar y perfeccionar este controlador por lo que su investigación es notable frente a otros controladores.

Soporta la interacción con dispositivos Openflow y es usado para crear reglas de encaminamiento en una red definida por SDN, depurar redes SDN, crear redes virtuales, programación de módulos y controladores, etc. Uno de los fines de quienes crearon POX es definir una API estable para los siguientes años.

Algunas características de POX que se pueden encontrar:

- Interfaz Openflow denominada “Pythonic” referente al ambiente de programación Python.
- Soportado en Sistemas operativos: Linux, Windows, Mac OS entre otras, gracias al uso de un entorno de ejecución “PyPy” que ejecuta programas basados en Python y que generalmente ahorra más tiempo de ejecución que el intérprete común de Python llamado CPython. Es portable y de fácil instalación en el paquete de POX al momento de ejecutarlo.

- Componentes reusables en ejemplos añadidos en su directorio principal, topologías, direccionamiento, etc.
- Está basado en el lenguaje C++ y Python, este último es el más usado.
- Permite englobar la creación de la red virtual mininet: topología, controlador y ejecución dentro de un único archivo.
- Mayores prestaciones a diferencia de NOX que actualmente está devaluado y descontinuado, tanto NOX y POX están en condiciones de ser excelentes contrincantes al momento de desarrollar soluciones de redes virtuales, pero en el lenguaje Python, es POX el que posee la ventaja potencial de uso [8].

Componentes de POX

POX mantiene su invocación principal con el uso de la instrucción "/pox.py", uno de los archivos principales para correr o ejecutar cualquier acción o archivo que se genere a través de Python. POX provee de componentes ya ejecutables y programados por sus desarrolladores. Los componentes que se pueden citar son los siguientes:

- forwarding.l2_learning: Éste complemento hace al switch OpenFlow manejarse como un switch de capa 2. Hace que el switch aprenda direcciones, además los flujos que se aplican son exactamente coincidentes para la mayoría de campos.
- forwarding.l3_learning: Complemento que es parecido con el anterior, salvo que éste actúa y posee funcionalidades como el switch de capa 3. Aprende y escucha sobre las IP que encuentra y maneja peticiones ARP (Protocolo de Resolución de Direcciones).
- samples.spanning_tree: Gracias a este componente se puede crear una vista de la topología de red, construye un spanning tree, y desconfigura el flujo en los puertos del switch que no están en la topología spanning tree.
- Web.webcore: Este componente inicia un servidor web con un proceso POX. Otros componentes pueden ser los que interactúen con este componente web.webcore para que provean contenido dinámico o estático.

- Messanger: Este componente proporciona una interfaz para que POX funcione con procesos externos, tratada como una API ya que la comunicación se la emplea por otros componentes llamados de transporte. Este componente está libre de uso para TCP y HTTP.
- Openflow.of_01: Este componente permite la comunicación entre versiones de switches Openflow 1.0.
- Openflow.discovery: Componente que envía mensajes LLDP (detección de nodos en una red) programados hacia los switches Openflow para poder revelar la topología actual de la red.
- Openflow.debug: Este componente hace que POX realice búsquedas en los mensajes Openflow.
- Openflow.keepalive: Componente que fuerza a POX a enviar solicitud 'echo' a los switch conectados para ver el tiempo de demora de respuesta por parte de los switch.
- Proto.dhcp_client: Es un componente de cliente DHCP. No es tan útil por sí mismo, pero en conjunto con otros componentes puede llegar a ser de utilidad.
- Proto.pong: El componente pong es ineficiente ya que solo mira las solicitudes ICMP(al hacer ping) y los responde.

Ahora, el usuario además de estos componentes puede ser capaz de desarrollar sus propios componentes para funciones que se presenten en cualquier escenario. El directorio donde se puede agregar los componentes creados pueden hallarse en dos partes: bajo el directorio /pox/pox/ext o en el directorio /pox/pox/forwarding. Dichos directorios son utilizados por Python para utilizar componentes y una más fácil manera de interactuar con ellos.

Requerimientos e instalación de POX

POX requiere de una versión de Python 2.7. POX corre bajo los Sistemas Operativos Windows, Mac OS y Linux. Muchos desarrollos se han hecho en Mac OS, ocasionalmente

se ha probado para los otros sistemas operativos de igual manera; la diferencia existente es el tiempo de como una función se lleva a cabo o de como los reportes de errores se realizan.

POX puede ser usado con el intérprete estándar de Python (CPython), pero también es soportado en PyPy.

La instalación del controlador POX se realiza a través de las siguientes líneas de comando:

- ***sudo apt-get install git***
- ***git clone https://github.com/noxrepo/pox***

Invocación de POX

Para invocar POX se lo hace corriendo o aplicando el fichero pox.py en una terminal. Pox también posee algunas opciones de inicialización que se muestran en la **TABLA N° 2** las mismas que se pueden utilizar en líneas de comando.

TABLA N° 2: OPCIONES PARA LA EJECUCIÓN DE POX

OPCIONES	DESCRIPCIÓN
--verbose	Muestra información extra (especialmente para problemas de inicio).
--no-cli	No inicializa con interfaz interactiva de comando.
--no-openflow	No empieza automáticamente escuchando conexiones Openflow.

Fuente: https://openflow.stanford.edu/display/ONL/POX+Wiki

POX por sí mismo realiza casi actividades nulas, la funcionalidad de POX se radica usando componentes (de la mano cuenta con algunos componentes, pero también es posible desarrollar algunos para uso exclusivo o de acuerdo al escenario con que se encuentre) descritas anteriormente, un ejemplo es el componente forwarding.l2_learning. Este componente hace que el switch se comporte como uno de capa dos [18].

2.5.2 CONTROLADOR RYU

RYU es un software soportado en plataforma Linux está basado en componentes para desarrollo de SDN. RYU provee de estos componentes de software con un API bien definido que ayuda a desarrollar nuevas aplicaciones de control y administración de redes. Soporta varios protocolos de administración de red entre ellas Openflow.

Ryu a diferencia de otros controladores permite la integración con otros componentes uno de ellos y con más prestaciones OpenStack que es un software que permite desarrollo en la nube a través de un plugin desarrollado en Ryu.

Algunas características del Controlador Ryu se las menciona a continuación:

- Tiene su propia interfaz Openflow que hace fácil la creación de nuevas aplicaciones de control.
- Esta soportado en sistema operativo Linux de código abierto y disponible bajo la Licencia Apache 2.0.
- Provee de componentes ya existentes y reusables, además de que se puede modificar e implementar nuevos componentes.
- Completamente basada en lenguaje Python, fácil y rápido de aprender.
- La creación de la red virtual es independiente del controlador y su ejecución.
- Utilización de librerías específicas para cada desarrollo de aplicaciones que requieran conexiones, llamadas, lectura de datos, etc.

Componentes de RYU

RYU provee de instrucciones y comandos para poder correr aplicaciones y comandos realizados en este controlador. La instrucción PYTHONPATH =. ./bin/ryu-manager permite correr o ejecutar cualquier acción que se genere a través de RYU. RYU también ofrece componentes ya ejecutables en sus archivos. Los componentes que se pueden citar son los siguientes:

- ryu.base.app_manager: Este permite la administración central de aplicaciones realizadas en RYU; carga aplicaciones en RYU, guía los mensajes entre aplicaciones y provee contexto a las aplicaciones en RYU.
- ryu.controller.controller: Es el principal componente del controlador OpenFlow; maneja las conexiones entre los switches.
- ryu.controller.dpset: Se encarga de administrar los switches y que es reemplazado por la topología ryu (ryu/topology).
- ryu.controller.ofp_event: Es el encargado de definir eventos OpenFlow.
- ryu.controller.ofp_handler: Maneja actividades básicas de OpenFlow incluyendo las negociaciones.
- ryu.app.simple_switch: Aplicación que pertenece a un switch Openflow versión 1.0 que su función es que dicho switch actúe como un switch de capa 2.
- ryu.app.gre_tunnel: Permite la integración de OpenStack.
- ryu.app.rest_conf_switch: Es un módulo que hace posible el uso de varias API REST para la manipulación de un switch.

Ahora el usuario además de disponer de estos componentes puede que desarrolle unos propios para cualquier escenario. El directorio donde los componentes nuevos pueden ser creados y almacenados puede hallarse bajo el directorio /ryu/app para su posterior uso y ejecución [2].

Requerimientos e instalación de RYU

Para la instalación de Ryu es necesario únicamente el sistema operativo Linux para que el controlador ryu se instale con éxito se requiere tener instalados los siguientes componentes: python-eventlet, python-routes, python-webob, python-paramiko. Esta instalación se realiza a traves de una sola línea de código que es la siguiente:

Time sudo apt-get install python-eventlet python-routes python-webob python-paramiko

Una vez que los requisitos para el uso de Ryu están instalados, el controlador se instala con las siguientes líneas de comando:

- ***pip install ryu***
- ***git clone git://github.com/osrg/ryu.git***
- ***cd ryu; python ./setup.py install***

Invocación de RYU

Para hacer uso de este controlador es necesario hacer el archivo ryu-manager, este permite la ejecución del controlador, la línea que hace posible la ejecución permite usar la opción **--verbose** esta hace posible obtener más información de la ejecución del controlador.

2.5.3 CONTROLADOR PYRETIC

Pyretic basado en plataforma Linux permite a los desarrolladores de aplicaciones enfocarse en cómo se puede especificar una política de red a un alto nivel de abstracción, en lugar de implementarlos usando mecanismos OpenFlow de bajo nivel.

Con Pyretic no se especifica reglas específicas de switches, en lugar de eso permite realizar una política general para toda la red una sola vez desde paquetes de ingreso a un conjunto de paquetes de salida [10].

Pyretc ofrece políticas predefinidas, además de que permite la creación de varias de estas manualmente. La política más usada es flood ( ), que hace que retorne un paquete por cada puerto local en una red de topología tree.

Algunas características del controlador Pyretic se las menciona a continuación:

- Tiene su propia interfaz Openflow que hace fácil la creación de nuevas aplicaciones y creación de módulos.
- Esta soportado en sistema operativo Linux de código abierto.
- Permite a los programadores de SDN realizar aplicaciones extensas, complejas y sofisticadas desde pequeños y simples módulos reusables que contiene Pyretic.
- Está basado en el lenguaje de programación Python.
- La creación de la red virtual es independiente del controlador y su ejecución.
- Permite realizar políticas o reglas de la red a través de funciones, estos toman paquetes de entrada y regresa un conjunto de paquetes como salida.
- Habilita a cada política para operar en una topología abstracta que implícitamente contiene que módulo se usará y que hará.

Componentes de PYRETIC

El módulo principal que contiene Pyretic es el llamado pyretic.py que es el módulo que permite ejecutar todas las aplicaciones que se han creado bajo este controlador y con el lenguaje Python. PYRETIC también ofrece componentes ya ejecutables en sus archivos. Los componentes que se pueden citar son los siguientes:

- mac_learner.py: Permite de cada host aprender y guardar direcciones MAC existentes.

- hub.py: Regresa un conjunto de paquetes de cada puerto existentes en un spanning tree.
- arp.py: Crea, envía y recibe mensajes ARP de todas las conexiones de los host existesntes en la red.
- of_tutorial-py: Implementa un comportamiento de un hub y de un switch, envía todos los paquetes a todos los puertos excepto por el puerto de entrada.
- topology_printer.py: Añade características de puertos a la topología y permite el uso de una topología a través de los IDS de los switches.
- rewrite.py: Emplea el uso de direcciones MAC e IPS para el envió de datos.
- backend.py: Recive conexiones y se encarga de manejar aplicaciones en segundo plano.

Requerimientos e instalación de PYRETIC

Para la instalación del controlador PYRETIC el único requerimiento es el sistema operativo Linux, este controlador viene dentro de una máquina virtual donde ya está instalado, y no provee de una fuente de descarga para la instalación. El directorio donde los componentes nuevos pueden ser creados y almacenados puede hallarse bajo el directorio /pyretic/modules para su posterior uso y ejecución.

Invocación de PYRETIC

Para invocar al controlador pyretic es necesario usar una línea de comando con **pyretic.py,** la línea de comando que se ejecuta es la siguiente:

pyretic.py –m p0 pyrectic.modules.mac_learner

Donde pyretic.py es el comando principal para la ejecución de pyretic, –m p0 indica el modo de operación de cómo se ejecuta pyretic y pyretic.modules.mac_learner es el modulo que se está ejecutando.

Una vez conocidas las características de cada controlador es necesario establecer parámetros comparativos para estudiar a cada uno de manera equitativa para ello se establece en la **TABLA N° 3** una comparación de los tres controladores: RYU, POX y PYRETIC.

TABLA N° 3: COMPARACIÓN TEÓRICA DE LOS CONTROLADORES POX, RYU Y PYRETIC

	Lenguaje	Plataforma	Rendimiento	Latencia	Inferfaz Openflow	Componentes reusables	Directorio – controladores	Tipo de ejecución
POX	Python	Linux, Windows y MAC	iperf	Mensaje ICMP	Si	Si	/pox/pox/ext	Englobado en un archivo
RYU	Python	Linux	iperf	Mensaje ICMP	Si	Si	/ryu/app	Independiente
PYRETIC	Python	Linux	iperf	Mensaje ICMP	Si	Si	/pyretic/modules	Independiente

Fuente: Fabián Gallegos, Catherine Yánez

Con la teoría no se puede corroborar que un controlador sea mejor que otro debido a que los tres controladores: POX, RYU y PIRETIC son de código libre y están propensos a constantes mejoras por lo que solo a través de pruebas se puede definir que controlador tiene un grado de ventaja sobre los demás.

CAPÍTULO III

3. COMPARACIÓN DE CONTROLADORES EN UN ESCENARIO DE PRUEBA

La investigación propuesta sugiere conocer y establecer una comparación de algunos de los controladores para determinar el más eficiente y con mejores características de entre todos y proponer la solución al problema. Se establece un escenario de prueba para los controladores POX, RYU, PYRETIC, se utiliza módulos predefinidos para él controlador y se definen parámetros de comparación que constan de rendimiento, latencia y líneas de código.

Para el análisis de resultados se define una valoración o medición de los datos obtenidos de forma cualitativa y cuantitativa la misma que consta en **TABLA N° 4**

TABLA N° 4: VALORACIÓN CUALITATIVA Y CUANTITATIVA DE DATOS

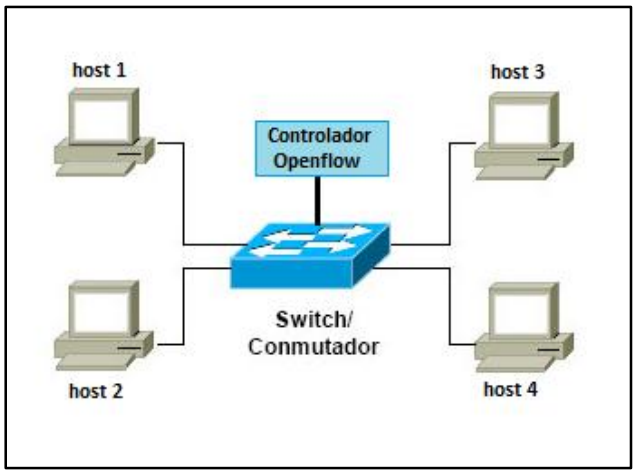
	MALO	REGULAR	BUENO	MUY BUENO	EXCELENTE
VALOR CUANTITATIVO	1	2	3	4	5
PORCENTAJES	20%	40%	60%	80%	100%

Fuente: Fabián Gallegos, Catherine Yánez

3.1 DESCRIPCIÓN DEL ESCENARIO

Para la parte comparativa se ha predeterminado un escenario de prueba el mismo que será aplicado con cada uno de los controladores: pox, ryu y pyretic, a través del uso de mininet usando Virtual Box, se creará la red virtual en la cual se harán las pruebas necesarias **FIGURA N° 4.**

FIGURA N° 4: ESCENARIO DE PRUEBA



Fuente: Fabián Gallegos, Catherine Yánez

Elementos del escenario: Se establece varios parámetros y características que contiene el escenario para la comparación, se ha escogido una topología simple debido a que únicamente se desea conocer el comportamiento y la funcionalidad del escenario con los controladores, las diferencias encontradas en los controladores dentro de esta topología también pueden ser las mismas en una topología más compleja pero a mayor escala.

- ✓ Topología simple
- ✓ Dispositivos virtuales:
  - 4 host
  - 1 switch: Openflow 1.3

- 1 controlador

✓ Características de conexión.

Los valores asignados de conexión para los hosts propuestos son acoplados a un escenario virtual por lo que éstos son propuestos al azar y así tratar de simular un escenario real. El porcentaje de perdida de paquetes en el escenario se establece con un rango de 1 a 5 por ciento ya que es una parte fundamental para la obtención de resultados.

- Ancho de banda: 10 (bw)
- Demora: 10 milisegundos (delay)
- Porcentaje de perdida de paquetes (loss)
- Tamaño de paquetes en cola: 1000 (max_queue_size)

3.2 EJECUCIÓN DE CONTROLADORES

Deliminación de la programación para los controladores: Los tres controladores están programados usando el lenguaje de programación Python, la programación de cada uno difiere debido a los paquetes de librerías que usa cada uno.

Para la comparación únicamente se toma en cuenta la parte estructural de la red: latencia, líneas de código y rendimiento de la red debido a que los controladores influyen en el comportamiento de la red virtual y por ende estos parámetros varian de controlador en controlador.

Para obtener los resultados se usó el software **iperf** del que dispone mininet, el mismo que proporciona información directa del rendimiento de una red y a través del incremento de paquetes ICMP (ping) se obtiene la latencia de la red, las líneas de código se revisaran de forma manual en el caso de cada uno de los controladores.

El controlador POX, RYU y PYRETIC, dentro de la red virtual realiza lo siguiente:

✓ Añade tablas de flujo.

✓ Envía paquetes a todos los host.

- ✓ Aprende direcciones MAC para evitar inundación (varios caminos a un mismo destino) de paquetes en un mismo host.

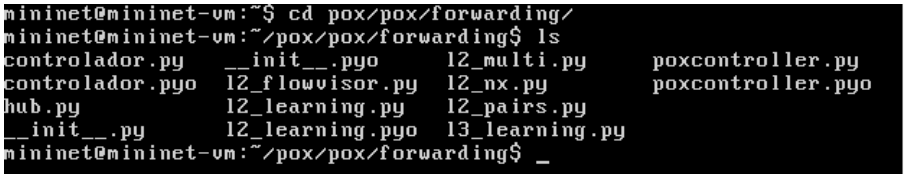
3.3 APLICACIÓN Y DESCRIPCIÓN DEL USO DE LOS CONTROLADORES Y LA RED VIRTUAL

La aplicación y ejecución de cada controlador varía, a continuación se muestra la ejecución de los tres controladores: POX, RYU y PYRETIC dentro de una mininet:

3.3.1 APLICACIÓN Y EJECUCIÓN DE POX

El Controlador POX para su correcto funcionamiento debe correr bajo un directorio específico, en este caso la dirección /pox/pox/forwarding es la óptima **FIGURA N° 5.** Dentro de esta carpeta se guarda el controlador con todos sus requerimientos y necesidades.

FIGURA N°5: DIRECTORIO FORWARDING DE POX



Fuente: Fabián Gallegos, Catherine Yánez

Una vez creado el controlador, es necesario crear el archivo que permite referenciar dicho controlador con la creación de la Mininet, el archivo que se usa para determinar estos dos puntos (controlador-mininet) en este caso se la ha llamado "pruebapox.py" siendo la extensión ".py" característica de archivos bajo la programación Python. La **FIGURA N° 6** explica cómo se ejecuta el archivo a través de la sentencia "sudo ./pruebapox.py" donde

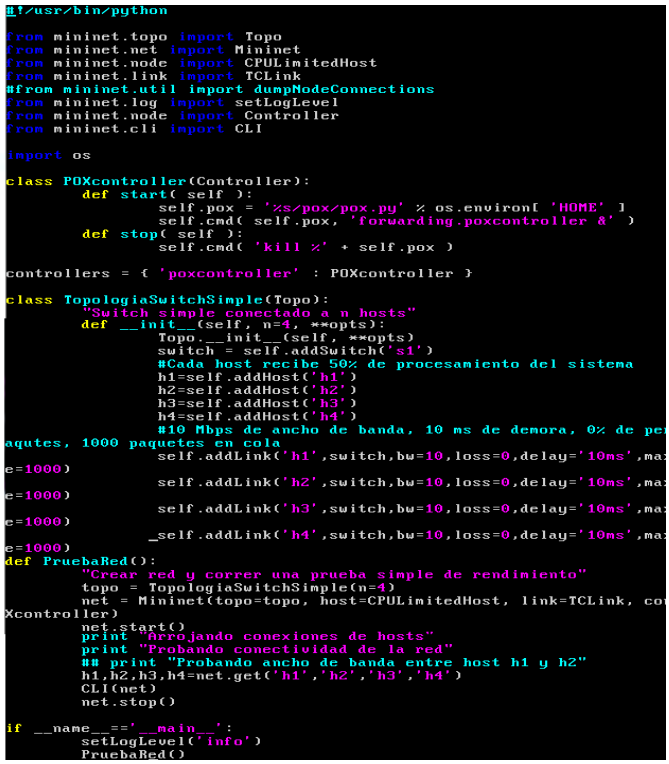
"sudo" es una sentencia Linux que permite a los usuarios ejecutar programas con privilegios de seguridad de otro usuario (root), la instrucción "./" es propia de POX y sirve para llamar a los archivos creados en Python.

FIGURA N° 6: LLAMADA AL ARCHIVO EN POX

Fuente: Fabián Gallegos, Catherine Yánez

En el archivo "pruebapox.py" ejecutado anteriormente, existen llamadas a librerías útiles para la creación de la red virtual mininet, una función para ejecutar la red, y además dos clases esenciales para que todo el escenario de la **FIGURA N° 4** funcione correctamente. En la **FIGURA N° 7** se presenta el contenido de éste archivo.

FIGURA N° 7: CONTENIDO ARCHIVO PRUEBAPOX.PY



```
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import CPULimitedHost
from mininet.link import TCLink
#from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel
from mininet.node import Controller
from mininet.cli import CLI

import os

class POXcontroller(Controller):
        def start( self ):
                self.pox = '%s/pox/pox.py' % os.environ[ 'HOME' ]
                self.cmd( self.pox, 'forwarding.poxcontroller &' )
        def stop( self ):
                self.cmd( 'kill %' + self.pox )

controllers = { 'poxcontroller' : POXcontroller }

class TopologiaSwitchSimple(Topo):
        "Switch simple conectado a n hosts"
        def __init__(self, n=4, **opts):
                Topo.__init__(self, **opts)
                switch = self.addSwitch('s1')
                #Cada host recibe 50% de procesamiento del sistema
                h1=self.addHost('h1')
                h2=self.addHost('h2')
                h3=self.addHost('h3')
                h4=self.addHost('h4')
                #10 Mbps de ancho de banda, 10 ms de demora, 0% de pe
aqutes, 1000 paquetes en cola
                self.addLink('h1',switch,bw=10,loss=0,delay='10ms',ma
e=1000)
                self.addLink('h2',switch,bw=10,loss=0,delay='10ms',ma
e=1000)
                self.addLink('h3',switch,bw=10,loss=0,delay='10ms',ma
e=1000)
                self.addLink('h4',switch,bw=10,loss=0,delay='10ms',ma
e=1000)
def PruebaRed():
        "Crear red y correr una prueba simple de rendimiento"
        topo = TopologiaSwitchSimple(n=4)
        net = Mininet(topo=topo, host=CPULimitedHost, link=TCLink, co
Xcontroller)
        net.start()
        print "Arrojando conexiones de hosts"
        print "Probando conectividad de la red"
        ## print "Probando ancho de banda entre host h1 y h2"
        h1,h2,h3,h4=net.get('h1','h2','h3','h4')
        CLI(net)
        net.stop()

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

Fuente: Fabián Gallegos, Catherine Yánez

Una de las clases que se encuentran en la **FIGURA N° 7** se le ha asignado con el nombre de "POXcontroller" es utilizada tanto para llamar al controlador desde su directorio donde se aloja, como para la ejecución del mismo.

Las librerías que se usan en este archivo son las siguientes:

- **Mininet.topo** import **Topo**: Clase base para las topologías de mininet.
- **Mininet.net** import **Mininet**: Permite el uso básico de comandos de la mininet.
- **Mininet.node** import **CPULimitedHost**: Limita y aísla características de la red. Librería que permite usar los hosts dentro de la mininet.
- **Mininet.link** import **TCLink**: Limita y aísla características de la red. Librería que permite usar los enlaces de la red.
- **Mininet.log** import **setLogLevel**: Ajusta en nivel de salida de la mininet (info: Es el que provee información mas útil)
- **Mininet.node** import **Controller:** Permite el uso del controlador dentro de la mininet.
- **Mininet.cli** import **CLI:** Librería que pertence a una interfaz de línea de comandos, que permite ejecutar comando en la mininet.

La clase **Topología switchSimple** permite la creación de los hosts, switch y enlaces de red con características específicas:

- Def__init__: Sirve para definir un método en python.
- Topo.__init__: Invocacion de la clase Topo para crear una topología.
- addSwitch: Añade un switch (s1) a la red.
- addHost: Añade un host (h1, h2, h3, h4) a la red.
- addLink: Añade un enlace bidireccional entre el switch y un host.
- Bw: Ancho de banda usado.
- Loss: Perdida de conexión.
- Delay: Demora para cada enlace entre el switch y un host
- Max_queue_size: Tamaño de los paquetes en cola.

El método **def_PruebaRed** permite la creación de la red:

- topo: Llama a la clase creada: **Topología switchSimple.**
- net: Crea la red virtual mininet con las características definidas en la clase anterior.
- start(), stop(): Empieza y detiene la red creada.
- CLI(net): Interfaz de comando para la red.

if__name__ es el metodo principal donde se invocan funciones y métodos en este caso: **def_PruebaRed.**

Cabe recalcar que para la llamada del controlador se debe colocar tanto el ejecutable de POX: "/pox.py" como la dirección física del archivo del controlador en este caso "forwarding.poxcontroller" para ubicar el archivo de controlador que se ha creado. En la **FIGURA N° 8** se amplía más la descripción de este paso para el Controlador POX.

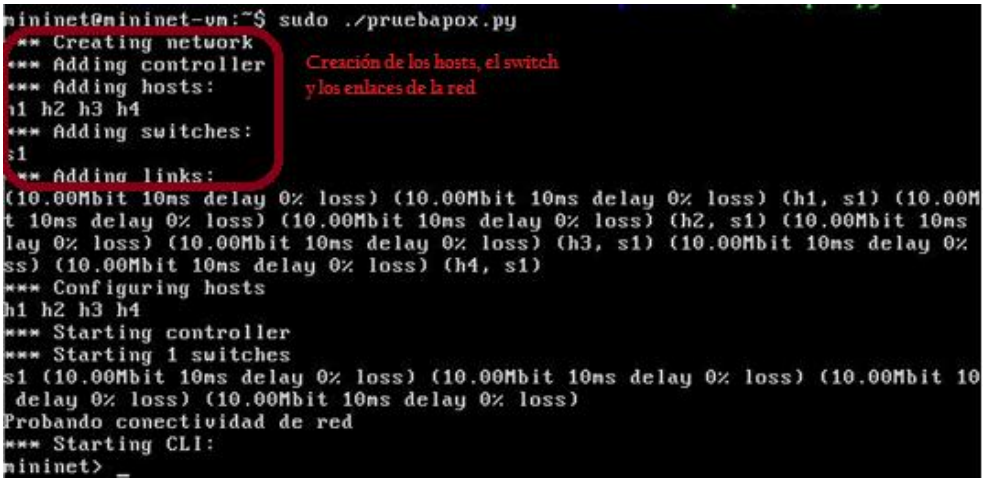
FIGURA N° 8: CLASE POXCONTROLLER

```
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel
from mininet.node import Controller
from mininet.cli import CLI

import os

class POXcontroller(Controller):
        def start( self ):
                self.pox = '%s/pox/pox.py' % os.environ[ 'HOME' ]
                self.cmd( self.pox, 'forwarding.poxcontroller &' )
        def stop( self ):
                self.cmd( 'kill %' + self.pox )

controllers = { 'poxcontroller' : POXcontroller }
```

Fuente: Fabián Gallegos, Catherine Yánez

La función que ejecuta la red que se acaba de crear se la ha demoninado "PruebaRed" dado en la **FIGURA N° 7** se halla la ejecución de la red, los números de host que contiene, utilización de un controlador externo y la interfaz con la que se manejará la red posteriormente **FIGURA N° 9.**

FIGURA N° 9: CREACIÓN DE LA RED CON EL CONTROLADOR POX



Fuente: Fabián Gallegos, Catherine Yánez

Para comprobar el uso de un controlador una vez ejecutado el archivo es necesario hacer un pingall **FIGURA N° 10** el mismo que comprobará que el controlador POX se está ejecutando

FIGURA N° 10: PINGALL CON POX

Fuente: Fabián Gallegos, Catherine Yánez

El archivo con las especificaciones anteriores debe ser guardado en el directorio raíz "/home/mininet/", para que todas las librerías que se alojan en el directorio Mininet puedan ser disponibles y además que el ejecutador de POX "pox.py" permita la ejecución del controlador. En la **FIGURA N° 11** se encuentra el alojamiento del archivo creado "pruebapox.py".

FIGURA N° 11: DIRECTORIO RAÍZ PARA POX

Fuente: Fabián Gallegos, Catherine Yánez

3.3.2 APLICACIÓN Y EJECUCIÓN DE RYU

En el caso del controlador ryu se hace uso de **dos** de las **n** terminales de las que dispone la Mininet. Para acceder a un nuevo terminal se emite la siguiente combinación de teclas, para la terminal 1: (Alt + F1), para la terminal 2: (Alt+F2), para la terminal n: (Alt+Fn).

Para el uso de RYU es necesario crear un controlador bajo la dirección **/home/ryu/ryu/app** dentro de esta carpeta se crean todos los controladores de ryu solo los controladores establecidos dentro de esta carpeta se puede usar sin ningún problema**,** en este caso **FIGURA N° 12** se le asignó el nombre de **controladorryu.py**.

FIGURA N° 12: DIRECTORIO APP DE RYU

Fuente: Fabián Gallegos, Catherine Yánez

A continuación en el terminal 1: (Alt+F1) se procede a crear la mininet **FIGURA N° 13** con las características del escenario de la **FIGURA N° 4.**

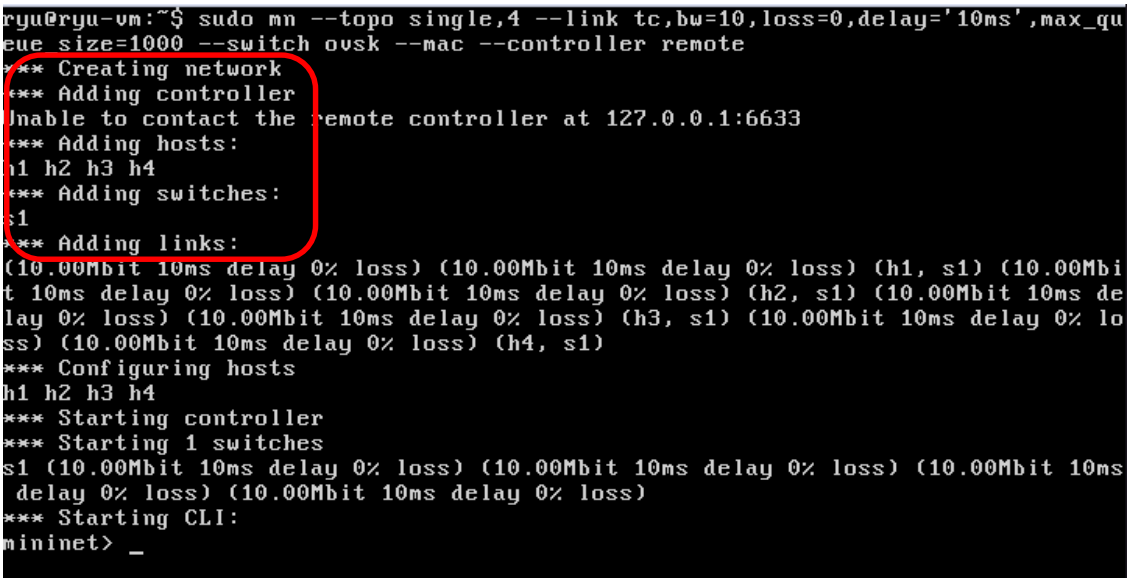
FIGURA N° 13: COMANDO PARA CREACIÓN DE LA MININET EN RYU

Fuente: Fabián Gallegos, Catherine Yánez

En la **FIGURA N° 13** se muestra la línea de comando para la creación de la red con una topología simple con cuatro host, un ancho de banda (bw) en la red de 10, demora (delay) de 10ms, tamaño de los paquetes en cola (max_queue_size) de 1000.

El resultado de la ejecución de esta línea de comando permite conocer paso a paso como se crea la red: se crea la red, se añade el controlador, se añaden los cuatro hosts, se añade un switch y los enlaces entre cada host y el swicth con las características específicas en la **FIGURA N° 14** se muestra la creación de la red.

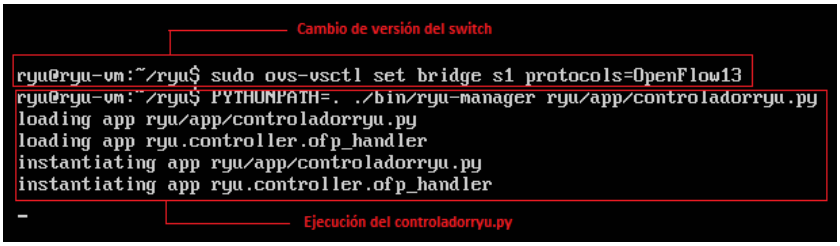
FIGURA N° 14: CREACIÓN DE LA RED CON EL CONTROLADOR RYU



Fuente: Fabián Gallegos, Catherine Yánez

En el terminal dos: (Alt+F2) se ejecuta el controlador: **controladorryu.py**, pero antes es necesario tener en cuenta la versión del switch de la mininet, generalmente el switch no es compatible con el protocolo openflow razón por la cual es necesario cambiar la versión al switch a una versión 3. **FIGURA N° 15**

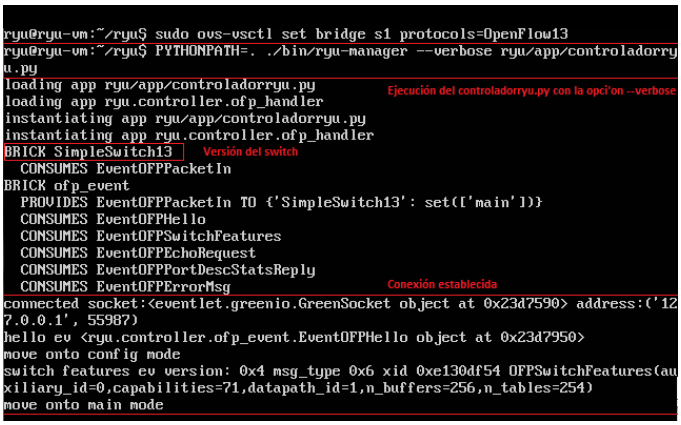
FIGURA N° 15: EJECUCIÓN DEL CONTROLADOR RYU - CONTROLADORRYU.PY



Fuente: Fabián Gallegos, Catherine Yánez

En la **FIGURA N° 15,** para la ejecución del controlador la línea de código que se usa llama al ejecutable principal de ryu **/bin/ryu-manager**, este archivo es el que permite la ejecución del controlador en ryu. Si a esta línea de código se le añade la opción –verbose se pueden confirmar que la conexión con el switch se estableció correctamente **FIGURA N° 16.**

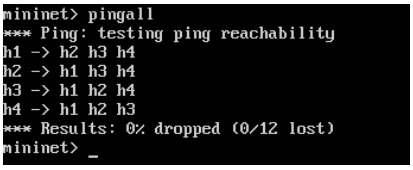
FIGURA N° 16: USO DE LA OPCIÓN –VERBOSE EN RYU



Fuente: Fabián Gallegos, Catherine Yánez

Para realizar la comparación en términos equitativos es necesario cumplir con las características del controlador preestablecido inicialmente por lo que se requiere hacer un **pingall** en el terminal 1 donde está creada la mininet **FIGURA N° 17.**

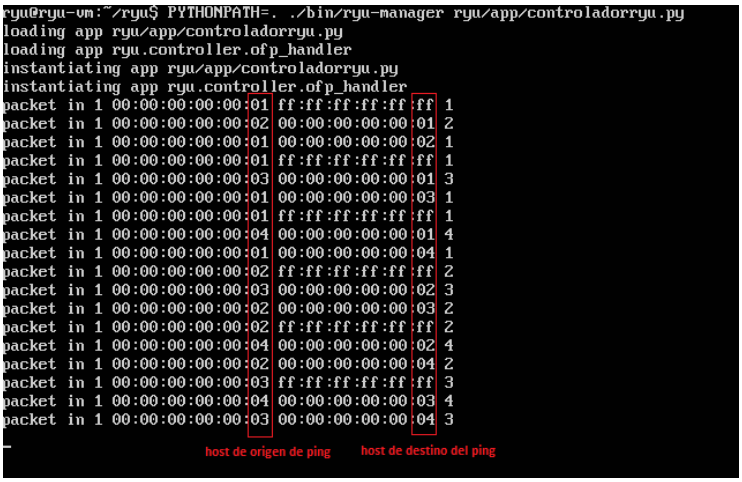
FIGURA N° 17: PINGALL CON RYU



Fuente: Fabián Gallegos, Catherine Yánez

El pingall se ve en el controlador ryu **FIGURA N° 18** que se ejecutó en el terminal 2**.**

FIGURA N° 18: PINGALL DE LA MININET REFLEJADO EN EL CONTROLADOR RYU



Fuente: Fabián Gallegos, Catherine Yánez

3.3.3 APLICACIÓN Y EJECUCIÓN DE PYRETIC

Al igual que el anterior controlador (RYU), el desarrollo normal se lo debe realizar con el uso de dos terminales que ofrece Linux. Para acceder a un nuevo terminal se emite la siguiente combinación de teclas, para la terminal 1: (Alt + F1), para la terminal 2: (Alt+F2), para n terminales: (Alt+Fn).

Para el caso del controlador PYRETIC, el directorio que aloja todos los controladores está bajo: pyretic/pyretic/modules/, es donde PYRETIC hace uso de todos los controladores creados. En la **FIGURA N° 19** se visualiza este directorio y el controlador que se ha creado, en este caso con el nombre: controladorpyretic.py.

FIGURA N° 19: DIRECTORIO MODULES EN PYRETIC

Fuente: Fabián Gallegos, Catherine Yánez

A continuación en el terminal 1: (Alt+F1) se procede a crear la Mininet **FIGURA N° 20** con las características del escenario.

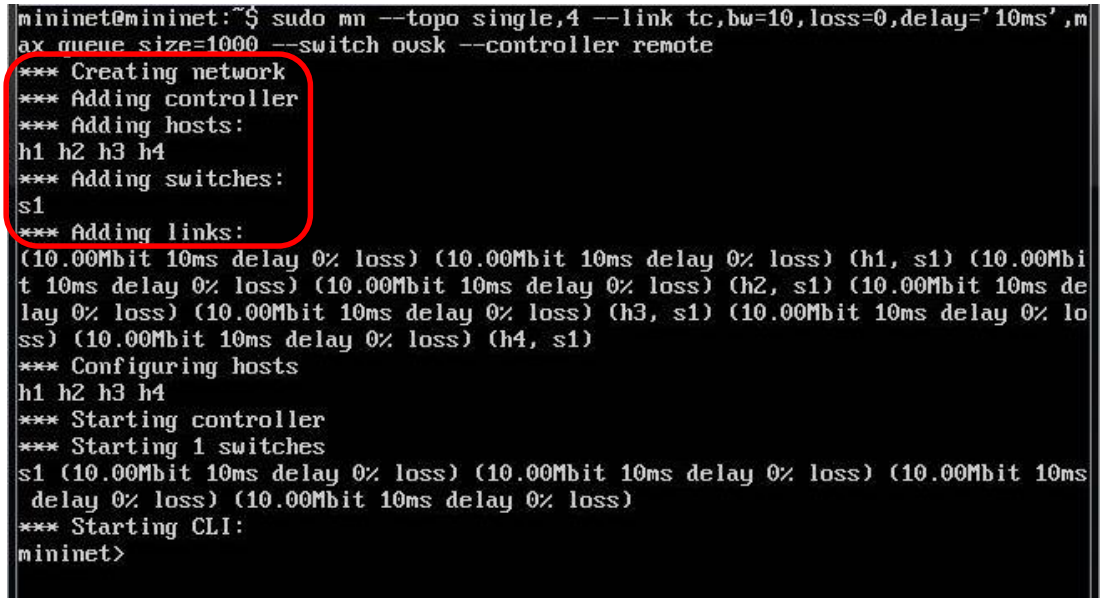
FIGURA N° 20: COMANDO PARA LA CREACIÓN DE LA RED EN PYRETIC

Fuente: Fabián Gallegos, Catherine Yánez

En la **FIGURA N° 20** se muestra la creación de red con una topología simple con 4 host, un ancho de banda (bw) en la red de 10, demora (delay) de 10ms, tamaño de los paquetes en cola (max_queue_size) de 1000.

La ejecución de la línea de comando anterior se muestra en la **FIGURA N° 21** se crea la red, se añade el controlador, se añaden los cuatro host, el switch y los enlaces entre el switch y cada uno de los hosts.

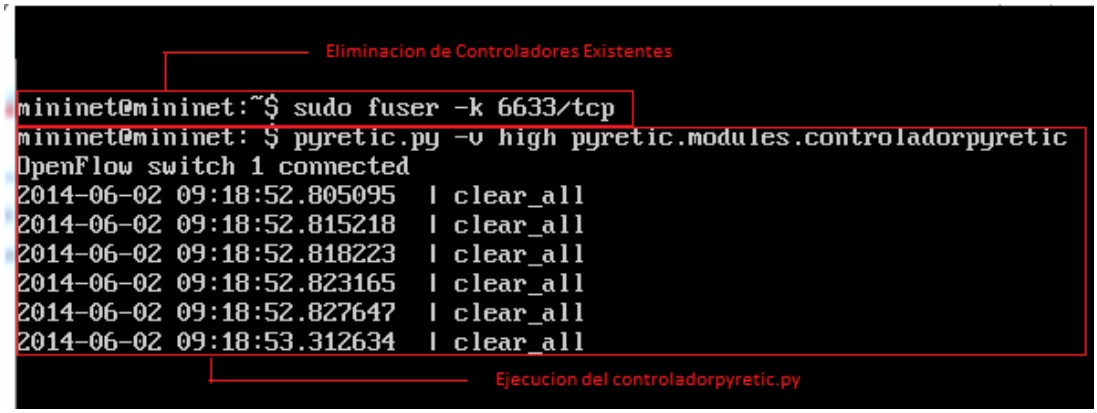
FIGURA N° 21: CREACIÓN DE LA RED CON EL CONTROLADOR PYRETIC



Fuente: Fabián Gallegos, Catherine Yánez

En la terminal dos: (ALT + F2) se ejecuta el controlador PYRETIC que se ha desarrollado, pero antes de esta parte se debe eliminar todos los controladores externos que se pudieron haber creado anteriormente al nuevo que se va a ejecutar para ello la instrucción sudo fuser –k 6633/tcp detallada en la **FIGURA N° 22**, se encarga de que en el puerto 6633 se eliminen cualquier enlace existente con controladores ya creados.

FIGURA N° 22: EJECUCIÓN DEL CONTROLADOR PYRETIC – CONTROLADORPYRETIC.PY

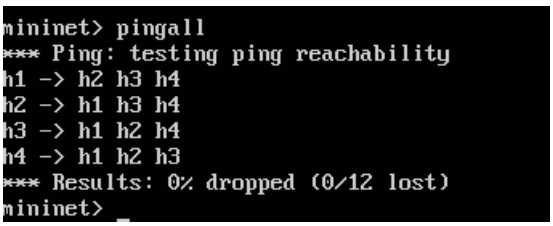


Fuente: Fabián Gallegos, Catherine Yánez

En la **FIGURA N° 22** se aprecia como el controlador basado en PYRETIC se ejecuta correctamente tecleando la siguiente instrucción: pyretic.py –v high pyretic.modules.controladorpyretic. Si se desea correr algún otro controlador la línea en la instrucción anterior y que antecede al nombre del controlador (controladorpyretic) debe ser la misma.

Con la ejecución de la red virtual y el controlador se realizará una prueba entre todos los host de la red a través de un pingall en la terminal 1 que se detalla en la **FIGURA N° 23.**

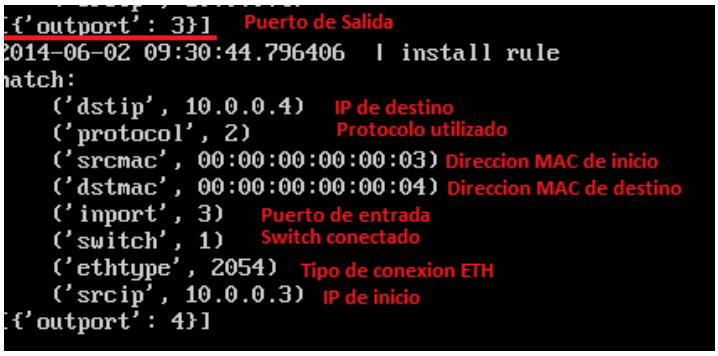
FIGURA N° 23: PINGALL CON RYRETIC



Fuente: Fabián Gallegos, Catherine Yánez

La acción realizada en la red virtual se ve reflejada en el controlador **FIGURA N° 24,** que se ejecutó en la terminal dos.

FIGURA N° 24: PINGALL DE LA MININET REFLEJADO EN EL CONTROLADOR PYRETIC



Fuente: Fabián Gallegos, Catherine Yánez

3.4 COMPARACIÓN DE LOS CONTROLADORES

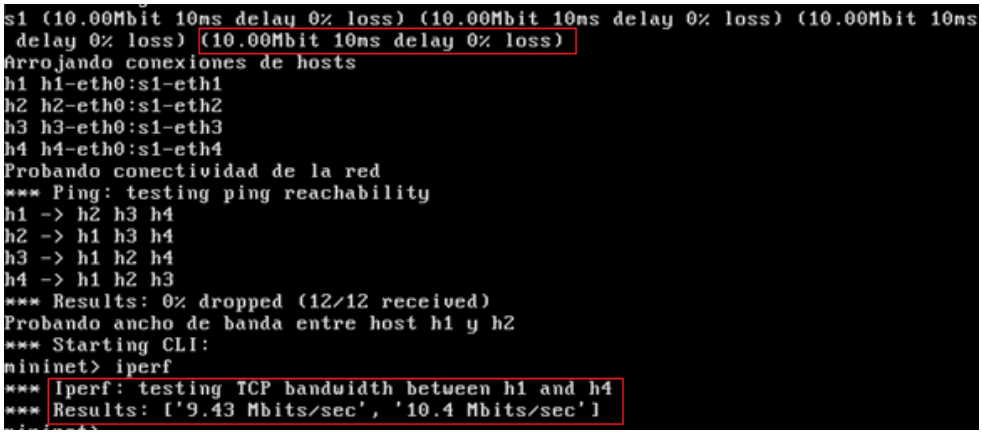
Indicadores de evaluación y análisis, para la comparación de los controladores: Como se mencionó anteriormente los indicadores a considerar para realizar la parte comparativa son:

a) **Medición de Rendimiento:** Para medir el rendimiento se usará el software iperf. Para obtener resultados relevantes máximos y mínimos, se realiza las pruebas con diferentes valores de porcentaje de probabilidad de perdida de conexión entre switch-host, la perdida en la creación de la Mininet está representado con la palabra "loss" por lo que las pruebas se harán sin perdida y con pérdida del 1 al 5 % (loss=1, loss=2,… loss=5), estas variaciones de perdida se hacen al crear la topología de red:

sudo mn –topo single,4 –link tc,bw=10 loss=0,delay='10ms' ,max_queue_size=1000 –switch ovsk –mac –controller remote

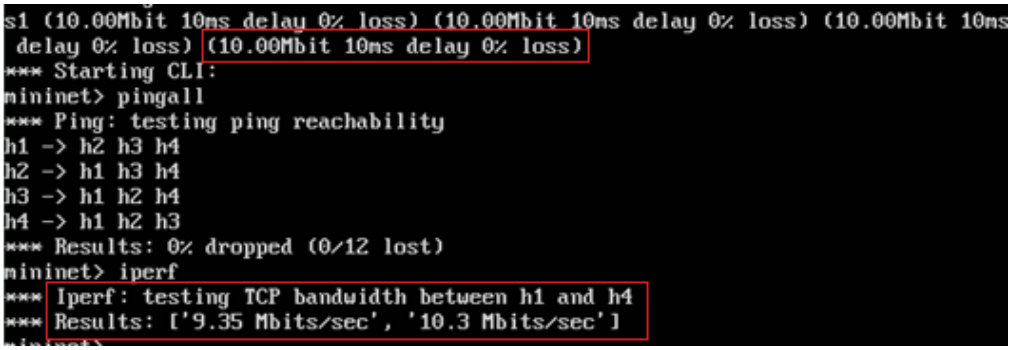
En la **FIGURA N° 25, 26** y **27** se muestran los resultados de rendimiento obtenidos con cada uno de los controladores considerando una probabilidad de pérdida del 0% los demás resultados se adjuntan en el ver **ANEXO N° 1**

FIGURA N° 25: RENDIMIENTO CON 0% PERDIDA EN POX



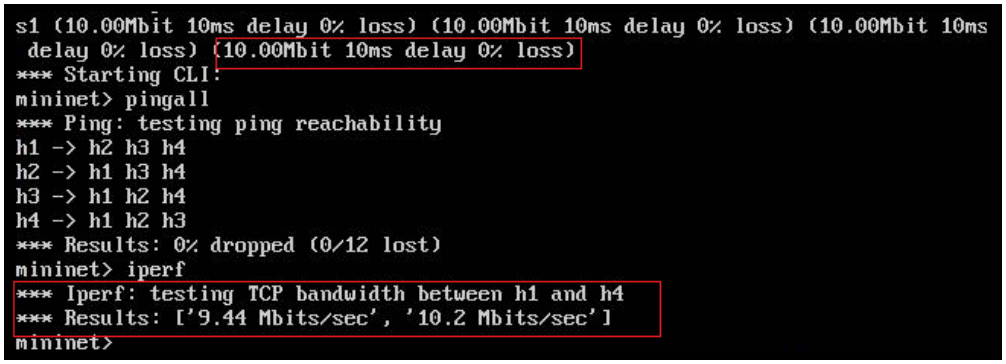
Fuente: Fabián Gallegos, Catherine Yánez

FIGURA N° 26: RENDIMIENTO CON 0% PERDIDA EN RYU



Fuente: Fabián Gallegos, Catherine Yánez

FIGURA N° 27: RENDIMIENTO CON 0% PERDIDA EN PYRETIC



Fuente: Fabián Gallegos, Catherine Yánez

Cabe recalcar que el porcentaje de perdida influye en el decremento o incremento del rendimiento de red, para cada caso el valor del rendimiento se mide en Mbits/sec o Kbits/sec, obteniendo la **TABLA N° 5** de valores**.**

TABLA N° 5: RENDIMIENTO DE LOS CONTROLADORES

	Sin perdida Mbits/sec		**Loss=1 Mbits/sec**		**Loss=2 Mbits/sec**		**Loss=3 Mbits/sec**		**Loss=4 Kbits/sec**		**Loss=5 Kbits/sec**	
	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max
POX	9.43	10.4	2.21	2.23	1.62	1.64	1.30	1.32	785	840	626	635
RYU	9.35	10.3	1.96	1.99	1.34	1.35	1.13	1.15	782	792	436	434
PYRETIC	9.44	10.2	2.03	2.07	1.37	1.40	1.11	1.22	755	782	555	558

Fuente: Fabián Gallegos, Catherine Yánez

En la **TABLA N° 5** se sitúan los valores obtenidos con el iperf para cada controlador, cada celda contiene un valor definido como la velocidad con la que se transmiten los datos en tiempo de un segundo. Se ha usado una probabilidad de pérdida del 0 al 5% esto quiere decir que del 100% de conexiones entre switch y hosts existe un porcentaje de error de conexión que hace que el rendimiento de la red se vea afectado y por esta razón se ve la diferencia entre uno y otro controlador.

Para el uso del software iperf se realizó previamente la ejecución de la instrucción "pingall", es decir, considerando que el escenario de la red virtual posee 4 host, cada uno realiza un ping simple a los tres restantes, obteniendo 12 ping's en total; iperf actúa sobre estos 12 ping's y selecciona de estos el valor máximo y mínimo de ancho de banda.

Análisis del Indicador de rendimiento

En la **TABLA N° 5** se puede obtener valores de rendimiento de los controladores POX, RYU y PYRETIC, con una probabilidad de pérdida de conexiones. Valores que hacen referencia al ancho de banda de la conexión, estos valores equivalen al máximo y mínimo de rendimiento.

Para este análisis es necesario resumir los resultados finales que se explican en la **TABLA N° 6**. En esta tabla se refleja un resultado de la división entre la suma de los valores máximos y el número de pruebas realizadas de la **TABLA N° 5**, este proceso será aplicado también para los valores mínimos y que se detalla en la siguiente fórmula:

$$MAX\ o\ MIN = \frac{Sin\ perdida + Loss = 1 + Loss = 2 + Loss = 3 + Loss = 4 + Loss = 5}{6}$$

TABLA N° 6: TABLA RESULTADO DE RENDIMIENTO DE LOS CONTROLADORES

Controladores	Rendimiento (Mbits/sec)		Promedio
	Min	Max	
POX	2.67	2.84	2.755
RYU	2.50	2.67	2.585
PYRETIC	2.54	2.71	2.625

Fuente: Fabián Gallegos, Catherine Yánez

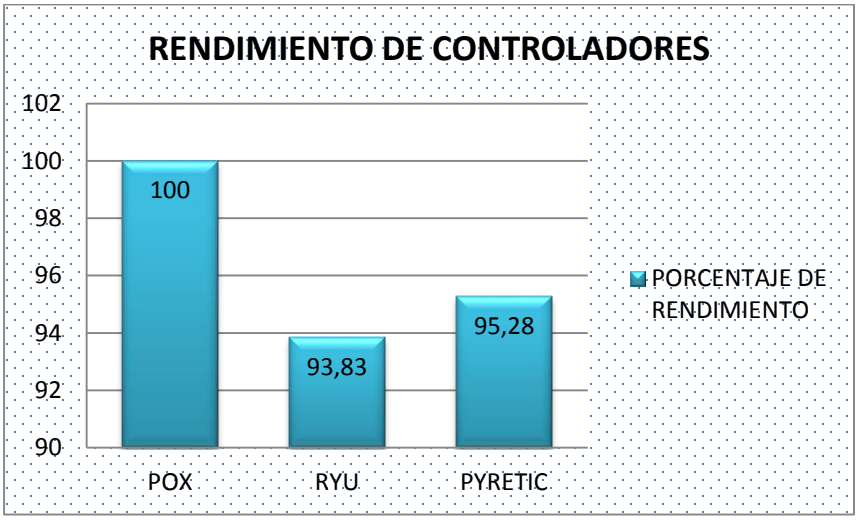
Para la toma de decisiones se realiza un promedio entre el rendimiento máximo y rendimiento mínimo el mismo que consta en la **TABLA N° 6**, este sirve para asignar un porcentaje del 100% para el valor mayor. Los demás porcentajes se obtienen utilizando una "regla de tres simple".

TABLA N° 7: PORCENTAJES DEL PROMEDIO DE RENDIMIENTO DE LOS CONTROLADORES

CONTROLADORES	PORCENTAJE RENDIMIENTO
POX	100%
RYU	93.83%
PYRETIC	95.28%

Fuente: Fabián Gallegos, Catherine Yánez

FIGURA N° 28: PORCENTAJE DEL PROMEDIO DE RENDIMIENTO DE LOS CONTROLADORES



Fuente: Fabián Gallegos, Catherine Yánez

De acuerdo a los resultados obtenidos en porcentajes en la **TABLA N° 7** el controlador POX es *excelente* (100%) tanto en rendimiento porque en la red mientras exista más ancho de banda se puede obtener mayor velocidad de transmisión de datos, menos perdida de datos y menor impacto de latencia haciendo que la red sea la más óptima, el controlador Ryu (93.83%) y Pyretic (95.28%) son *muy buenos*, pero tienen un menor ancho de banda en la red que el controlador POX.

b) **Medición de latencia:** La latencia varía dependiendo del número de paquetes que se transmiten y se mide a través de un mensaje ICMP (ping) el mismo que nos permite obtener varios valores de tiempos los cuales se definen a continuación:

Rtt: Hace referencia al tiempo que se demora un paquete en ser enviado desde el emisor hasta volver al mismo emisor después de haber pasado por el receptor.

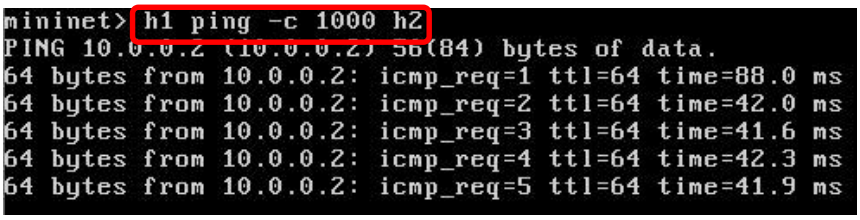
- **Min:** Tiempo mínimo

- **Max:** Tiempo máximo
- **Avg**: Promedio de los tiempos
- **Mdev**: Es la desviación estándar de los tiempos, valor numérico que representa una fluctuación de más o menos tiempo del promedio de tiempos. La fórmula para calcular este campo es el siguiente:

$$\sqrt[2]{\frac{(Max - Avg)^2 (Min - Avg)^2}{Numero\ de\ host}}$$

Para obtener los valores de latencia cabe recalcar que el ping se realiza manualmente desde Mininet, como se muestra en la **FIGURA N° 29**.

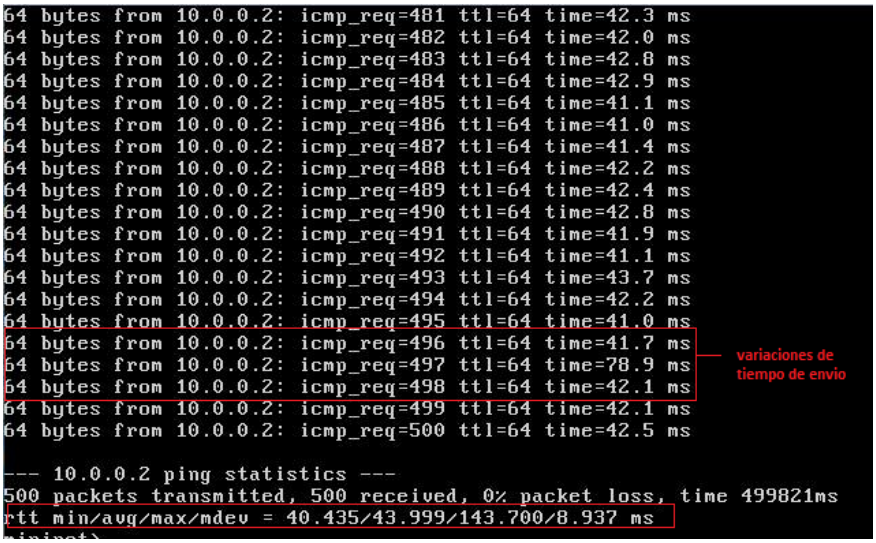
FIGURA N° 29: PING MANUAL PARA LA OBTENCIÓN DE LA LATENCIA



Fuente: Fabián Gallegos, Catherine Yánez

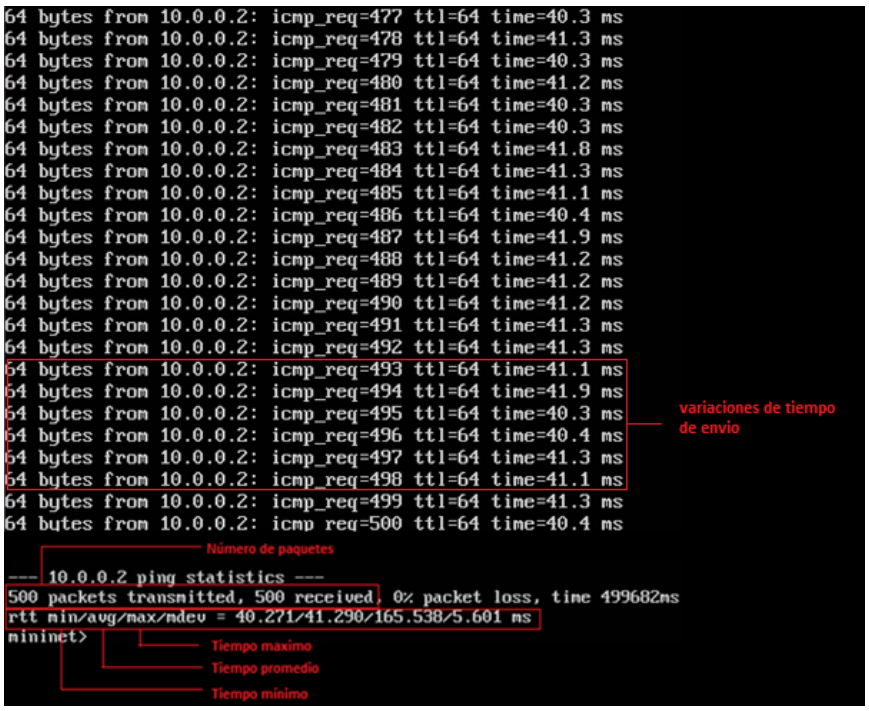
En la **FIGURA N° 30, 31** y **32** se muestran los resultados de latencia obtenidos con cada uno de los controladores considerando el ping de h1 y h2 con 500 pings los demás resultados se muestran en el **ANEXO N° 2**:

FIGURA N° 30: LATENCIA ENTRE H1 Y H2 CON 500 PINGS PARA POX



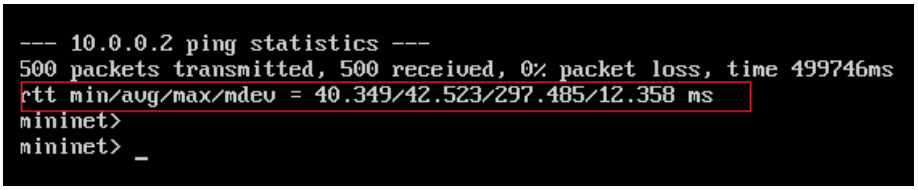
Fuente: Fabián Gallegos, Catherine Yánez

FIGURA N° 31: LATENCIA ENTRE H1 Y H2 CON 500 PINGS PARA RYU



Fuente: Fabián Gallegos, Catherine Yánez

FIGURA N° 32: LATENCIA ENTRE H1 Y H2 CON 500 PINGS PARA PYRETIC



Fuente: Fabián Gallegos, Catherine Yánez

En la **TABLA N° 8** se muestran nueve pings tanto para POX, RYU y PYRETIC: entre el host uno y host dos, host dos y host tres, host tres y host cuatro y con variaciones de 500, 1000 y 1500 pings, valores obtenidos en milisegundos, cada celda de la tabla muestra los valores: max, min, avg y mdev de cada ping realizado, existiendo diferencia de valores entre los controladores.

TABLA N° 8: LATENCIA DE LOS CONTROLADORES

	POX			RYU			PYRETIC		
	H1-H2 (ms)	**H2-H3 (ms)**	**H3-H4 (ms)**	**H1-H2 (ms)**	**H2-H3 (ms)**	**H3-H4 (ms)**	**H1-H2 (ms)**	**H2-H3 (ms)**	**H3-H4 (ms)**
500	Max:143,700 Min:40.435 Avg:43.999 Mdev:8.937	Max:131.658 Min:40.290 Avg:43.665 Mdev:8.130	Max:118.496 Min:40.303 Avg:43.972 Mdev:9.390	Max:165.538 Min:40.271 Avg:41.290 Mdev:5.601	Max:172.466 Min:40.238 Avg:41.247 Mdev:5.909	Max:173.840 Min:40.250 Avg:41.359 Mdev:5.968	Max:297.485 Min:40.349 Avg:42.523 Mdev:12.358	Max:284.498 Min:40.254 Avg:42.212 Mdev:6.846	Max:287.326 Min:40.289 Avg:42.371 Mdev:8.576
1000	Max:142.563 Min:40.446 Avg:43.859 Mdev:8.647	Max:173.406 Min:40.363 Avg:43.372 Mdev:9.348	Max:204.851 Min:40.374 Avg:43.881 Mdev:9.679	Max:169.451 Min:40.291 Avg:41.287 Mdev:4.118	Max:166.360 Min:40.264 Avg:41.309 Mdev:4.082	Max:167.119 Min:40.278 Avg:41.370 Mdev:4.053	Max:347.333 Min:40.334 Avg:42.789 Mdev:14.052	Max:263.612 Min:40.277 Avg:42.265 Mdev:7.611	Max:341.392 Min:40.254 Avg:42.192 Mdev:7.995
1500	Max:139.987 Min:40.610 Avg:43.310 Mdev:8.056	Max:130.776 Min:40.410 Avg:42.883 Mdev:6.565	Max:160.677 Min:40.358 Avg:43.712 Mdev:8.626	Max:169.706 Min:40.314 Avg:41.259 Mdev:3.397	Max:172.064 Min:40.308 Avg:41.337 Mdev:3.478	Max:168.630 Min:40.274 Avg:41.394 Mdev:3.505	Max:266.097 Min:40.433 Avg:42.687 Mdev:10.781	Max:289.141 Min:40.254 Avg:42.361 Mdev:8.164	Max:275.809 Min:40.276 Avg:42.249 Mdev:6.723

Fuente: Fabián Gallegos, Catherine Yánez

Análisis del indicador de latencia

En la **TABLA N° 8** se encuentran los valores de latencia para cada controlador, es necesario aclarar que al hacer 500, 1000 y 1500 pings, existe un valor promedio (avg), razón por la cual usamos únicamente estos valores para obtener un promedio final para 500, 1000 y 1500 pings que se obtienen aplicando la siguiente fórmula:

$$500\ o\ 1000\ o\ 1500 = \frac{avg(h1, h2) + avg(h2, h3) + avg\ (h3, h4)}{3}$$

Los resultados con los promedios finales se muestran en la **TABLA N° 9.**

TABLA N° 9: TABLA RESULTADO DE LA LATENCIA DE LOS CONTROLADORES

	500 ms	**1000 ms**	**1500 ms**
POX	43,87	43,7	43,3
RYU	41,30	41,32	41,33
PYRETIC	42,36	42,41	42,42

Fuente: Fabián Gallegos, Catherine Yánez

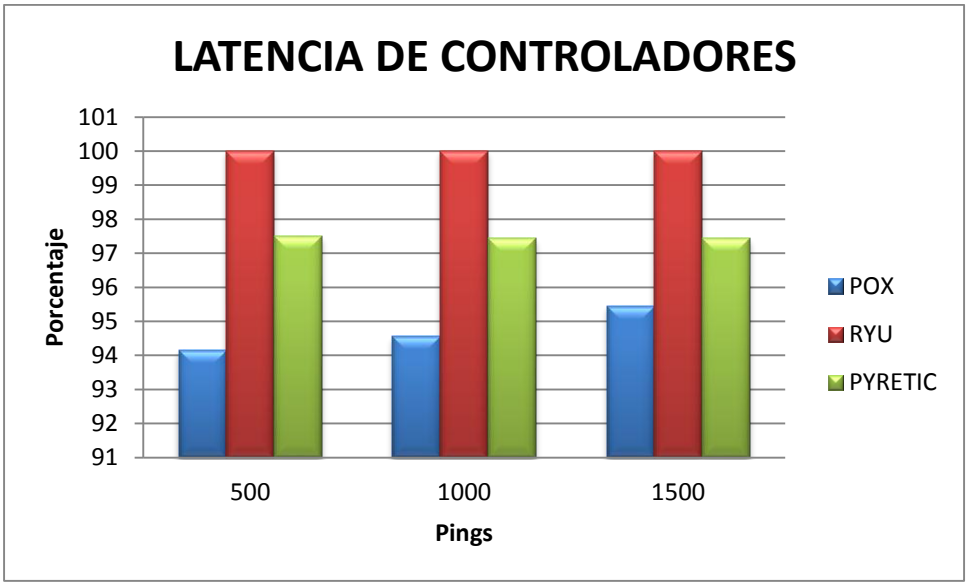
Para la correcta toma de decisiones se aplica el método tabular de estadística descriptiva, definiendo al 100% para el valor menor. Los demás datos se los obtiene usando una "regla de tres simple directa" **TABLA N° 10.**

TABLA N° 10: PORCENTAJE DEL PROMEDIO DE LA LATENCIA DE CONTROLADORES

	500	1000	1500
POX	94.14%	94.55%	95.45%
RYU	100%	100%	100%
PYRETIC	97.49%	97.43%	97.43%

Fuente: Fabián Gallegos, Catherine Yánez

FIGURA N° 33: PORCENTAJE DEL PROMEDIO DE LA LATENCIA DE CONTROLADORES



Fuente: Fabián Gallegos, Catherine Yánez

Los resultados descritos en la **TABLA N° 10** y **FIGURA N° 33** indican que para el caso de 500, 1000 y 1500 pings, el controlador que posee menor latencia es RYU, por lo tanto, se lo categoriza como *excelente* (100%) debido a que en cada 'ping' **FIGURA N° 33** el tiempo de envió de un ping no varía demasiado con el tiempo de envió del siguiente. El

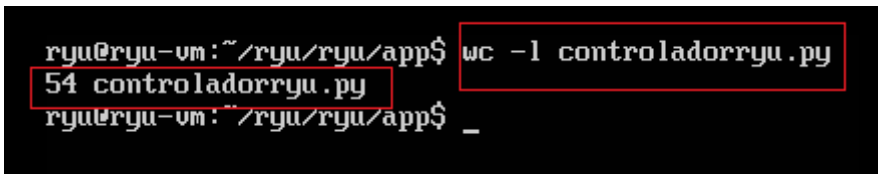
controlador POX (500: 94.14%, 1000: 94.55%, 1500: 95.45%) y PYRETIC (500: 97.49%, 1000: 97.43%, 1500: 97.43%) son *muy buenos* porque a diferencia del controlador RYU, el tiempo de envió de un paquete varia notablemente con el tiempo de envió del siguiente **FIGURA N° 30**y **FIGURA N° 32.**

Una observación interesante que se pudo recatar de los resultados de la **TABLA N° 9** es que aun siendo RYU el controlador con menor latencia se puede observar que mientras más pings son enviados su latencia aumenta, a diferencia de POX que mientras más pings se envían su latencia decrece, PYRETIC es impredecible en este aspecto. Considerando que en una red el número de paquetes es notablemente grande, la latencia del controlador POX en algún momento llegará a ser igual o menor que la latencia de RYU.

c) **Líneas de código:** Tanto POX, RYU y PYRETIC manejan sus propias formas librerías y propias maneras de ejecución. Es necesario determinar las líneas de código de cada controlador, conociendo que cada uno están destinados a cumplir la misma función, para determinar el controlador más óptimo referente a líneas de código es decir el más fácil para programar establecemos algunas características y parámetros: cantidad de líneas de código y el más compacto.

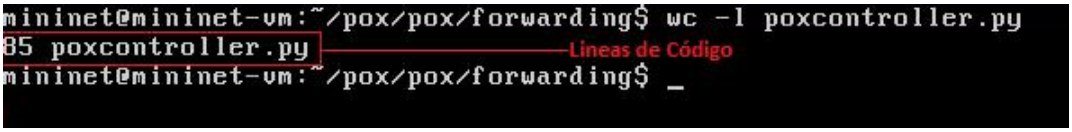
Cantidad de líneas de código: Este indicador hace referencia al aspecto cuantitativo al momento de desarrollar tanto el controlador como la red virtual, para obtener estos resultados se utilizó la instrucción de Linux : wc –l 'nombre del archivo.py' en la **FIGURA N° 34**, **35, 36** se muestra los resultados obtenidos en líneas de código en el caso de cada controlador:

FIGURA N° 34: LÍNEAS DE CÓDIGO DEL CONTROLADOR RYU



Fuente: Fabián Gallegos, Catherine Yánez

FIGURA N° 35: LÍNEAS DE CÓDIGO DEL CONTROLADOR POX



Fuente: Fabián Gallegos, Catherine Yánez

FIGURA N° 36: LÍNEAS DE CÓDIGO DEL CONTROLADOR PYRETIC

Fuente: Fabián Gallegos, Catherine Yánez

Es necesario mencionar que el controlador POX es el único de los tres controladores que permite programar la creación de la red virtual, la llamada al controlador POX y la ejecución de la red en un único archivo a diferencia de los controladores RYU y PYRETIC que necesitan un comando de mininet para crea la red y otro comando para ejecutar el controlador, razón por la cual es necesario conocer las líneas de código utilizadas de este único archivo de POX **FIGURA N°37.**

FIGURA N° 37: LÍNEAS DE CÓDIGO DE LA MININET POX

Fuente: Fabián Gallegos, Catherine Yánez

Una vez conocido el número de las líneas de código es necesario realizar un cuadro que muestra los datos para este parámetro **TABLA N° 11.**

TABLA N° 11: PROMEDIO DE LAS LÍNEAS DE CÓDIGO PARA LOS CONTROLADORES

	Líneas de código			TOTAL
	Controlador	**Ejecución Red virtual**	**Archivo final**	
RYU	54	3	-	57
POX	85	-	52	137
PYRETIC	32	3	-	35

Fuente: Fabián Gallegos, Catherine Yánez

En la **TABLA N° 11** se observan los resultados finales de las líneas de código, en las columnas: **ejecución de la red virtual** y **Archivo final** se observa la diferencia entre el controlador POX y los dos restantes, por ello los resultados cuantitativos no será un factor relevante pero servirá para el análisis posterior.

El más compacto: Este parámetro se refiere a que tanto la creación de la red virtual en mininet y la llamada y ejecución del controlador, se los puede compactar en un solo archivo. En la **TABLA N° 11** se aprecia que en el escenario con el controlador POX es el único de los tres que posee esta característica. El escenario con el controlador RYU y PYRECTIC necesita una instrucción para la creación de la red virtual con mininet y otra instrucción para la ejecución del controlador, es decir en estos dos escenarios se necesitan instrucciones independientes.

Análisis del indicador Líneas de Código: Dentro de este indicador se consideraron dos parámetros: Cantidad de líneas de código y el escenario con el controlador más compacto. De la **TABLA N° 4** se asignan valores a cada parámetro de manera cualitativa del 1 al 5 considerando 5 como el más alto.

Con estos datos se obtienen porcentajes y considerando que son dos los parámetros a evaluar; 10 puntos es el 100% y de esta manera a través de una "regla de tres simple" se conocerán los porcentajes para cada escenario. **TABLA N° 12**

TABLA N° 12: PORCENTAJE DEL PROMEDIO DE LAS LÍNEAS DE CÓDIGO

	N° LINEAS DE COGIDO	**MAS COMPACTO**	TOTAL	**%**
POX	3	5	**8**	80
RYU	5	1	**6**	60
PYRETIC	5	1	**6**	60

Fuente: Fabián Gallegos, Catherine Yánez

De acuerdo a los porcentajes obtenidos POX es *muy bueno* (80%) debido a que a través de un único archivo **FIGURA N°7** se puede programar manualmente tanto los enlaces, tiempos, utilización de la capacidad del procesador, es decir, se puede disponer de la red virtual completa y el uso de funciones de la mininet: ping, iperf, xterm, nodes. Además dentro de este archivo consta la creación y ejecución de la red y la llamada al controlador; obteniendo de esta manera un mayor control sobre la red. Mientras que RYU (60%) y PYRETIC (60%) son *buenos* porque en el escenario con cada controlador, ambos ejecutan la creación de la red y la llamada al controlador en comandos independientes.

3.5 DETERMINACIÓN DEL CONTROLADOR MÁS ÓPTIMO

A través del análisis que se realizó de los controladores se escogerá el controlador más óptimo con el que se desarrollara el prototipo de la red virtual para el HOTSPOT ESPOCH.

En la **TABLA N° 13** se resumen los resultados finales de la evaluación de los indicadores con cada controlador, basándose en la **TABLA N° 4** de valores.

TABLA N° 13: VALORES FINALES PARA CADA CONTROLADOR

CONTROLADORES	RENDIMIENTO	LATENCIA	LINEAS DE CODIGO	TOTAL
POX	5	4	4	**13**
RYU	4	5	3	**12**
PYRETIC	4	4	3	**11**

Fuente: Fabián Gallegos, Catherine Yánez

Considerando que los indicadores de evaluación son tres, el posible valor máximo obtenido es quince, de los valores de la **TABLA N° 13** se realiza una regla de tres simple para obtener los porcentajes del escenario de cada controlador y de esta manera seleccionar el controlador más óptimo **TABLA N° 14**

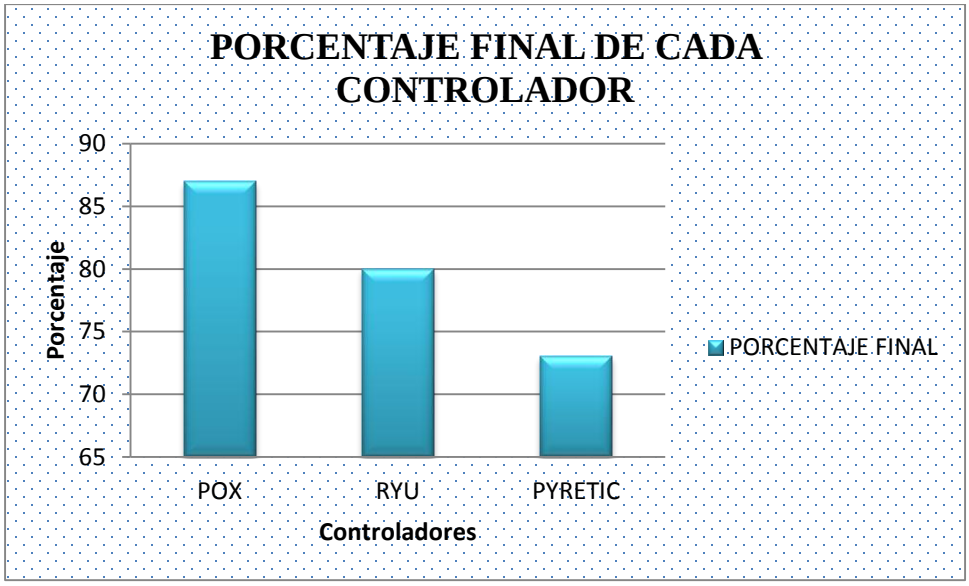
$$\% \boldsymbol{final} = \frac{TOTAL * 100}{15}$$

TABLA N° 14: PORCENTAJE FINAL DE CADA CONTROLADOR

CONTROLADORES	PORCENTAJE FINAL
POX	**87%**
RYU	80%
PYRETIC	73%

Fuente: Fabián Gallegos, Catherine Yánez

FIGURA N° 38: PORCENTAJE FINAL DE CADA CONTROLADOR



Fuente: Fabián Gallegos, Catherine Yánez

3.6 RESULTADO FINAL

De acuerdo a la **TABLA N° 14** y **FIGURA N° 38**, el controlador escogido para continuar con el trabajo investigativo es **POX** con el 87%, debido a que es *excelente* en rendimiento porque direcciona y maneja de mejor manera las peticiones en la red de esta manera la velocidad de transmisión de datos puede mejorar y el impacto de latencia en la red puede ser menor, además el controlador POX en latencia es *muy bueno* según las pruebas realizadas a pesar de obtener valores más bajos que el controlador RYU, se pudo constatar que mientras más paquetes se envían su latencia decrece, razón por la cual en algún momento su latencia será igual o menor que RYU, de igual manera POX es *muy bueno* en líneas de código ya que a través de un único archivo se puede programar manualmente tanto los enlaces, tiempos, utilización de la capacidad del procesador y se puede disponer de la red virtual completa. Además dentro de este archivo consta la creación de la red virtual en Mininet, la llamada y ejecución del controlador.

El controlador RYU obtuvo un 80% y el controlador PYRETIC tiene un 73%, ambos son muy buenos en rendimiento pero tienen valores más bajos que el rendimiento de POX, en líneas de código, son solo buenos porque ambos controladores ejecutan la creación de la red y la llamada al controlador en comandos independientes a diferencia de POX que todo lo realiza en un único archivo, en rendimiento RYU es excelente debido a que en cada ‘ping’ el tiempo de envió de un ping no varía demasiado con el tiempo de envió del siguiente ping, sin embargo mientras más paquetes son enviados su latencia aumenta, siendo no tan efectivo en cuanto a latencia al transmitir grandes cantidades de paquetes.

CAPÍTULO IV

4. CREACIÓN DEL PROTOTIPO SDN

La construcción del prototipo SDN es la parte central de la investigación propuesta ya que se simula un escenario real con dispositivos físicos y el uso de la teoría de SDN (Plano de Datos y de Control), también se trabaja con un prototipo cercano a la realidad de la red HOTSPOT-ESPOCH, de esta manera se visualiza comportamientos y características de cada prototipo y se llega a establecer diferencias a través de pruebas, que concluyen en la demostración de la hipótesis.

4.1 CREACIÓN DEL PROTOTIPO DE RED VIRTUAL SDN CASO DE LA RED HOTSPOT-ESPOCH

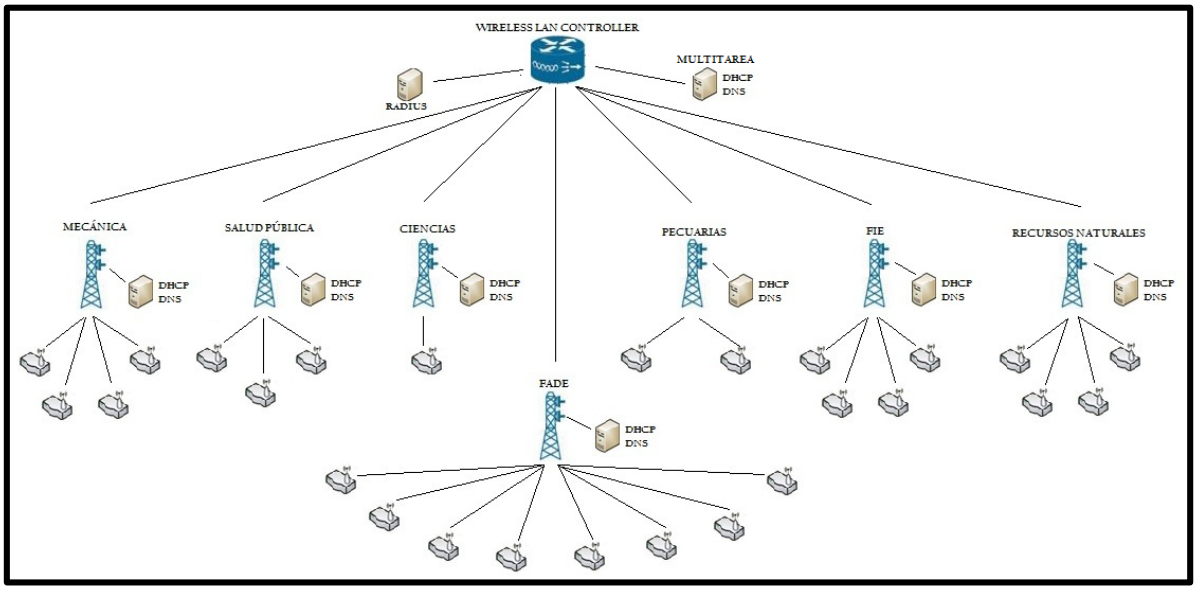
En base a los estudios realizados se determina a POX como el controlador mas idóneo para crear el prototipo SDN, en este se usarán dispositivos reales y el controlador POX catalogado como la mejor opción para esta implementación. Primero se detalla la estructura de la red actual del HOTSPOT-ESPOCH, con algunos de sus componentes ya que no se puede llevar toda la red a un prototipo, a continuación se presenta la propuesta de la investigación que dispone de una estructura SDN capaz de entender paquetes Openflow y posteriormente realizar la programación del controlador que se destina al dispositivo principal. Finalmente se realizan pruebas de rendimiento con el comando iperf que permite obtener conclusiones acertadas sobre la propuesta.

Las pruebas de tráfico son realizadas con el comando dpctl que sirve para conocer los paquetes Openflow que circulan en el prototipo SDN, a través del capturador de tráfico wireshark, además muestra algunos detalles de la red como puertos activos del switch, interfaces habilitadas, entre otras características.

4.1.1 TOPOLOGÍA DEL ESCENARIO DE LA RED HOTSPOT-ESPOCH ACTUAL

La red HOTSPOT-ESPOCH tiene varios puntos de acceso principales de los cuales uno de ellos es el Espoch-Wifi, en la **FIGURA N° 39** se presenta la topología actual de la red inalámbrica establecida en la ESPOCH, información obtenida desde DTIC: Ing. Roberto Morales también se recurre a cada Facultad para obtener información más detallada de los sectores wifi .

FIGURA N° 39: TOPOLOGÍA ACTUAL DE LA RED INALÁMBRICA DE LA HOTSPOT- ESPOCH



Fuente: Fabián Gallegos, Catherine Yánez

Elementos de la topología actual HOTSPOT- ESPOCH: A continuación se especifican los elementos físicos de la Espoch-wifi, con sus características de la **FIGURA N° 39:**

- **Wireless LAN Controller:** De la empresa CISCO, desde aquí se maneja la configuración de los Access Point de la red inalámbrica de cada una de las facultades.
- **Servidor multitarea:** Es un servidor CENTOS que sirve como DHCP y DNS para proveer de internet al rectorado y al sector administrativo de la ESPOCH
- **Siete Antenas:** Receptoras del Wireless LAN Controller, una ubicada en cada facultad.
- **Access Point:** Todos de la empresa CISCO, internos en cada facultad:
    - **Facultad de Informática y Electrónica:** cuatro Access Point, **Fuente:** Ruth Barba Vera.
    - **Facultad de Ciencias**: un Access Point solo en el sector administrativo, **Fuente:** Rogel Miges.
    - **Facultad de Pecuarias:** dos Access Point uno en el centro de cómputo y otro en la biblioteca**, Fuente:** Victor Miranda
    - **Facultad de Recursos Naturales:** cuatro Access Point organizados en un área de wifi, **Fuente:** Franklin Cuadrado.
    - **Facultad de Salud Pública:** tres Access Point uno en cada facultad y uno en la biblioteca.
    - **Facultad de Administración de Empresas:** ocho Access Point distribuidos en todas las escuelas de la facultad, **Fuente:** Marco Ortiz
    - **Facultad de Mecánica:** tres Access Point.
- Existe un servidor DHCP, DNS para cada facultad, además de un servidor RADIUS para todas las facultades, todos ellos bajo el sistema operativo Linux.

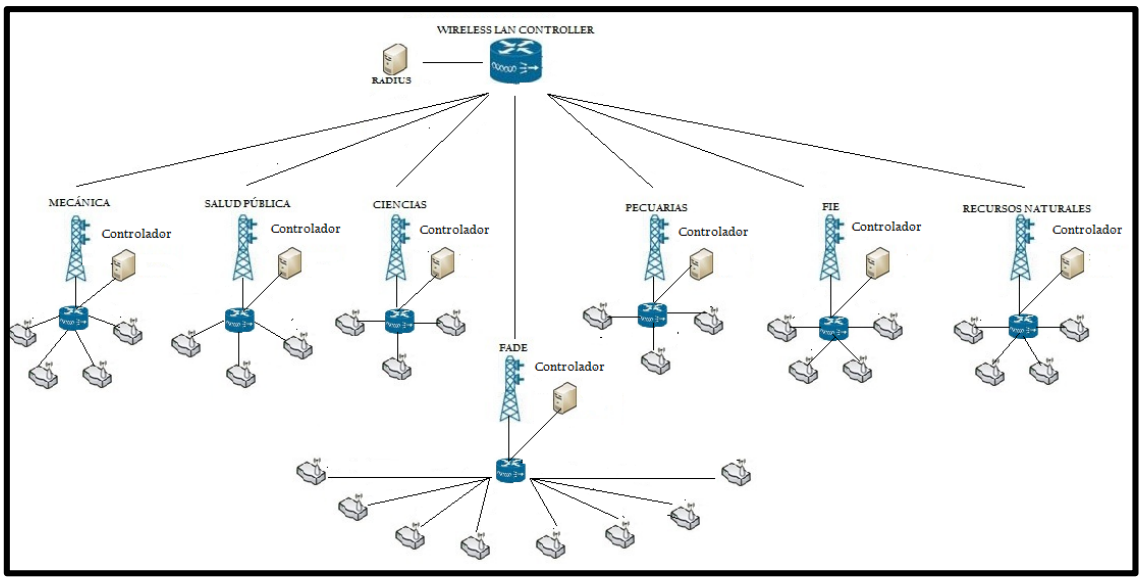
4.1.2 REQUERIMIENTOS DE LA ESPOCH-WIFI

- Cada facultad necesita más cobertura de la red inalámbrica.
- Es necesario organización e igualdad de equipos inalámbricos distribuidos en la ESPOCH, esta deficiencia se debe al costo de cada equipo CISCO.
- Restructuración total de la red wifi actual debido a las fallas y malas prestaciones de la red.
- Suprimir las redes privadas que actualmente existen en la red inalámbrica ESPOCH-wifi.
- Proveer de información de mejor manera sobre la estructura de la red inalámbrica a cada uno de los administradores de cada facultad para que de esta manera la administración de la red wifi sea más adecuada.

4.1.3 DISEÑO DE LA PROPUESTA APLICANDO LA TECNOLOGÍA SDN

La propuesta que se muestra a continuación parte del diseño actual de la red ESPOCH-WIFI, implementando la tecnología SDN a través del protocolo OpenFlow y el uso del controlador POX. En la **FIGURA N° 40** se muestra el diseño de la propuesta con características específicas:

FIGURA N° 40: DISEÑO DE LA PROPUESTA CON LA TECNOLOGÍA SDN



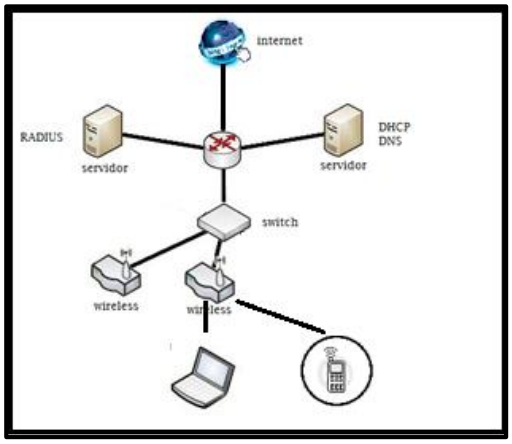
Fuente: Fabián Gallegos, Catherine Yánez

Elementos de la propuesta SDN: A continuación se especifican los elementos físicos de la propuesta SDN, con sus características de la **FIGURA N° 40:**

- **Wireless Lan Controller:** Desde aquí se maneja la configuración de los Access Point de la red inalámbrica de cada una de las facultades.
- **Servidor Radius:** En Ubuntu en el cual constan todos los usuarios que podrán acceder al a red a traves de un usuario y su contraseña.
- **Siete antenas:** Una para cada facultad, sirve para expandir la señal.
- **Siete routers:** Maneja peticiones openflow a traves del firmware OpenWRT, instalados en cada router, para comunicarse con el controlador.
- **Siete servidores controladores:** En Linux Ubuntu dentro de cada uno se encuentra el controlador basado en python llamado POX, que se encarga de instalar reglas de flujo para toda la red.
- **Access Point por facultad:** Los Access Point, no necesariamente de la empresa CISCO de esta manera el número de Access Point será equitativo en la ESPOCH.

Para la comparación se presenta un prototipo similar a la estructura de la red original de cada facultad el mismo que se muestra a continuación:

FIGURA N° 41: ESTRUCTURA DEL PROTOTIPO HOTSPOT-ESPOCH



Fuente: Fabián Gallegos, Catherine Yánez

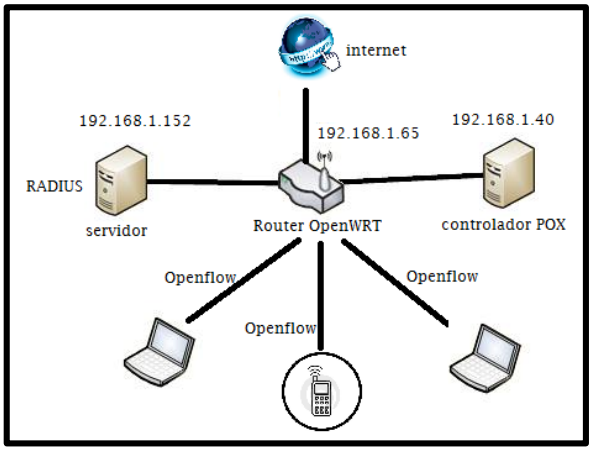
En la **FIGURA N° 41** se muestra un prototipo de la estructura que tiene la red Espoch-Wifi que consta de:

- Un servidor DHCP a través del uso del software TFTPD32 que simula las funciones del servidor
- Un servidor DNS a través del uso del software TFTPD32 que simula las funciones del servidor.
- Un Router CISCO Linksys WRH54G
- Un servidor radius en Linux Ubuntu 12.04
- Dos Access Point: Un DLINK610 y un Huawei HG530

4.1.4 TOPOLOGÍA DE LA PROPUESTA PARA LA RED SDN

La presente investigación está destinada a comprobar si la propuesta del prototipo SDN cubre o mejora las características de la actual estructura de la red Espoch - Wifi, para ello en el **FIGURA N° 42** se describe la topología del prototipo a utilizar:

FIGURA N° 42: TOPOLOGÍA DE LA PROPUESTA SDN



Fuente: Fabián Gallegos, Catherine Yánez

En la topología de la **FIGURA N° 42** se observan los siguientes dispositivos:

- Un servidor Radius (IP: 192.168.1.152): El mismo que se encarga de la autenticación de usuario para el acceso a la red.
- Un router wireless (IP: 192.168.1.65): Firmware OpenWRT habilitado para el uso de Openflow
- Controlador POX (IP: 192.168.1.40): Es la parte central del prototipo SDN que permite manejar reglas de flujo en la red: protocolo ARP, ICMP, IP.
- Dispositivos móviles conectados: laptops, tablets, celulares, etc.

La topología presentada se encuentra dentro de una misma red: 192.168.1.XX debido a que la finalidad del prototipo es conocer el funcionamiento de SDN.

Dentro del esquema de la topología existen elementos de hadware los mismos que se detallan a continuación:

Router wireless

Se usa un router wireless TP-LINK modelo TL-WR1043ND, que trabaja tambien como un switch Openflow, tiene las siguientes características:

- Un puerto WAN para internet de 10-100 y 1000 Mbps
- Cuatro puertos LAN de 10-100 y 1000 Mbps
- Tres antenas externas desmontables 5dBi
- Permite firmware OpenWRT
- Estándares: IEEE 802.11b, IEEE 802.11g y IEEE 802.11n
- Un puerto USB 2.0

Servidor Radius

Se utiliza una computadora laptop HP modelo G42-364LA con las características siguientes:

- Sistema Operativo Linux Ubuntu 12.04
- Memoria RAM de 2Gb
- Disco Duro de 500 Gb
- Procesador Intel Core i3

Servidor controlador POX

El controlador que escucha peticiones Openflow se encuentra en una laptop TOSHIBA modelo SATELITE L745 con las siguientes características.

- Sistema Operativo Linux Ubuntu 14.04
- Memoria RAM de 2 Gb
- Disco Duro de 300 Gb

- Procesador Intel i5

4.1.5 FUNCIONALIDADES DE LOS COMPONENTES EN LA TOPOLOGÍA SDN

Router Wireless

Es necesario para la investigación que el router maneje mensajes y peticiones Openflow, el firmware original TP-LINK no soporta este protocolo razón por la cual el firmware debe actualizarse a una imagen OpenWRT que soporte estas peticiones.

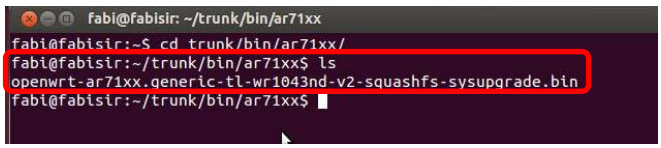
Para la mayoría de router wireless existe una imagen OpenWRT pero que no incluye el protocolo Openflow, necesitando generarse un firmware capaz de interpretar este protocolo.

Lo primero a realizarse es subir una imagen que ofrece la página oficial de OpenWRT para el router TP-LINK: *http://downloads.openwrt.org/snapshots/trunk/ar71xx/openwrt-ar71xx-generic-tl-wr1043nd-v2-squashfs-factory.bin*, el router debe ser flasheado con este firmware. Esto también se puede realizar a través de la consola usando telnel o ssh (usuario y contraseña) o desde la interfaz gráfica ver **ANEXO N° 3.**

Como segundo paso se construye un firmware que soporta OpenWRT y que dispone del protocolo Openflow para posteriormente actualizar el router con este firmware, la creación del firmware se demora aproximadamente dos horas, ver en el **ANEXO N° 4** donde se encuentra el paso a paso de su creación.

El firmware se lo reconoce como un archivo con extensión **.bin** que se encuentra bajo el directorio*:/trunk/bin/ar71xx/* **FIGURA N° 43.**

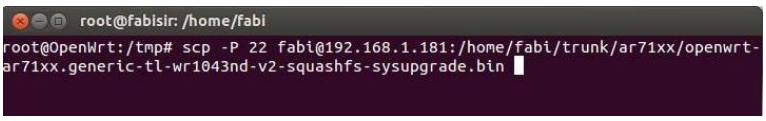
FIGURA N° 43: UBICACIÓN DEL FIRMWARE .BIN



Fuente: Fabián Gallegos, Catherine Yánez

La manera más eficiente de actualizar con esta imagen el router es a través del comando telnet, es necesario en primera instancia copiar la imagen creada al directorio tmp del router para esto desde el router se usa el comando scp: *scp –P 22 "nombre_maquina"@"direccion_ip_maquina":/"directorio_destino"* donde scp es el comando que permite copiar remotamente la opción –P que es el puerto por donde se transmiten los datos en este caso el puerto 22 para nuestro caso la línea de comando es la siguiente **FIGURA N° 44**

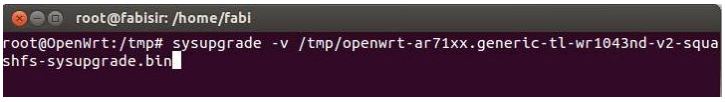
FIGURA N° 44: COPIA DEL ARCHIVO .BIN AL ROUTER



Fuente: Fabián Gallegos, Catherine Yánez

Una vez que el firmware ha sido copiado al directorio tmp del router es necesario cargar la imagen, esto se logra a través del siguiente comando: *sysupgrade –v /tmp/ openwrt-ar71xx.generic-tl-wr1043nd-v2-squashfs-sysupgrade.bin* donde sysupgrade actualiza el sistema del router y la opción –v es verbose que permite obtener información adicional de la ejecución **FIGURA N° 45**.

FIGURA N° 45: CARGA DEL ARCHIVO .BIN OPENWRT EN ROUTER



Fuente: Fabián Gallegos, Catherine Yánez

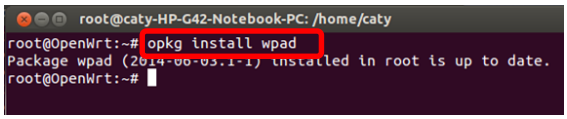
Luego de cargar la imagen el router ya tiene un nuevo firmware en este caso OpenWRT y que escucha al protocolo Openflow, tras cargar esta imagen es necesario configurar tres archivos dentro del roter, bajo el directorio etc/config: **network** donde se asigna la dirección IP del router y se añaden las interfaces LAN, el segundo archivo **wireless** donde se coloca la dirección MAC del router y se comprueban las configuraciones que se hicieron

por interfaz gráfica, finalmente se configura el archivo **openflow** donde se introduce la IP del controlador y el puerto donde escucha en el ver **ANEXO N° 5** donde se muestra más detalladamente las configuraciones de estos archivos.

Servidor radius

El servidor radius controla el acceso a la red mediante la autenticación de usuarios y contraseña previamente definidos, para disponer de un servidor radius es necesario instalar en el router OpenWRT un paquete denominado: **wpad** que se refiere a un tipo de encriptación de la red wifi, normalmente se lo conoce como WPA (Wifi Protected Access) y que trabaja en conjunto con el servidor radius, la instalación se realiza con el comando: *opkg install wpad,* **FIGURA N° 46.**

FIGURA N° 46: INSTALACIÓN DE WPAD EN EL ROUTER



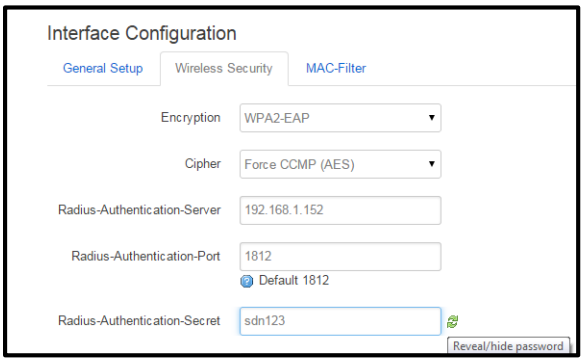
Fuente: Fabián Gallegos, Catherine Yánez

Una vez instalado este paquete en el router es necesario instalar la herramienta freeradius que convierte a la pc en un servidor radius, para ello abrimos una terminal dentro de Ubuntu y ejecutamos el siguiente comando: *apt-get install freeradius* donde apt-get permite la instalación de paquetes desde repositorios externos comúnmente situados en internet, install que permite la instalación del paquete freeradius. A continuación se modifican dos archivos: *user* donde se colocan los usuarios y contraseñas que servirán para la autenticación a la red y el segundo archivo: *clients.conf* donde se coloca la IP, el password y el nombre de la red propios del router, las configuraciones realizadas se detallan en el ver en **ANEXO N° 6**.

A través de la interfaz gráfica del router se procede a configurar la pestaña: Wireless Security donde el tipo de encriptación debe ser: WPA2-EAP seguido de esto se despliegan campos para establecer e identificar al servidor radius, el mismo que se instaló en la pc:

192.168.1.152 y escucha en el puerto 1812 (propio de radius) en la **FIGURA N° 47** se muestra la configuración realizada.

FIGURA N° 47: CONFIGURACIÓN DEL RADIUS EN LA INTERFAZ GRÁFICA

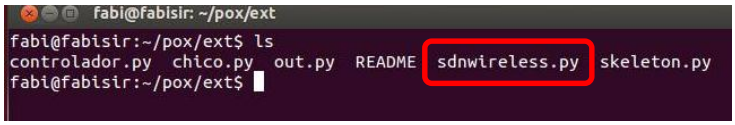


Fuente: Fabián Gallegos, Catherine Yánez

Servidor controlador POX

Este servidor tiene los archivos y configuraciones para que el controlador funcione sobre la red, para ello el primer paso es la instalación del controlador POX ver **ANEXO N° 7**. Posteriormente se crea el archivo en python con extensión .py "**sdnwireless.py"** que manejará el tráfico de red y flujos que cubre los requisitos para el presente prototipo, este archivo se ubica dentro del directorio *pox/ext* **FIGURA N° 48**, el controlador es ubicado dentro de este directorio debido a que es el lugar más conveniente ya que pox lee directamente desde este directorio los módulos creados.

FIGURA N° 48: UBICACIÓN DEL CONTROLADOR SDNWIRELESS.PY



Fuente: Fabián Gallegos, Catherine Yánez

Programación del controlador POX

La implementación del controlador permite crear un componente que hace que el router se comporte como un switch de capa dos además este componente establece flujos para el switch. En el presente prototipo la base central es la parte inalámbrica, razón por la cual la programación del componente está dirigido a esta, la parte LAN está habilitada para pruebas pero para esta investigación su programación no es necesaria, dependerá de futuras investigaciones que su programación sea tomada en cuenta. Posteriormente se añade el código del componente, en el cual se adjunta la explicación de la manera en que se trabaja con la parte inalámbrica.

Es necesaria la instalación del controlador POX. Ver en el **ANEXO N° 7** donde se muestra paso a paso la instalación de este. El componente que se presenta en este prototipo debe ubicarse bajo el directorio *pox/ext* en el que se recomienda guardar.

El código del componente sirve para que el switch reciba reglas de flujo específicas a través de mensajes de modificación de flujos, estas políticas están establecidas para el manejo únicamente de tráfico IP y ARP. Algunos protocolos no pueden ser manejados en políticas de flujo debido a que el componente del controlador necesariamente requiere de Ips de origen y destino para realizar el envío de flujos.

A continuación se presenta el código utilizado para el componente del prototipo */pox/ext/sdnwireless.py*

```
#Permite que el archivo sea reconocido como scrip python

#/usr/bin/python

#Se importan las librerías desde el núcleo de POX y para el manejo de mensajes openflow

from pox.core import core

import pox.openflow.libopenflow_01 as of

#Permite guardar información a manera de diario
```

```
log = core.getLogger()

#Se define una regla de flujo: flow0, para mensajes ARP

# flow0:

# Identificador de switch

switch0 = 000000000001

#Permite crear un flujo

flow0msg = of.ofp_flow_mod()

flow0msg.cookie = 0

flow0msg.priority = 32768

#Tipo de mensaje, ARP=0x806

flow0msg.match.dl_type=0x0806

#Puerto de entrada de mensaje de flujo

flow0msg.match.in_port = 1

#Puerto de salida de mensaje de flujo, puerto inalámbrico

flow0out = of.ofp_action_output (port = 5)

flow0msg.actions = [flow0out]

#Regla de flujo inversa: mensajes ARP

# flow0:

switch1 = 000000000001

flow1msg = of.ofp_flow_mod()

flow1msg.cookie = 0
```

```
flow1msg.priority = 32768

flow1msg.match.dl_type=0x0806

flow1msg.match.in_port = 5

flow1out = of.ofp_action_output (port = 1)

flow1msg.actions = [flow1out]

#Regla de flujo para mensajes IP

# flow0:

switch2 = 000000000001

flow2msg = of.ofp_flow_mod()

flow2msg.cookie = 0

flow2msg.priority = 32768

#Tipo de mensaje IP=0x800

flow2msg.match.dl_type=0x0800

flow2msg.match.in_port = 1

flow2out = of.ofp_action_output (port = 5)

flow2msg.actions = [flow2out]

#Regla de flujo inversa de IP

# flow0:

switch3 = 000000000001

flow3msg = of.ofp_flow_mod()

flow3msg.cookie = 0
```

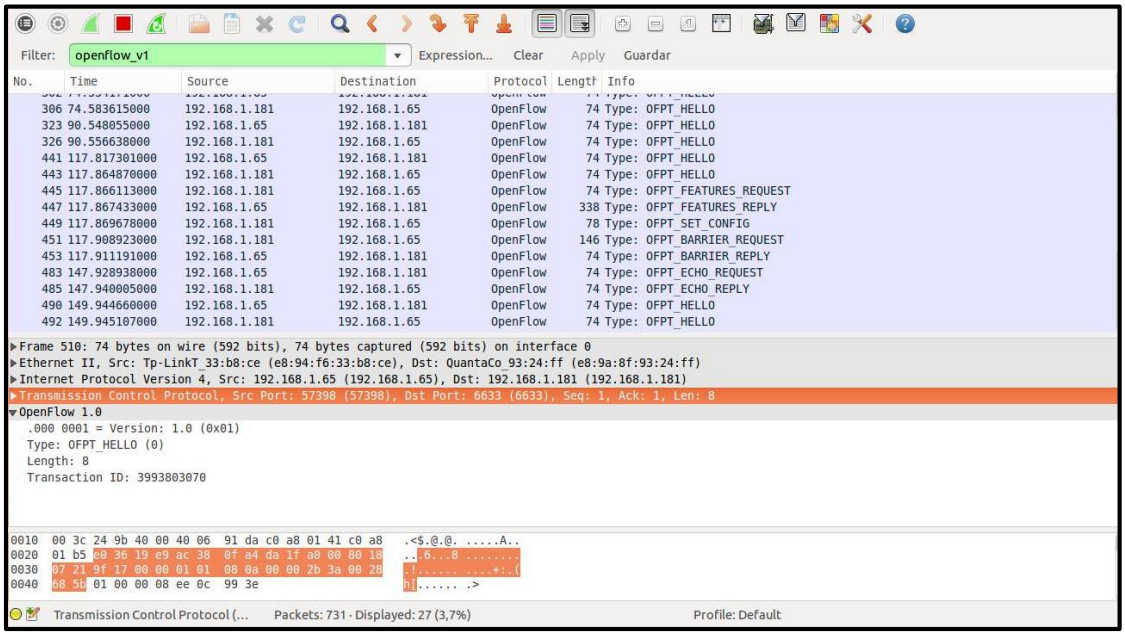
```
flow3msg.priority = 32768

flow3msg.match.dl_type=0x0800

flow3msg.match.in_port = 5

flow3out = of.ofp_action_output (port = 1)

flow3msg.actions = [flow3out]

#Metodo para la instalacion de flujos en el switch

def install_flows():

        log.info("Instalando flujos.... ?? ")

        # Instalación de flujos

        #Envio del ID de switch mas la regla de flujo

        core.openflow.sendToDPID(switch0, flow0msg)

        core.openflow.sendToDPID(switch1, flow1msg)

        core.openflow.sendToDPID(switch2, flow2msg)

        core.openflow.sendToDPID(switch3, flow3msg)

        log.info("Flujos Instalados!!! ")

def launch ():

        log.info ("....Empezar....")

        #Comienzo del método install_flows cada 15 segundos

        core.callDelayed (15, install_flows)

        log.info("Esperando por switch a conectar..... ")
```

Una vez ya creado el componente se procede a la ejecución en el servidor controlador con dirección IP 192.168.1.181 conectado al puerto físico uno del router.

4.2 PRUEBAS CON EL PROTOTIPO SDN

Conociendo que el prototipo está basado en el protocolo Openflow se hace la primera prueba para demostrar la existencia de este protocolo, para esto desde el servidor controlador se ejecuta el componente creado anteriormente y a través del uso del software wireshark se captura la existencia del protocolo Openflow y a través de las Ips la comunicación que existe entre el servidor controlador y el switch, como se muestra en la **FIGURA N° 49.**

FIGURA N° 49: CAPTURA DE CONTROLADOR OPENFLOW CON WIRESHARK



Fuente: Fabián Gallegos, Catherine Yánez

Los mensajes más comunes que existen cuando hay una conexión son:

- HELLO
- FEATURE_REQUEST

- SET_CONFIG
- FEATURES_REPLY
- BARRIER_REQUEST

HELLO: Desde el controlador al switch y desde el switch al controlador. El controlador envía su número de versión al switch y el switch responde con su número de versión Openflow.

FEATURE_REQUEST: Desde el controlador al switch. El controlador pregunta los puertos disponibles.

SET_CONFIG: Desde el controlador al switch. El controlador pregunta al switch para el envió de la caducidad de los flujos.

FEATURES_REPLY: Desde el switch al controlador. El switch responde con una lista de puertos, velocidad de puertos, tablas y acciones soportadas.

BARRIER_REQUEST/REPLY: Mensajes usados por el controlador para asegurarse que los mensajes han sido encontrados o para recibir notificaciones de operaciones completas [16]**.**

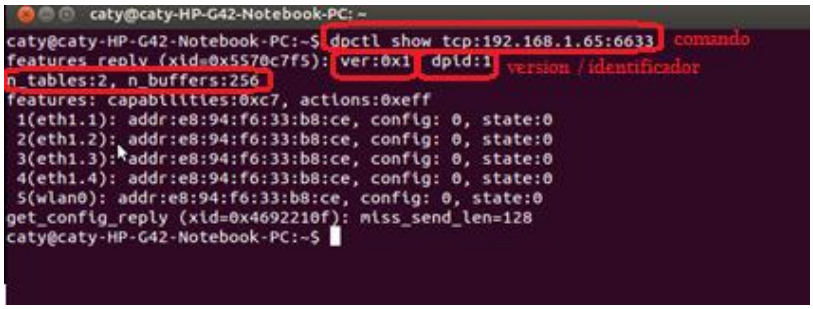
Para la siguiente prueba es necesario el uso del comando dpctl, a través del uso de este comando podemos verificar el intercambio de mensajes Openflow. Esta herramienta se va a ejecutar desde el servidor controlador ya que en este se instaló tanto el dpctl como el wireshark, ver **ANEXO N° 8** donde se muestra las instalaciones de estas dos herramientas.

En la **FIGURA N° 50** al ejecutar el comando: ***dpctl show tcp:192.168.1.65:6633*** se obtienen datos que permiten conocer las características del equipo Openflow a continuación se lista esta información:

- *ver:* Permite conocer la versión de Openflow. Hace referencia a una versión 1.0.
- *dpid:* Es un identificador que asigna el switch Openflow.
- *n_tables:* Son las políticas o reglas de flujo que se establecieron en la tabla de flujos.
- *n_buffers:* Es el espacio que el buffer del dispositivo está ocupando.

Y se lista la información que corresponde a los puertos, datos sobre cada puerto definido en el switch Openflow: su respectiva MAC y estado de cada puerto.

FIGURA N° 50: USO DEL COMANDO DPCTL SHOW TCP

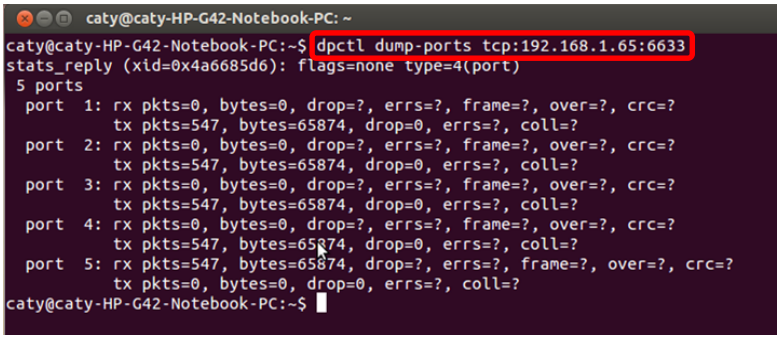


Fuente: Fabián Gallegos, Catherine Yánez

La siguiente prueba consiste en la ejecución del comando *dump-ports* el mismo que ayuda a obtener datos específicos de los puertos físicos del switch Openflow **FIGURA N° 51**, el comando se ejecuta de la siguiente manera:

dpctl dump-ports tcp:192.168.1.65:6633

FIGURA N° 51: USO DEL COMANDO DPCTL DUMP-PORTS



Fuente: Fabián Gallegos, Catherine Yánez

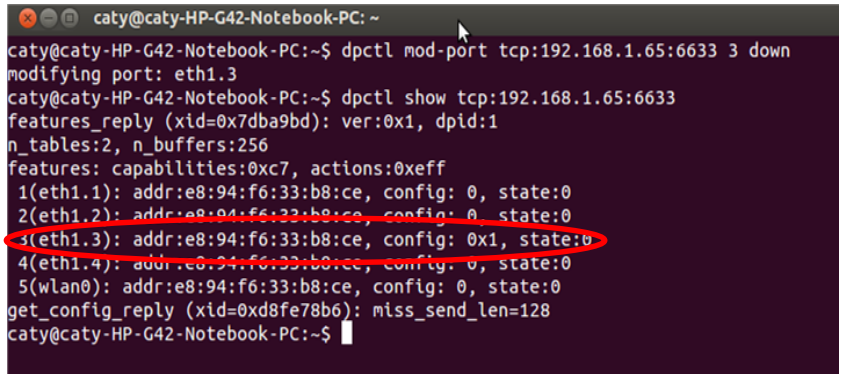
En la **FIGURA N° 51** se observa varia información que se obtiene con la ejecución del comando dump-ports, los datos que se listan son los siguientes:

- Número de puertos que escuchan mensajes Openflow en este caso 5 puertos.
- Información de cada puerto que escucha mensajes Openflow: transmisión, recepción de paquetes así como también paquetes eliminados y con error, si no existe error en la transmisión de mensajes se identifica con un signo de interrogación.

Una de las pruebas realizadas consiste en la modificación de uno de los puertos, en este caso se va a apagar el puerto tres (**eth1.3**) para comprobar el estado del puerto es necesario ejecutar el comando ***dpctl show tcp*** donde la configuración del puerto cambia de 0 a 0x1 este cambio indica que el puerto se deshabilita **FIGURA N° 52**, el comando usado para deshabilitar el puerto es el siguiente:

Dpctl mod-port tcp:192.168.1.65:6633 3 down

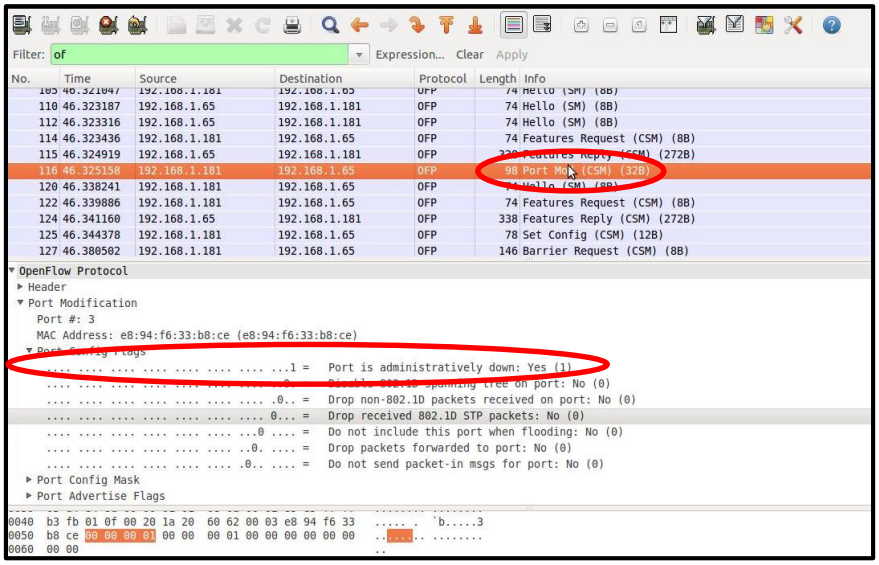
FIGURA N° 52: USO DEL COMANDO DPCTL MOD-PORT TCP



Fuente: Fabián Gallegos, Catherine Yánez

De igual manera al capturar con la herramienta Wireshark en el **FIGURA N° 53** se puede obtener información de que el puerto ha sido deshabilitado a través del mensaje **Port Mod** he indica que el puerto administrativamente ha sido deshabilitado**: Port is administratively down: Yes (1)**

FIGURA N° 53: WIRESHARK, PUERTO DESHABILITADO

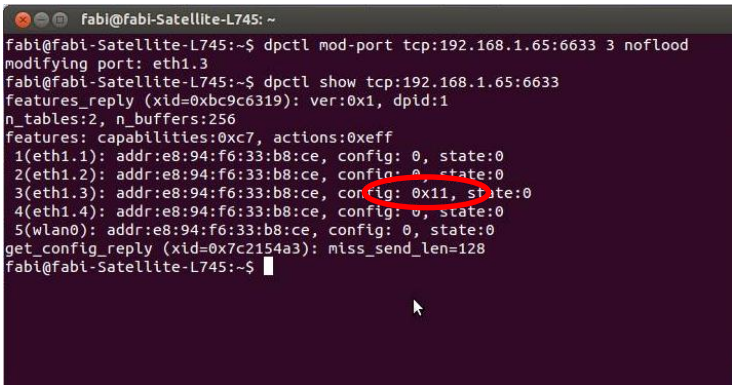


Fuente: Fabián Gallegos, Catherine Yánez

Otra prueba realizada es deshabilitar la inundación (flooding) de paquetes en puertos especificios en este caso el eth 1.3 en la **FIGURA N° 54** se muestra la ejecución del comando que permite esta característica, el comando que permite esta función es:

dpctl mod-port tcp:192.168.1.65:6633 3 noflood

FIGURA N° 54: USO DEL COMANDO DPCTL MOD-PORT NOFLOOD

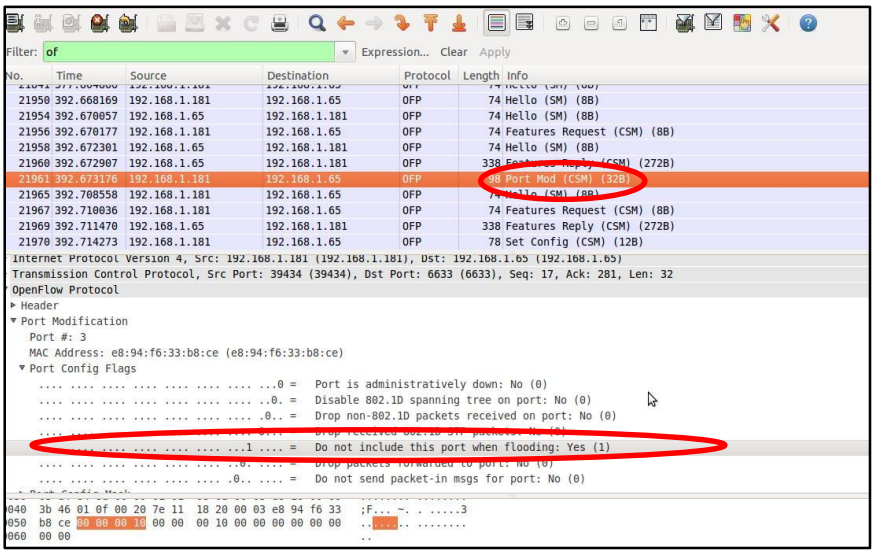


Fuente: Fabián Gallegos, Catherine Yánez

La ejecución de la instrucción anterior refleja que en el puerto tres (eth 1.3) se ha deshabilitado para que al enviar una inundación llegue a todos los puertos excepto a este, al

ejecutar el comando ***dpctl show*** se puede ver como la configuración de este puerto pasa de 0 a 0x11 esto indica que el puerto ya está deshabilitado para flooding. En este caso en la **FIGURA N° 55** se muestra una captura con Wireshark con el mensaje Port Mod indica que el puerto cambio sus características en este caso no se incluirá el puerto para flooding: **Do not include this port when flooding: Yes (1)**

FIGURA N° 55: WIRESHARK, PUERTO DESHABILITADO PARA FLOODING



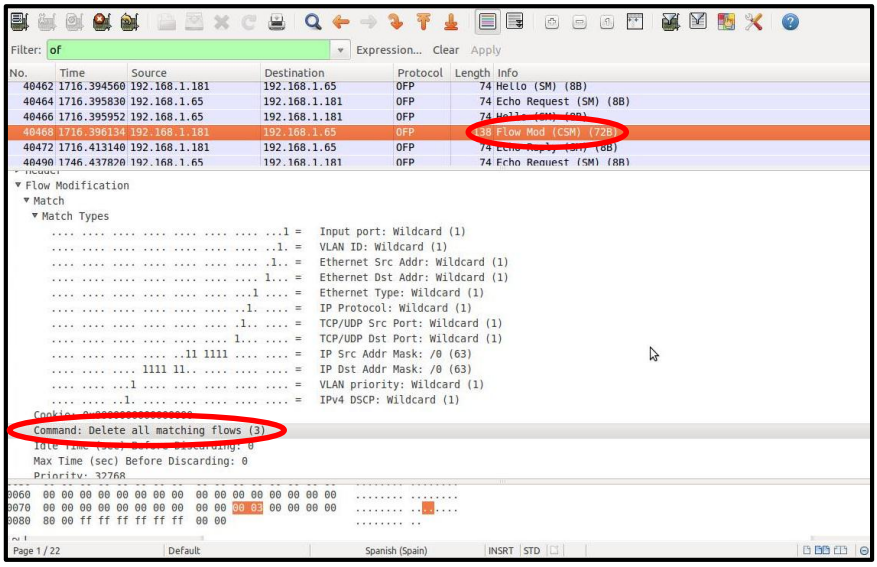
Fuente: Fabián Gallegos, Catherine Yánez

La última prueba con la que cuenta Openflow permite eliminar todos los flujos existentes en el switch para ello se ejecuta el siguiente comando:

dpctl del-flows tcp:192.168.1.65:6633

En la **FIGURA N° 56** se muestra la captura que se hizo con la herramienta Wireshark donde la ejecución de este comando se demuestra a través de un mensaje **Flow Mod** en la información adicional de este mensaje se muestra la eliminación de tres flujos: **Delete all matching flows (3)**

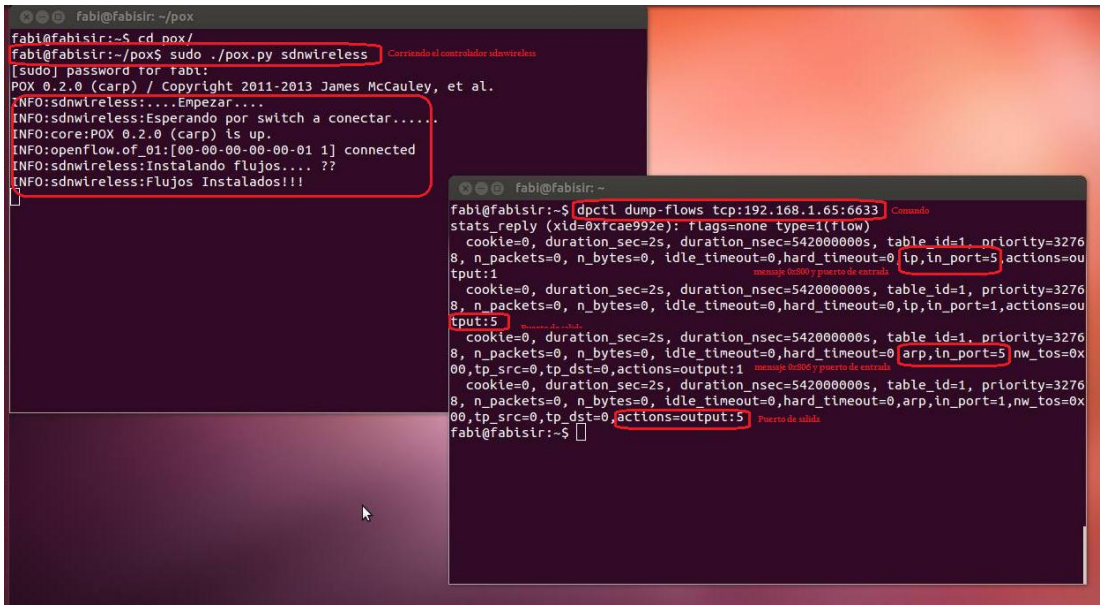
FIGURA N° 56: WIRESHARK FLOW MOD



Fuente: Fabián Gallegos, Catherine Yánez

La prueba para la constatar que el controlador está funcionando en el prototipo, se la realiza a traves del comando *dpctl dump-flows*, para obtener los resultados es necesario que el controlador se esté ejecutando en la **FIGURA N° 57** se muestra los resultados:

FIGURA N° 57: EJECUCIÓN DEL CONTROLADOR SDNWIRELESS.PY



Fuente: Fabián Gallegos, Catherine Yánez

En la **FIGURA N° 57** después de la ejecución del controlador se puede observas las dos reglas de flujo que se programaron: mensajes ARP, mensajes IP y puertos de salida y entrada: 1 y 5.

4.3 DEMOSTRACIÓN DE LA HIPÓTESIS PLANTEADA

Para la demostración de la hipótesis se aplica el método inductivo y la estadística descriptiva la misma que permite recopilar, organizar y analizar datos, estos datos a través de métodos gráficos ayudan a determinar las características específicas, es necesario establecer una comparación entre el prototipo SDN planteado en el presente trabajo de investigación y el actual prototipo del escenario de HOTSPOT ESPOCH a través de la medición de rendimiento de las dos redes.

Al establecer un prototipo SDN **FIGURA N° 42,** las pruebas realizadas no se efectuaron en la red original del HOTSPOT ESPOCH es por eso la necesidad de crear un escenario con similares características denominado Prototipo ESPOCH **FIGURA N° 41.**

4.3.1 INDICADORES DE EVALUACIÓN PARA LA COMPARACIÓN DE PROTOTIPOS

Los indicadores a considerar para la comparación son:

a) Rendimiento: Este indicador se medirá a través de un software denominada iperf, ver **ANEXO N° 9** el mismo que se ejecuta desde un cliente a un servidor en este caso el cliente iperf correrá en el equipo CLIENTE y el servidor iperf correrá en el servidor RADIUS para el caso del prototipo SDN y el prototipo ESPOCH. Cabe recalcar que los valores obtenidos en el cliente y servidor iperf son los mismos.

Las pruebas a realizarse son las siguientes:

1. **iperf general**

Para realizar esta prueba tanto el cliente como el servidor disponen del software iperf. El servidor iperf se ejecuta la instrucción: **iperf -s** la misma que permite escuchar peticiones desde el cliente iperf el cual usa la instrucción: **iperf -c <dirección ip servidor>**. En el **ANEXO N° 10** se muestran las pruebas realizadas para esta sección. Los resultados obtenidos se resumen en la siguiente tabla:

TABLA N° 15: IPERF GENERAL DE LOS PROTOTIPOS

	Prototipo ESPOCH	**Prototipo SDN**
	Cliente a Servidor iperf	**Cliente a Servidor iperf**
Ancho de banda (Mbits/s)	24.9	11.6
Transferencia (Mbytes)	29.9	14.1

Fuente: Fabián Gallegos, Catherine Yánez

En la **TABLA N° 15** los resultados muestran que para el prototipo ESPOCH el ancho de banda de la red y la velocidad de transferencia es mejor debido a que en el prototipo SDN las reglas de flujo necesitan pasar primero por el controlador antes de llegar a su destino por lo que la red ofrece un menor ancho de banda.

2. iperf bidireccional

Esta prueba se realizó de dos maneras: bidireccional secuencial y bidireccional simultaneo; para la primera prueba se utiliza en el cliente iperf la instrucción: **iperf -c <dirección ip servidor> -r**, donde -r es el argumento que especifica la prueba bidireccional secuencial, para la segunda prueba se usa la instrucción en el cliente iperf: i**perf -c <dirección ip servidor> -d**, donde -d es el argumento que especifica la prueba

bidireccional simultánea, en ambos casos la instrucción del servidor es: **iperf -s**. Ver **ANEXO N° 10** donde se muestra las pruebas realizadas. Los resultados se muestran en la siguiente tabla:

TABLA N° 16: IPERF BIDIRECCIONAL SECUENCIAL (-R) DE LOS PROTOTIPOS

	Prototipo ESPOCH		**Prototipo SDN**	
	Cliente → Servidor	Servidor → Cliente	Cliente → Servidor	Servidor → Cliente
Ancho de banda (Mbits/s)	24.7	21.2	15.9	32.9
Transferencia (Mbytes)	29.6	25.5	19.1	39.4

Fuente: Fabián Gallegos, Catherine Yánez

En la **TABLA N° 16** como se puede observar el prototipo ESPOCH mantiene valores similares en la prueba individual a diferencia del Prototipo SDN donde los valores tienen una variación notable, el ancho de banda del Cliente al Servidor es menor porque necesita pasar y recopilar información del controlador.

TABLA N° 17: IPERF BIDIRECCIONAL SIMULTANEO (-D) DE LOS PROTOTIPOS

	Prototipo ESPOCH		**Prototipo SDN**	
	Cliente → Servidor	Servidor → Cliente	Cliente → Servidor	Servidor → Cliente
Ancho de banda (Mbits/s)	8.78	9.87	28.1	2.37
Transferencia (Mbytes)	10.6	12.0	33.8	2.88

Fuente: Fabián Gallegos, Catherine Yánez

Los resultados de la **TABLA N° 16** y la **TABLA N° 17** representan los valores de iperf dados bidireccionalmente, es decir se realizó una medida de ancho de banda y transferencia tanto de cliente a servidor como de servidor a cliente, para los resultados "secuencial" primero se ejecuta un iperf de cliente a servidor y se obtiene un resultado, luego se ejecuta un iperf de servidor a cliente y de igual manera se obtiene un resultado; los resultados "simultáneos" se ejecuta un iperf de cliente a servidor y de servidor a cliente al mismo tiempo.

3. Jitter

El jitter es una variación de latencia pero no es latencia en sí, se recomienda tener un bajo jitter y tiempos de respuesta altos, para obtener el jitter es necesario la transferencia de paquetes UDP, en el cliente iperf se ejecuta la siguiente instrucción: **iperf -c <direccion ip servidor> -u -b10m**, donde -u es el argumento que hace referencia a UDP y -b es el argumento que permite asignar un ancho de banda deseado, en el servidor se ejecuta la siguiente instrucción: i**perf -s -u -i1** donde -i es el argumento que permite asignar un intervalo de tiempo específico en este caso 1 segundo. Las pruebas realizadas se muestran en el **ANEXO N° 10**. Los resultados obtenidos se muestran en la siguiente tabla:

TABLA N° 18: JITTER DE LOS PROTOTIPOS.

	Prototipo ESPOCH	**Prototipo SDN**
Jitter (ms)	0.860	3.264
Cantidad pérdida de datagramas	0/8444	8437/8444

Fuente: Fabián Gallegos, Catherine Yánez

Los resultados de la **TABLA N° 18** refleja que el prototipo SDN tiene un mayor jitter sin embargo los dos prototipos se encuentran dentro de un rango aceptable. Cabe considerar que el Prototipo ESPOCH da mayores prestaciones debido a que el impacto de jitter es

mucho menor permitiendo que haya un mayor número de paquetes llegados a tiempo a su destino.

La pérdida de datagramas para el caso del Prototipo SDN es casi total dando como referencia que una red SDN aplicada a redes inalámbricas no es recomendable.

b) Latencia: Este indicador se mide a través de un ping entre el cliente y el servidor para 10000, 20000 y 30000 paquetes, de estas pruebas se obtiene un tiempo máximo (Max), un tiempo mínimo (Min), Promedio (Media) tiempos medidos en milisegundos y los paquetes perdidos. Estas pruebas se muestran en el **ANEXO N° 10**, y los resultados se resumen en la siguiente tabla:

TABLA N° 19: LATENCIA DE LOS PROTOTIPOS

Pings	Prototipo ESPOCH	Prototipo SDN
10000	Max: 449 Min: 1 Media: 2	Max: 1272 Min: 0 Media: 2
20000	Max: 142 Min: 0 Media: 1	Max: 247 Min: 0 Media: 1
30000	Max: 38 Min: 0 Media: 1	Max: 1575 Min: 0 Media: 1

Fuente: Fabián Gallegos, Catherine Yánez

En la **TABLA N° 19** se observa que tanto el prototipo SDN y prototipo ESPOCH tienen el mismo valor de media para 10000, 20000 y 30000 ping esto indica que por más número de ping que se envié desde el cliente al servidor el resultado será el mismo. Dado

que los resultados de los ping en el valor de media son los mismos se opta por no realizar la tabla de valores para este indicador y así descartar este indicador para la toma de decisiones.

4.3.2 ANÁLISIS Y DISCUSIÓN DE RESULTADOS DE LA COMPARACIÓN DE PROTOTIPOS

Para el análisis se define una valoración de datos la que consta en la **TABLA N° 4.**

Rendimiento: En la **TABLA N° 15, 16, 17** y **18** se obtienen resultados de distintas ejecuciones de iperf razón por la cual se evaluará independiente a cada uno al final se hará una sumatoria total para obtener el porcentaje más alto de rendimiento entre los dos prototipos.

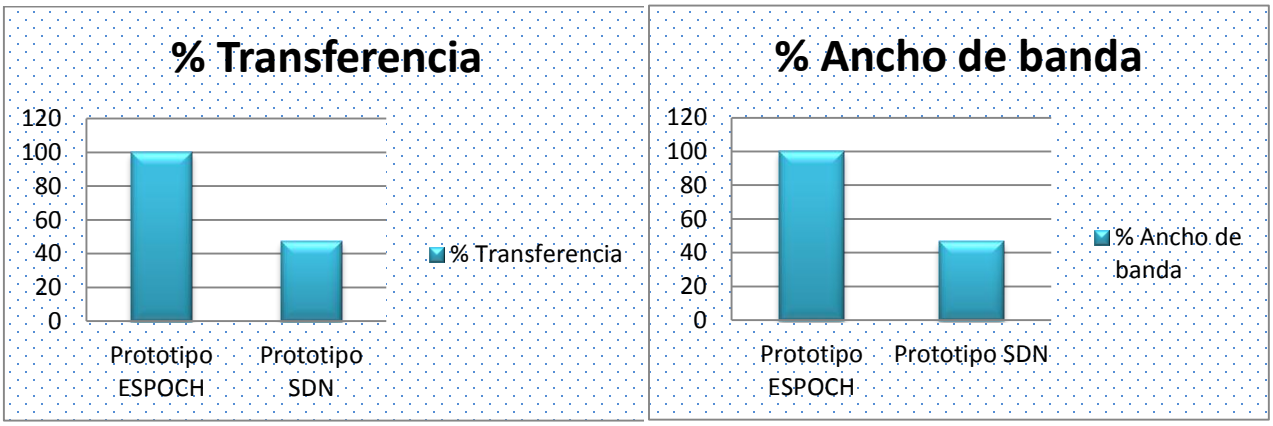
1. **Iperf general**: Para este análisis como se observa en la **TABLA N° 20** se obtienen valores de ancho de banda y transferencia para cada prototipo, para la toma de decisiones los valores serán evaluados en porcentajes, asignándole el 100% al valor más alto, los demás porcentajes se obtienen usando regla de tres simple.

TABLA N° 20: PORCENTAJES DEL IPERF GENERAL DE LOS PROTOTIPOS

	Prototipo ESPOCH	**Prototipo SDN**
	Servidor y Cliente iperf	**Servidor y Cliente iperf**
% Ancho de banda	100	46.59
% Transferencia	100	47.15

Fuente: Fabián Gallegos, Catherine Yánez

FIGURA N° 58: PORCENTAJES DEL IPERF GENERAL DE LOS PROTOTIPOS



Fuente: Fabián Gallegos, Catherine Yánez

De acuerdo a la **TABLA N° 20** y la **FIGURA N° 58** se observa que el Prototipo ESPOCH supera en porcentaje al PROTOTIPO SDN debido a que tiene un mayor ancho de banda y mayor cantidad de transferencia de datos.

2. Iperf bidireccional

Dado que en este caso se tiene dos resultados del iperf para cada prototipo **TABLA N° 16** y **TABLA N° 17**, es necesario realizar un promedio entre los dos resultados y realizar así la tabla de valores igualmente evaluados en porcentajes.

La fórmula para obtener el promedio tanto para ancho de banda como transferencia es la siguiente:

$$promedio = \frac{bandwithclienteservidor + bandwithservidorcliente}{2}$$

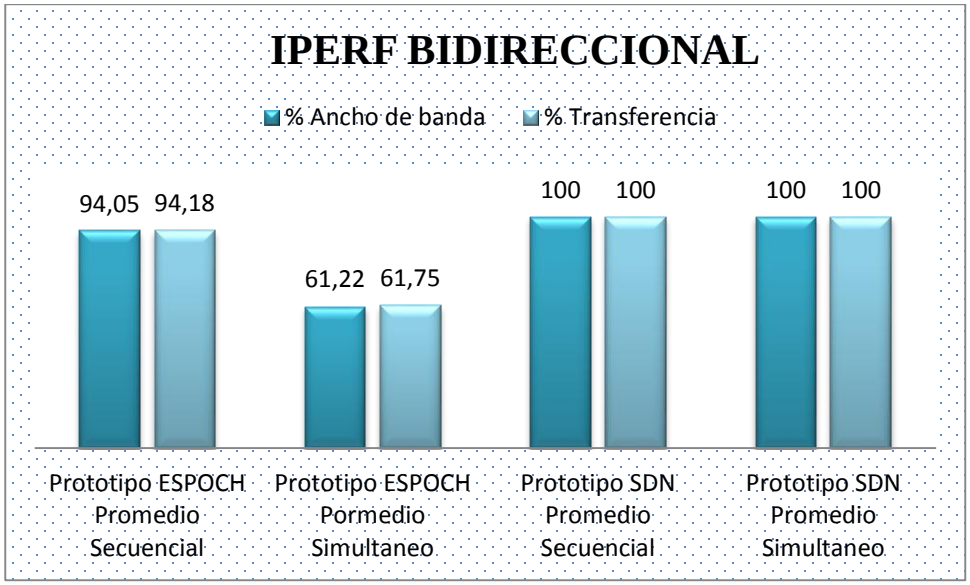
TABLA N° 21: PORCENTAJE IPERF BIDIRECCIONAL DE LOS PROTOTIPOS

	Prototipo ESPOCH		**Prototipo SDN**	
	Promedio Secuencial	**Pormedio Simultaneo**	**Promedio Secuencial**	**Promedio Simultaneo**
% Ancho de banda	94.05	61.22	100	100

% Transferencia	94.18	61.75	100	100

Fuente: Fabián Gallegos, Catherine Yánez

FIGURA N° 59: PORCENTAJE IPERF BIDIRECCIONAL DE LOS PROTOTIPOS



Fuente: Fabián Gallegos, Catherine Yánez

Después de realizar un promedio entre los valores se obtuvieron porcentajes considerando el valor más alto como el 100%, dando como resultado de esta prueba iperf bidireccional, que el prototipo SDN en ambos casos es de mayor prestación.

3. Jitter

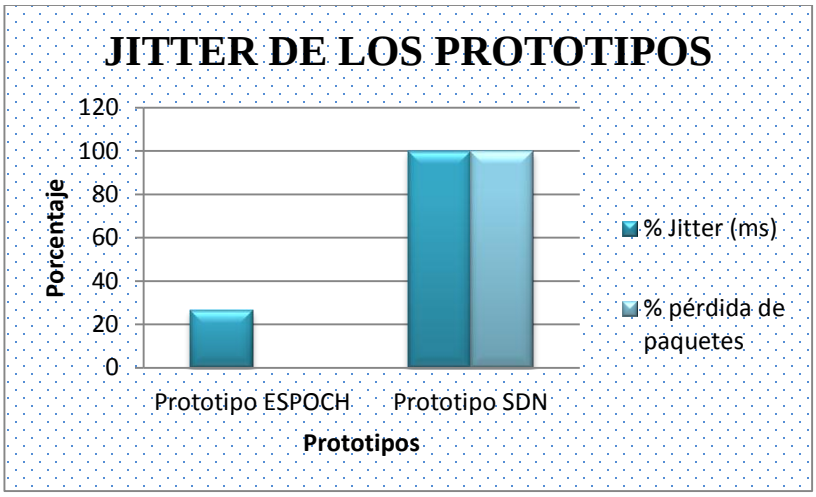
Para este análisis se toma de referencia los valores de la **TABLA N° 18**, el resultado esperado para el jitter en un escenario óptimo debe ser lo más bajo posible debido a que esto representa que un paquete llegue a tiempo o a destiempo a su destino lo recomendado es un valor menor a 100 ms. La cantidad de perdida de paquetes recomendable debe ser menor al 1%.

TABLA N° 22: PORCENTAJE DEL JITTER DE LOS PROTOTIPOS

	Prototipo ESPOCH	Prototipo SDN
% Jitter (ms)	26.35	100
% pérdida de paquetes	0	99.91

Fuente: Fabián Gallegos, Catherine Yánez

FIGURA N° 60: PORCENTAJE DEL JITTER DE LOS PROTOTIPOS



Fuente: Fabián Gallegos, Catherine Yánez

En la **TABLA N° 22** y la **FIGURA N° 60** muestran un valor alto de jitter y una gran daño de datagramas para el prototipo SDN resultado que no es óptimo para una red por la casi total pérdida de información.

- **Tabla de valoración final Indicador de rendimiento**

Para obtener este resultado final los valores de las **TABLAS N° 20, 21 y 22** serán tomados en cuenta en referencia a la **TABLA N° 4** estos resultados se observan en la **TABLA N° 23.**

TABLA N° 23: TABLA DE VALORACIÓN DEL RENDIMIENTO DE LOS PROTOTIPOS

		Prototipo ESPOCH	Prototipo SDN
Iperf general	Ancho de banda	5	3
	Transferencia	5	3
Iperf bidireccional	Ancho de banda	4	5
	Transferencia	4	5
	Ancho de banda	3	5
	Transferencia	3	5
Jitter	Jitter	4	1
	Daño de datagramas	5	1
TOTAL		33	28

Fuente: Fabián Gallegos, Catherine Yánez

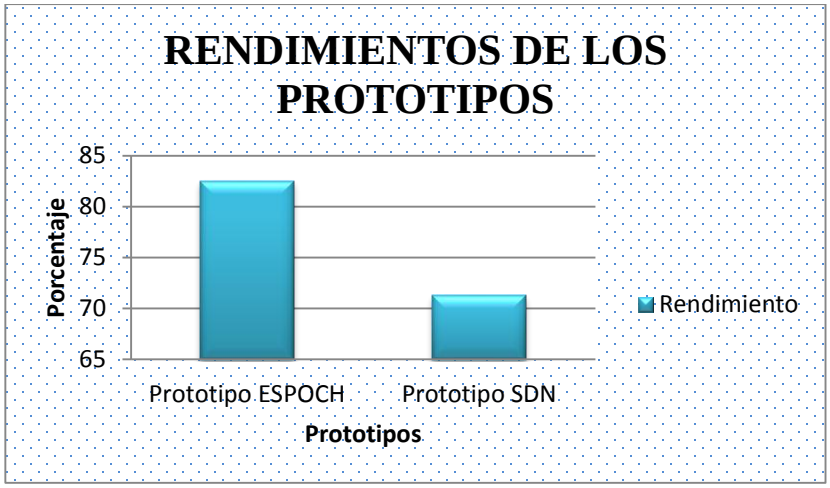
Teniendo en cuenta que son ocho los campos analizados dentro del rendimiento el valor máximo que se obtendría es 40, del valor total obtenido en la **TABLA N° 23** para cada prototipo se realizará una regla de tres simple para obtener los porcentajes los mismos que se muestran en la **TABLA N° 24.**

TABLA N° 24: PORCENTAJES FINALES DEL RENDIMIENTO DE LOS PROTOTIPOS

	Prototipo ESPOCH	Prototipo SDN
Rendimiento	**82,5**	71,25

Fuente: Fabián Gallegos, Catherine Yánez

FIGURA N° 61: PORCENTAJES FINALES DEL RENDIMIENTO DE LOS PROTOTIPOS



Fuente: Fabián Gallegos, Catherine Yánez

De acuerdo a la **TABLA N° 24** y la **FIGURA N° 61** después de realizar las pruebas de jitter, ancho de banda, transferencia y daño de datagramas se define al prototipo ESPOCH con un porcentaje de 82,5% frente al prototipo SDN que tiene un porcentaje de 71,25% estableciéndose una diferencia entre ambos.

4.3.3 RESULTADO FINAL DE LA HIPÓTESIS

H0: La implementación de un prototipo de red definida por software (SDN) no permitirá mejorar el rendimiento y gestión de dispositivos móviles del HOTSPOT-ESPOCH a través de un controlador basado en Openflow.

H1: La implementación de un prototipo de red definida por software (SDN) permitirá mejorar el rendimiento y gestión de dispositivos móviles del HOTSPOT-ESPOCH a través de un controlador basado en Openflow.

μ_{SDN}: Rendimiento del prototipo SDN

μ_{ESPOCH}: Rendimiento del prototipo ESPOCH (HOTSPOT - ESPOCH)

$$H0 = \mu_{SDN} < \mu_{ESPOCH:}$$

$$H1 = \mu_{ESPOCH} < \mu_{SDN}$$

Variables Independientes: Prototipo SDN y Prototipo ESPOCH

Variable Dependiente: Rendimiento

Al analizarse el indicador de rendimiento se pudo constatar que el prototipo SDN para una red inalámbrica no es apto, si bien es cierto la velocidad de transmisión y el ancho de banda es bueno lo más importante de una red es que la información llegue a su destino a tiempo, en el prototipo SDN los resultados muestran que el daño de información en esta red es casi total, dejando en claro que este prototipo no cumple con las expectativas planteadas en la hipótesis, por lo tanto a través de la estadística descriptiva se procede a aceptar **H0**.

Basados en todo lo comprobado y concluido la hipótesis planteada queda **negada**.

5 CONCLUSIONES

- El controlador POX es el más óptimo para el desarrollo de la red SDN HOTSPOT-ESPOCH con 87%, porque posee un mejor rendimiento de la red a diferencia de RYU con 80% y PYRETIC con 73% que tienen un rendimiento menor.
- Al realizar la comparación de los prototipos en rendimiento, se obtiene que el prototipo ESPOCH diseñado con características del fabricante, supera en un 11,25% al prototipo SDN diseñado con el controlador POX.
- El uso de la imagen Openwrt basada en Openflow que sustituye a la imagen original del router mejora el desempeño de un router sea en redes inalámbricas o redes fijas.
- El diseño SDN para la red HOTSPOT-ESPOCH, dispone en cada facultad de un controlador que está embebido en el servidor que cada una de ellas posee. Este controlador redirige la carga de datos, proporciona un firewall y añade reglas de flujo a la red.
- La propuesta física del prototipo SDN consta de un plano de control que se centra en el controlador POX el cual añade reglas de flujo ARP e IP enviadas al hardware programable bajo la directiva Openflow, además se tiene un servidor RADIUS que autentica a los usuarios simulando el escenario actual de la ESPOCH.
- Una red SDN no es apta para desarrollar redes inalámbricas, ya que en las pruebas de prototipos el daño en los datagramas es casi total: 8499/8504, su uso es apropiado en redes fijas y Data Centers.

6 RECOMENDACIONES

- Estudiar varios controladores basados en lenguajes de programación diferentes a Python tales como: Java, .Net, Ruby para buscar un controlador que mejore el rendimiento de la red.

- Se sugiere el estudio e implementación de una red SDN en la red fija de la ESPOCH para optimizar su uso y recursos físicos.

- A través del uso del controlador POX diseñar una solución SDN para el Data Center de la ESPOCH.

- Es apropiado un estudio más detallado del emulador de red virtual Mininet para explotar de mejor manera sus funciones de uso, alcances y componentes.

- Programar con el emulador Mininet una red virtual destinada para la ESPOCH, conectando dispositivos externos y realizando una configuración independientemente tanto de los hosts, controladores y switches.

RESUMEN

La investigación permite implementar un prototipo de red definida por software mediante un controlador basado en Openflow orientada a la red inalámbrica denominada HOTSPOT de la Dirección de Tecnologías de la Información y Comunicación (DTIC) en la Escuela Superior Politécnica de Chimborazo.

La investigación consiste en comparar tres controladores basados en lenguaje Python: POX, RYU y PYRETIC, obtenemos el más óptimo usando indicadores comparativos: rendimiento, latencia y líneas de código. Posteriormente construimos dos prototipos: un prototipo denominado ACTUAL que simula la red HOTSPOT y otro prototipo denominado SOLUCIÓN utilizando el controlador que resultó más adecuado. Ambos prototipos son comparados por pruebas de rendimiento; éstas permiten conocer si el prototipo SOLUCIÓN mejora la red HOTSPOT.

Los prototipos se crean y comparan usando un router y tres laptops físicas. Para obtener resultados utilizamos software iperf y wireshark que proporcionan información de rendimiento para cada prototipo. Aplicamos el método inductivo para demostrar si el prototipo SOLUCIÓN mejora la situación en la red HOTSPOT a través de estadística descriptiva que detalla valoración cuantitativa y cualitativa.

Los resultados de comparar los controladores son: POX con 87%, RYU con 80% y PYRETIC con 73%, en rendimiento y latencia. Comparando los dos prototipos obtenemos: prototipo ACTUAL con 82.5% en rendimiento y prototipo SOLUCIÓN con 71.25% en rendimiento. Concluimos que POX es el controlador más óptimo para crear el prototipo SOLUCIÓN y una solución de red definida por software no es apta para implementar redes inalámbricas. Recomendamos a investigadores profundizar el estudio e implementación de diversos controladores en distintos lenguajes de programación.

Palabras clave: <CONTROLADOR [POX]> <CONTROLADOR [RYU]> <CONTROLADOR [PYRETIC]><SOFTWARE IPERF><SOFTWARE WIRESHARK> <OPENFLOW> < DIRECCIÓN DE TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIÓN [DTIC]>

SUMMARY

This research allows to implement a prototype network using a software-defined based on an OpenFlow controller oriented to the wireless HOTSPOT network called direction of information technology and communication (DITC) in the Polytechnic School of Chimborazo.

This research consists of comparing three controllers based on Python: POX, RYU and PYRETIC, we obtain the optimum using shared indicators: throughput, latency and lines of code. Later we built two prototypes: a prototype called CURRENT simulating HOTSPOT and another prototype called network solution using the driver which was more suitable. Both prototypes are compared for performance testing; they show whether the prototype SOLUTION improves HOTSPOT network.

The prototypes are created and compared using a router and three laptops. To obtain the results iperf and wireshark were used which provide information about each prototype. Inductive method was applied to demonstrate if the SOLUTION prototype improves the situation in the HOTSPOT network through descrititive statistics which shows cualitative and cuantitative evaluation.

The results of comparing the drivers are: POX with 87%, RYU with 80% and PYRETIC with 73%, in performance and latency. When comparing both prototypes we obtain: CURRENT prototype with 82.5% in throughput and SOLUTION prototype with 71.25% yield.

It is concluded that POX is the most useful controller to create the SOLUTION prototype and a solution of a defined network by software is not suitable to implement wireless networks.

It is recommended to the researchers to make a further study and to implement different drivers in programming languages.

KEYWORDS: DRIVER (POX), DRIVER (RYU), DRIVER (PYRETIC), SOFTWARE IPERF, SOFTWARE WIRESHARK, OPENFLOW PROTOCOL, DIRECTION OF INFORMATION TECHNOLOGY AND COMMUNICATION (DITC).

BIBLIOGRAFÍA

[1] **BERNAL, P**. Entrevista: Sistema HOTSPOT - ESPOCH, Riobamba - Ecuador, 2014, Audio.

[2] **DOCUMENTACIÓN DE RYU**, Chiyoda - Japón, 2011, http://ryu.readthedocs.org/en/latest/components.htm 2014-05-11.

[3] **MODELOS SDN PROACTIVO Y REACTIVO**, https://devcentral.f5.com/articles/reactive-proactive-predictive-sdn models#.U1mZK1V5Pws, 2014-02-20

[4] **GEORGIA, K KYRIAKOS, Z** Openflow Virtual Networking: A Flow-Based Network Virtualization Architecture, Estocolmo - Suecia, 2009, 80 P. http://www.diva-portal.org/smash/get/diva2%3A302700/FULLTEXT01.pdf, 2014-22-01.

[5] **INTRODUCCION A MININET**, San Francisco - Estados Unidos, 2014, https://github.com/mininet/mininet/wiki/Introduction-to-Mininet 2014-01-25.

[6] **TUTORIAL MININET**, Washington DC - Estados Unidos, 2013, http://eventos.redclara.net/indico/getFile.py/access?contribId=1&resId=4&materialId=slides&confId=197, 2014-01-22.

[7] **JIMÉNEZ, J**. Implementación de un prototipo de una red definida por software (SDN) empleando una solución basada en hardware, Quito - Ecuador, 2013, 210 P

http://bibdigital.epn.edu.ec/bitstream/15000/6681/1/CD-5065.pdf,
2013-11-18

[8] **JIMÉNEZ, J**. Implementación de un prototipo de una red definida por software (SDN) empleando una solución basada en hardware, Quito - Ecuador, 2013, Pp 8
http://bibdigital.epn.edu.ec/bitstream/15000/6681/1/CD-5065.pdf,
2013-12-10

[9] **SDN MARCA EL FUTURO DEL NEWORKING**, Boston - Estados Unidos, 2013,
http://www.idg.es/macworld/content.asp?idn=131216
2014-01-22.

[10] **MONSANTO, C. REICH, J. FOSTER, N. REXFORD, J. WALKER, D**. Composing Software-Defined Netwoks, Lombard - Francia, 2013, 13P, http://frenetic-lang.org/publications/composing-nsdi13.pdf,
2014-02-22.

[11] **REDES DEFINIDAS POR SOFTWARE**, Sunnyvale - Estados Unidos, 2014, https://www.juniper.net/es/es/products-services/sdn
2014-07-30.

[12] **OPENFLOW: PROACTIVE VS REACTIVE FLOWS**, Washington DC - Estados Unidos, 2014, http://networkstatic.net/openflow-proactive-vs-reactive-flows/
2014-02-20.

[13] **OCAMPO, O**. Programación en Castellano, Ciudad Real - España, 2013, http://www.programacion.com/articulo/guia_de_aprendizaje_de_python_65
2013-09-29.

[14] **SOFTWARE - DEFINED NETWORKING (SDN)**, Palo Alto - Estados Unidos , 2014, https://www.opennetworking.org/sdn-resources/sdn-definition 2014-12-22.

[15] **OPENFLOW SWITCH SPECIFICATION**, Palo Alto - Estados Unidos, 2011, Pp 14, http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf 2014- 07-05.

[16] **OPENFLOW TUTORIAL**, Palo Alto - Estados Unidos, 2011, http://archive.openflow.org/wk/index.php/OpenFlow_Tutorial 2014-01-12.

[17] **PYLIBOPENFLOW**, Palo Alto - Estados Unidos, 2011, http://archive.openflow.org/wk/index.php/Pylibopenflow 2013-10-25.

[18] **POX WIKI**, Palo Alto - Estados Unidos, 2012, https://openflow.stanford.edu/pages/viewpage.action?pageId=7045197,

2014-03-08.

[19] **PYLIBOPENFLOW**, Palo Alto - Estados Unidos, 2012, http://archive.openflow.org/wk/index.php/Pylibopenflow#Why_not_use_swig.3F 2012-12-12.

[20] **SCHMITZ, O**. Software-Defined Everything: Llevando la inteligencia a un nivel superior, Washington DC - Estados Unidos, 2014, https://www.linkedin.com/pulse/20140725120736-1573471-software-defined-everything-llevando-la-inteligencia-a-un-nivel-superior 2014-09-03.

[21] **INTRODUCCIÓN A SOFTWARE DEFINED NETWORKING**, Fort Lauderdale - Estados Unidos, 2014, 60 P, http://www.citrix.com/content/dam/citrix/en_us/documents/oth/sdn-101-an-introduction-to-software-defined-networking-es.pdf, 10 - 08 - 2014.

[22] **PYTHON**, 2010, Washington DC - Estados Unidos, http://www.techterms.com/definition/python, 12-12-2013.

[23] **PYTHON BASIC SYNTAX**, 2014, Hyderabad Telangana - INDIA, http://www.tutorialspoint.com/python/python_basic_syntax.htm, 22-12-2014.

[24] **PYTHON OVERVIEW**, Hyderabad Telangana - INDIA, 2014, http://www.tutorialspoint.com/python/python_overview.htm, 22-01-2014

[25] **VELASQUEZ, W**. Emulación de una red definida por software utilizando Mininet, Madrid - España, 2014, http://www.academia.edu/5730624/Emulacion_de_una_red_definida_por_software_utilizando_MiniNet, 12-02-2014.

GLOSARIO

ACCES POINT Dispositivo de red que interconecta equipos inalámbricos en una red inalámbrica.

ANCHO DE BANDA Recursos disponibles de datos en la red, medidos en bit/s.

CISCO Empresa dedicada a telecomunicaciones.

CLIENTE Aplicación o equipo que consume servicios de otro equipo a través de una red.

CONTROLADOR Parte central de una SDN que centraliza la comunicación entre los elementos de la red

DATAGRAMA Parte parcial o entera de un paquete de datos con información de comunicación y de destino

DIRECCION MAC Identificador de 48 bits perteneciente a una dirección física o también llamada dirección hardware.

DPCTL Utilidad de Openflow que permite visualizar el control sobre una tabla de flujo simple de un switch.

EHTERNET Estándar de transmisión de datos y de conexión.

ENCRIPTACIÓN Proceso en el cual los datos se vuelven ilegibles por seguridad.

FIRMWARE Programa grabado dentro de una memoria no volátil. Dicta reglas de operación del dispositivo.

HOST Referido a los equipos en una red que utilizan o proveen servicios.

HOTSPOT Equipo de acceso que ofrece cobertura inalámbrica a través de autenticación de usuarios (usuario y contraseña).

IPERF	Herramienta para realizar pruebas en redes informáticas.
JITTER	Variación en la cantidad de latencia entre paquetes de datos.
LATENCIA	Conjunto de retardos de tiempo dentro de una red.
MININET	Emulador de red que consta de dispositivos finales, switches, routers, y enlaces.
OPENFLOW	Protocolo de comunicación para establecer el enrutamiento y manejo de tráfico en una red.
OPENWRT	Distribución Linux que permite aprovechar mejores características a base de firmware de los routers convencionales.
PING	Comando que sirve para verificar el estado de una conexión desde un host hacia otro host remoto en una red.
PROTOCOLO	Conjunto de reglas usadas por computadores para comunicarse unas con otras en una red.
PROTOTIPO	Representación limitada de un producto o solución que permite probar situaciones reales y explorar su uso.
PUERTO	Lugar por donde ingresa información, sale información o ambos. Pueden ser físicos y lógicos.
RADIUS	Gestiona el acceso de redes a través de Autenticación, Autorización y Anotación.
RED	Conjunto de dispositivos interconectados entre sí a través de un medio en donde comparten e intercambian información.
ROOT	En Unix o derivados, es el nombre de la cuenta de usuario que posee todos los atributos y derechos de los modos mono y multiusuario.

ROUTER	Dispositivo de red que destina un paquete de datos entrante a su destino.
SERVIDOR	Equipo o computador donde se aloja información y datos que están disponibles para otros dispositivos en una misma red.
SWITCH	Dispositivo de interconexión para conectar equipos en red formando lo que se llama una LAN (Red de Área Local).
TOPOLOGIA	Es la disposición física en la que se conecta una red de computadores.
TP-LINK	Proveedor global de productos de sistemas de redes.
VIRTUAL BOX	Software de virtualización de arquitecturas x86 o x64.
WIFI	Mecanismo que facilita la conexión inalámbrica de distintos equipos.
WIRESHARK	Analizador de protocolos que sirve para solucionar problemas de redes de comunicaciones.

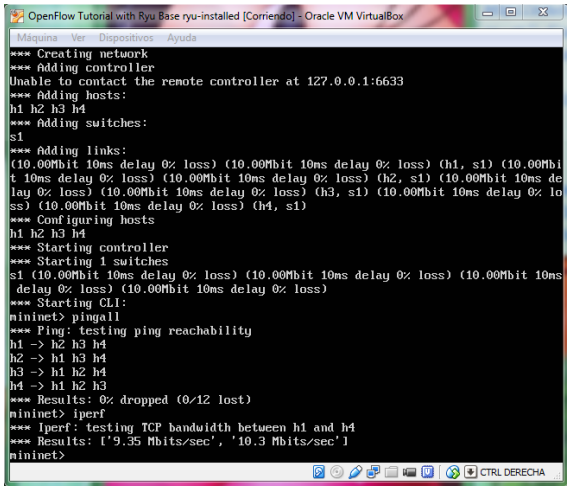
ANEXOS

ANEXO N° 1: CAPTURA DEL RENDIMIENTO OBTENIDO PARA EL CONTROLADOR RYU, POX Y PYRETIC EN CADA UNA DE LAS PRUEBAS

Captura del rendimiento obtenido para el controlador RYU en cada una de las pruebas

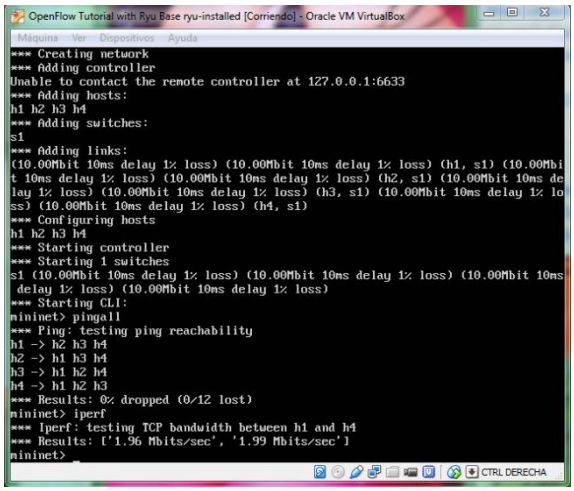
Sin perdida	**9.35 – 10.3 Mbits/sec**				
Con perdida	**Loss=1**	**Loss=2**	**Loss=3**	**Loss=4**	**Loss=5**
	1.96 – 1.99 Mbits/sec	1.34 -1.35 Mbits/sec	1.13-1.15 Mbits/sec	782-792 Kbits/sec	638-638 Kbits/seexitc

Sin pérdida

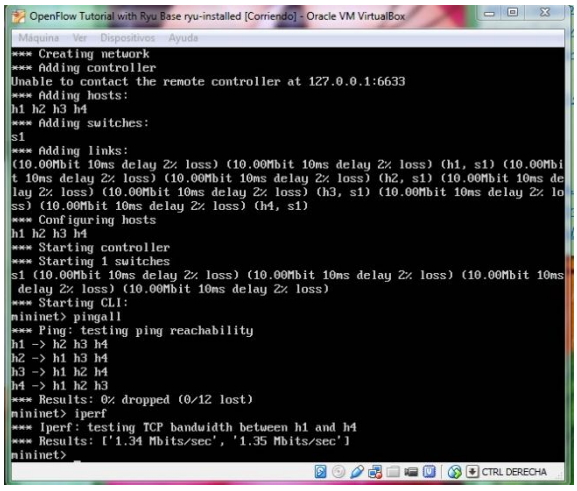


Con pérdida

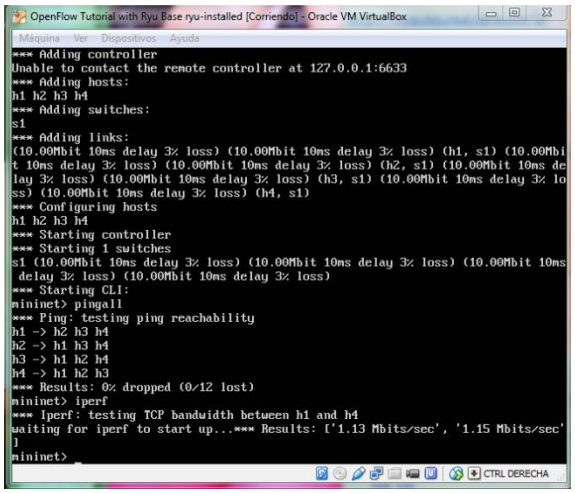
Loss=1



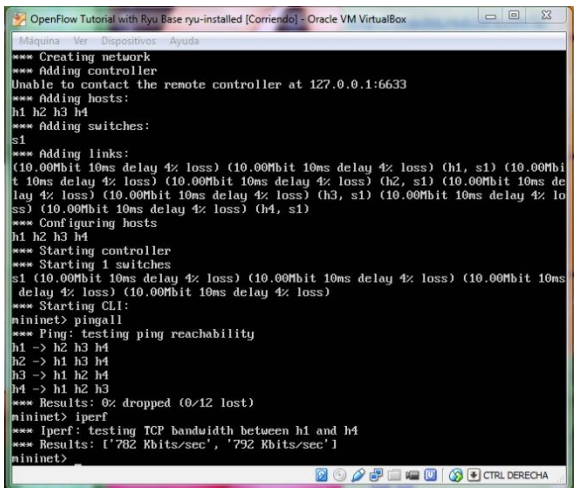
Loss=2



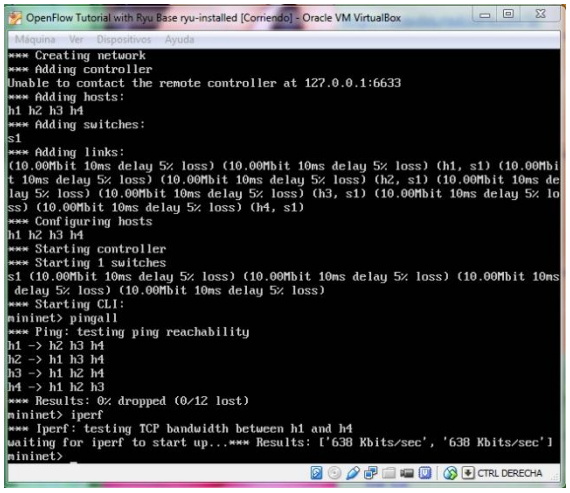
Loss=3



Loss=4



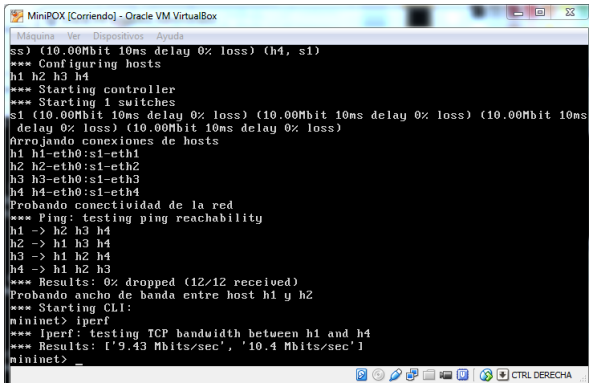
Loss=5



Captura del rendimiento obtenido para el controlador POX en cada una de las pruebas

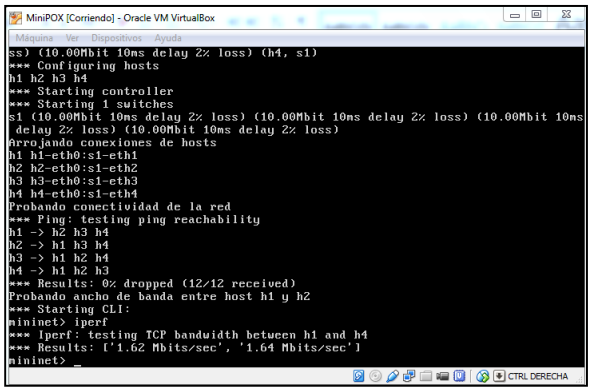
Sin perdida	**9.43 – 10.4 Mbits/sec**				
Con perdida	**Loss=1**	**Loss=2**	**Loss=3**	**Loss=4**	**Loss=5**
	1.96 – 1.99 Mbits/sec	1.62 -1.64 Mbits/sec	1.30 – 1.32 Mbits/sec	785-840 Kbits/sec	626-635 Kbits/seexitc

Sin Pérdida

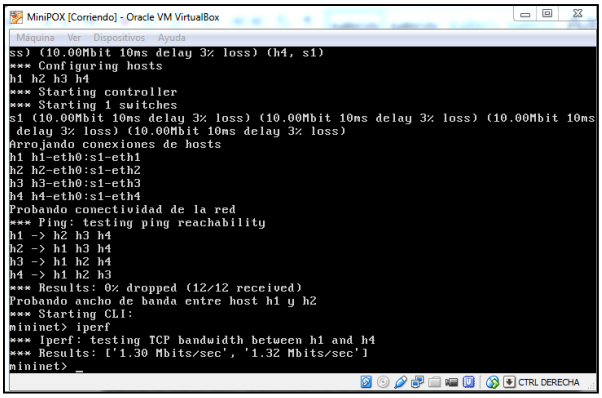


Cón Pérdida

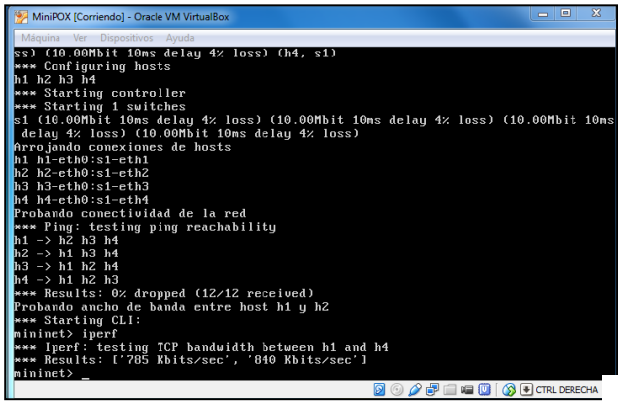
Loss = 2



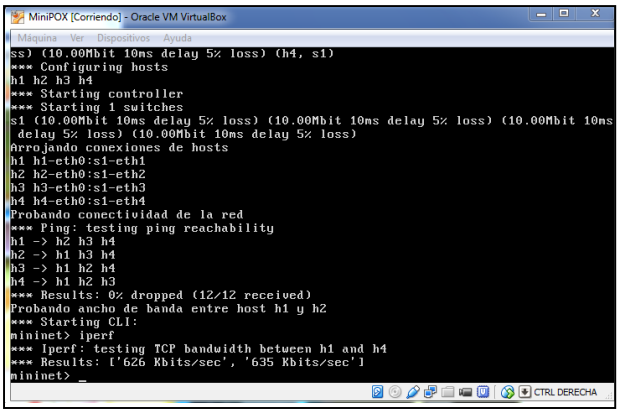
Loss = 3



Loss = 4



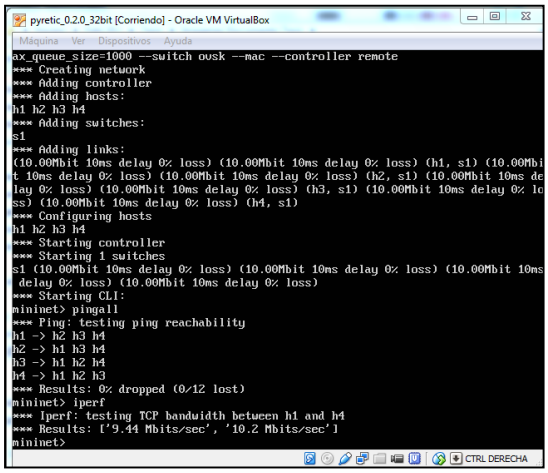
Loss = 5



Captura del rendimiento obtenido para el controlador PYRETIC en cada una de las pruebas

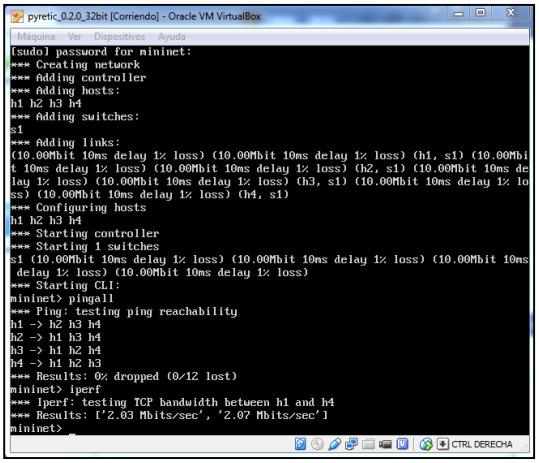
Sin perdida	**9.44 – 10.2 Mbits/sec**				
Con perdida	**Loss=1**	**Loss=2**	**Loss=3**	**Loss=4**	**Loss=5**
	2.03 – 2.07 Mbits/sec	1.37 -1.40 Mbits/sec	1.11 – 1.22 Mbits/sec	755-762 Kbits/sec	555-558 Kbits/seexitc

Sin Pérdida

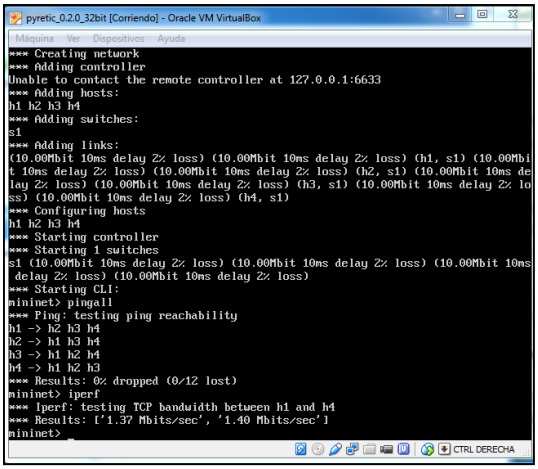


Con Pérdida

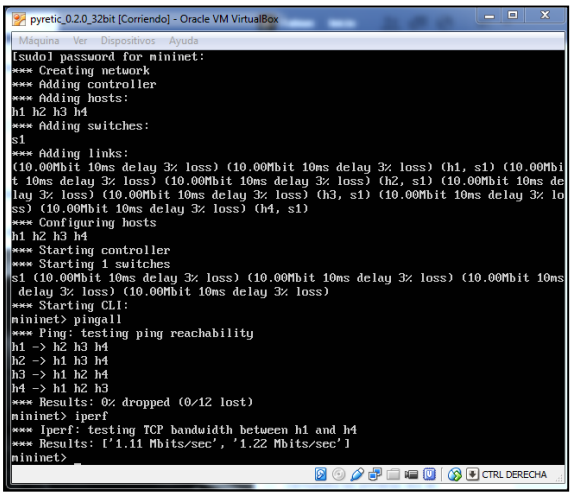
Loss = 1



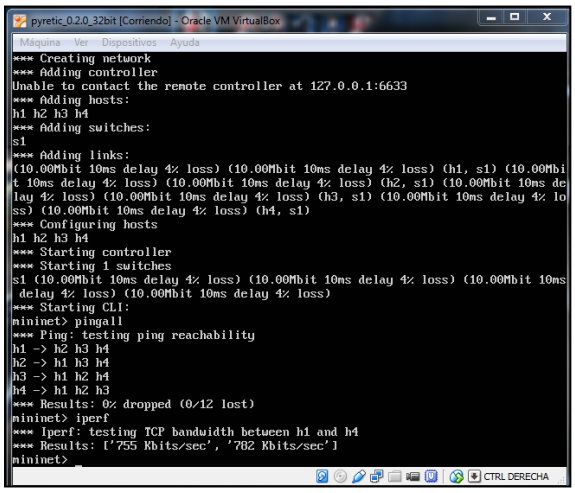
Loss = 2



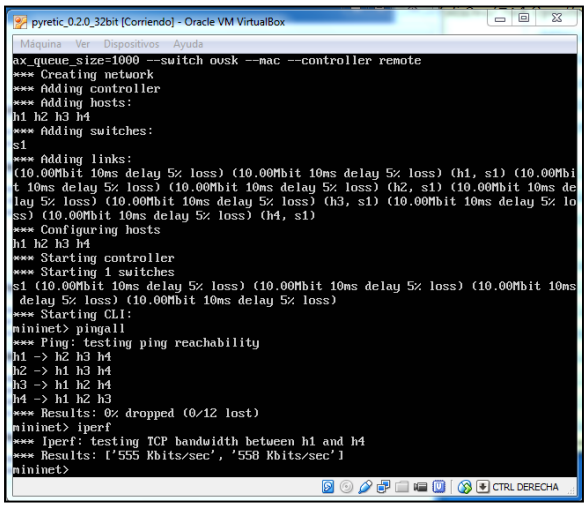
Loss = 3



Loss = 4



Loss = 5

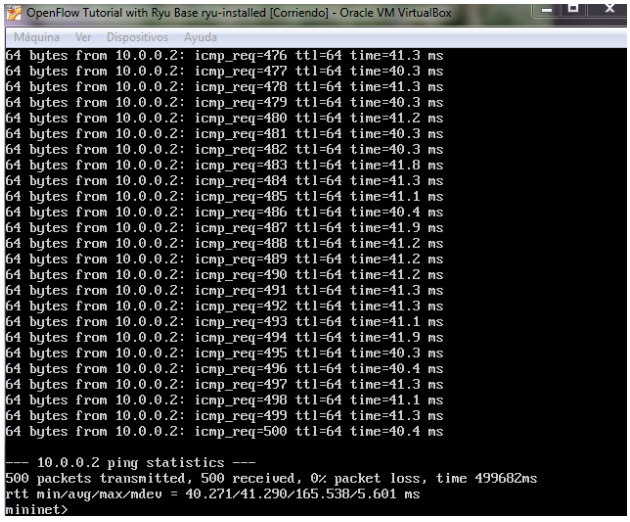


ANEXO N ° 2: CAPTURA DE LA LATENCIA OBTENIDA PARA LOS CONTROLADORES POX, RYU Y PYRETIC

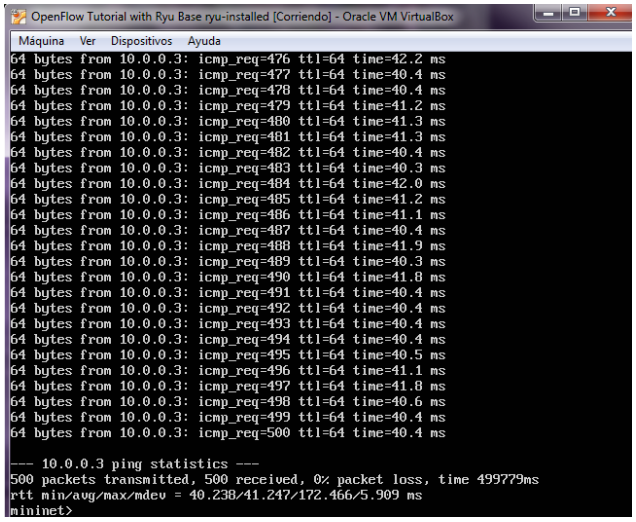
Captura de la latencia obtenida para el controlador RYU

Con 500 pings

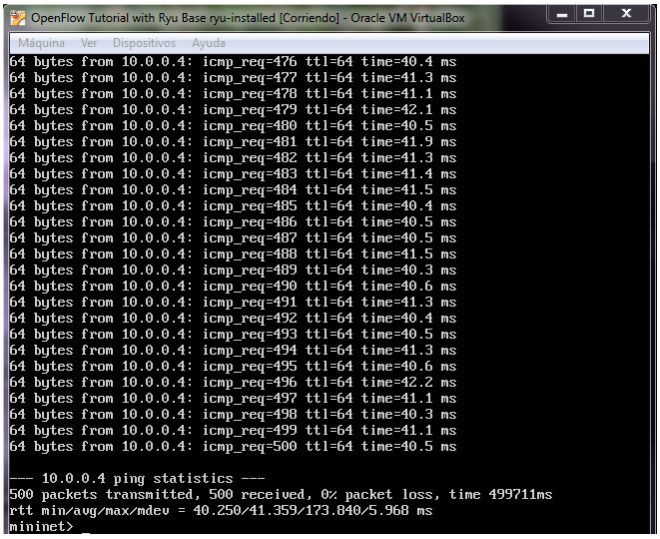
Host 1 – host 2



Host 2 – host 3

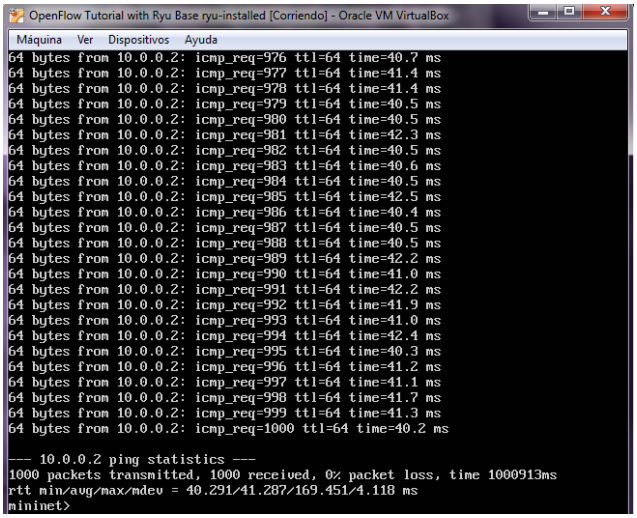


Host 3- host 4

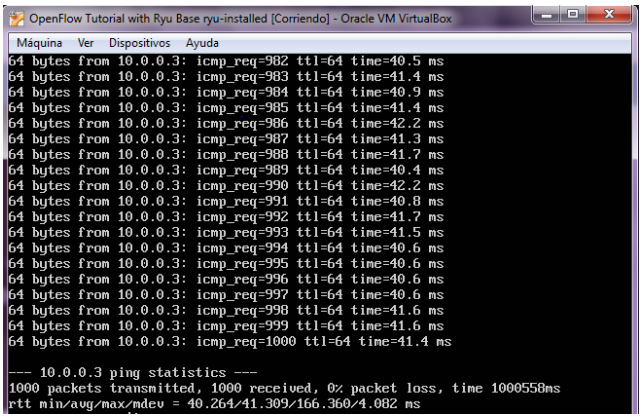


Con 1000 pings

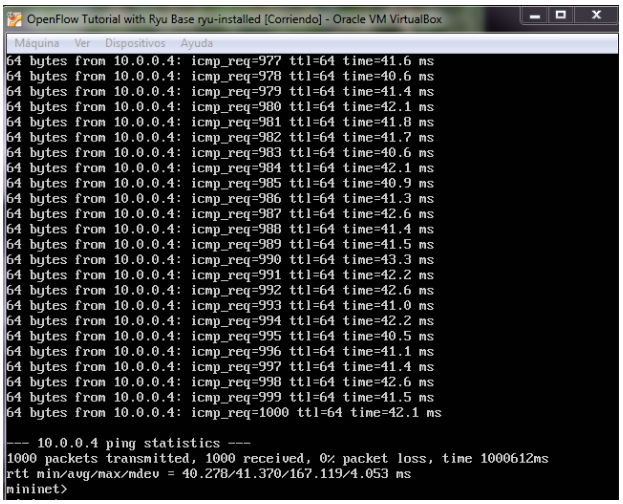
Host 1 – host 2



Host 2 – host 3

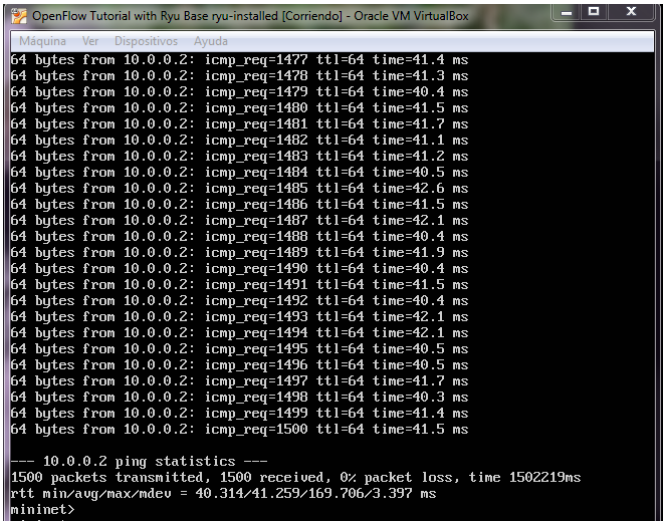


Host 3 – host 4

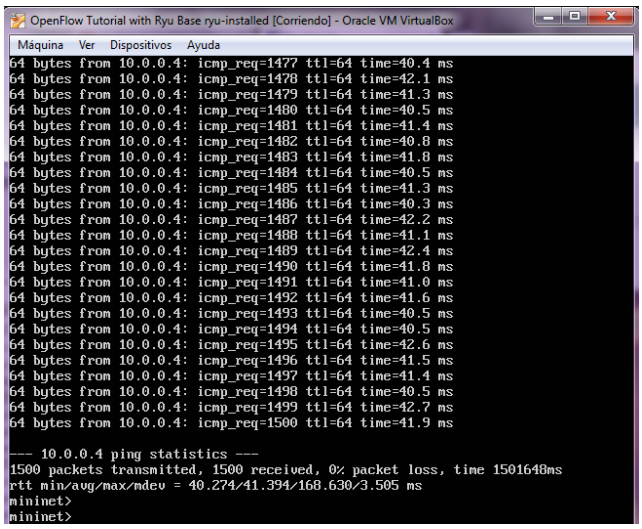


Con 1500 pings

Host 1 – host 2



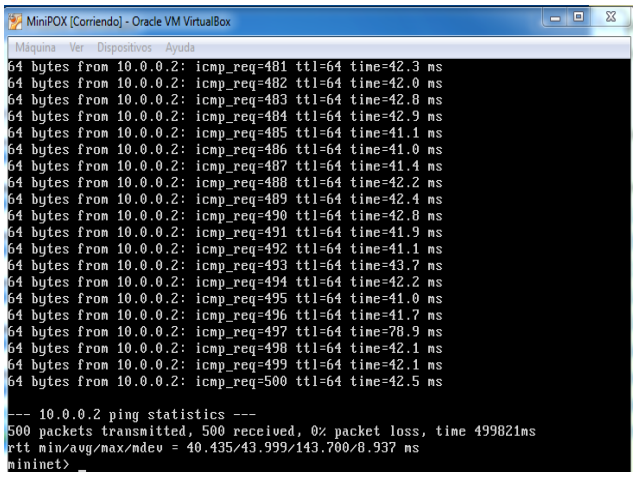
host 3 – host 4



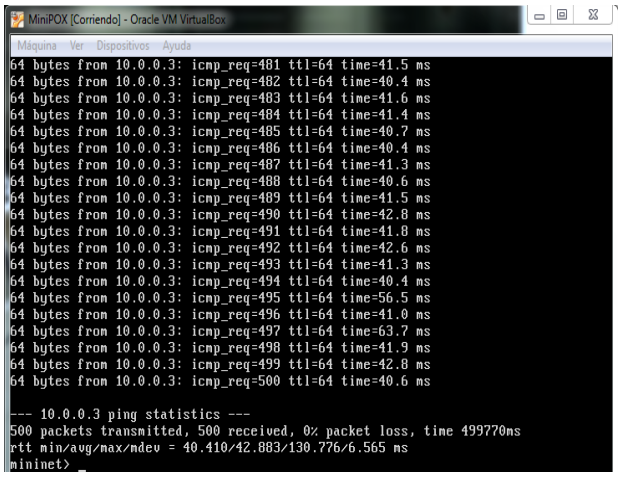
Captura de la latencia obtenida para el controlador POX

Con 500 pings

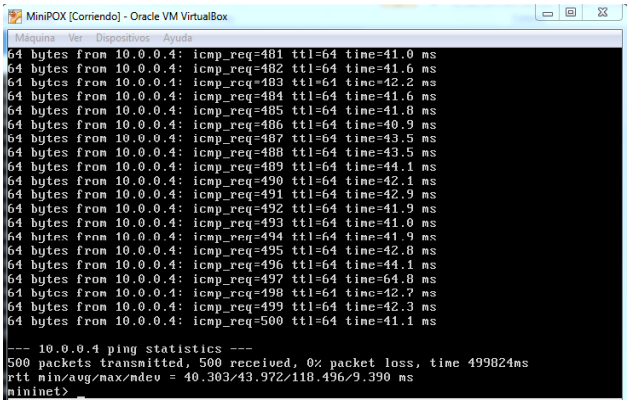
Host 1 – host 2



Host 2 – host 3

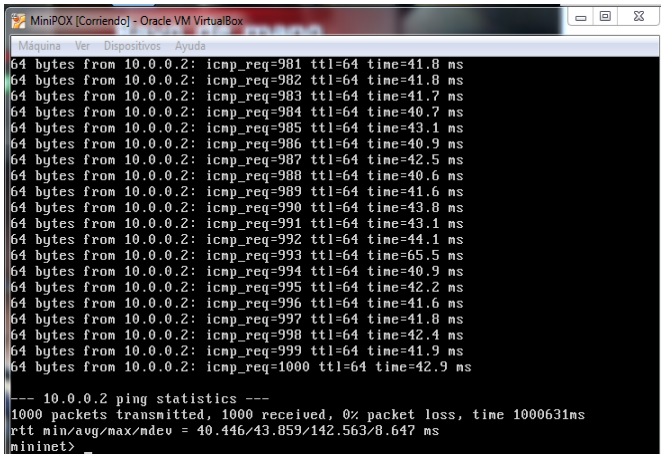


Host 3 – host 4

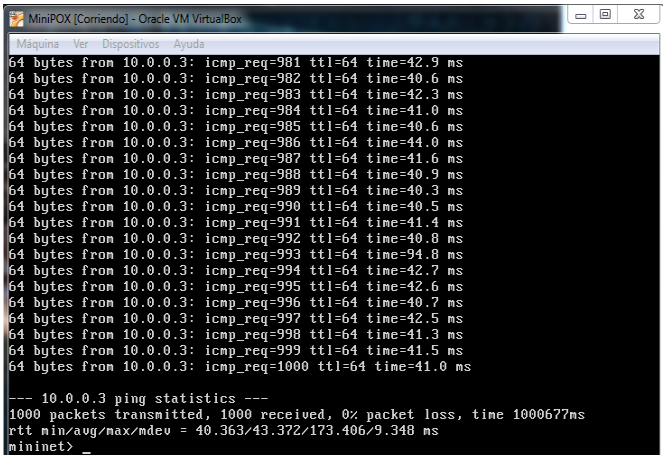


Con 1000 pings

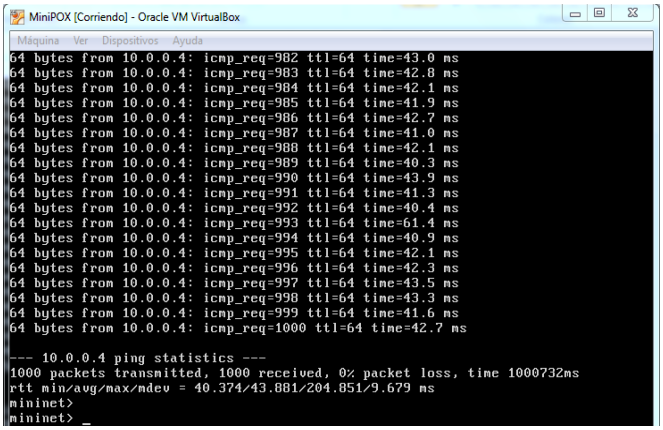
Host 1 – host 2



Host2 – host 3

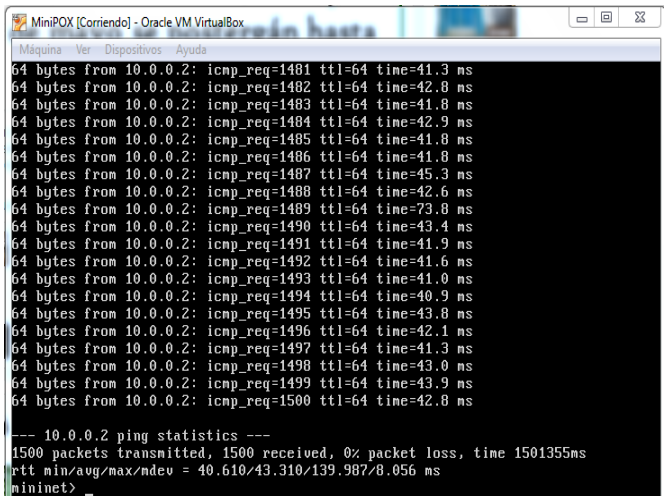


Host 3 – host 4

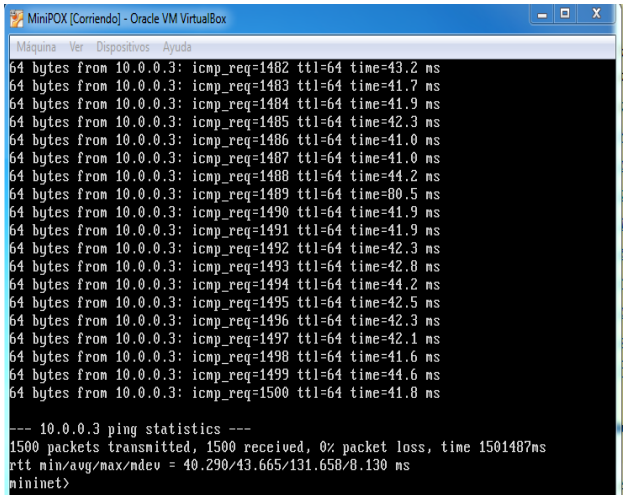


Con 1500 pings

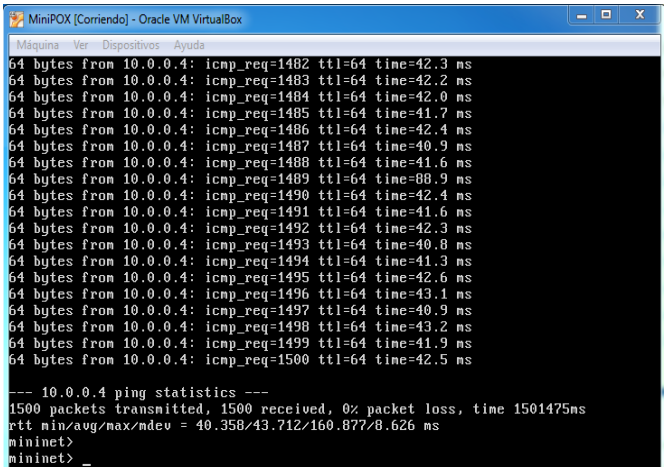
Host 1 – host 2



Host 2 – host 3



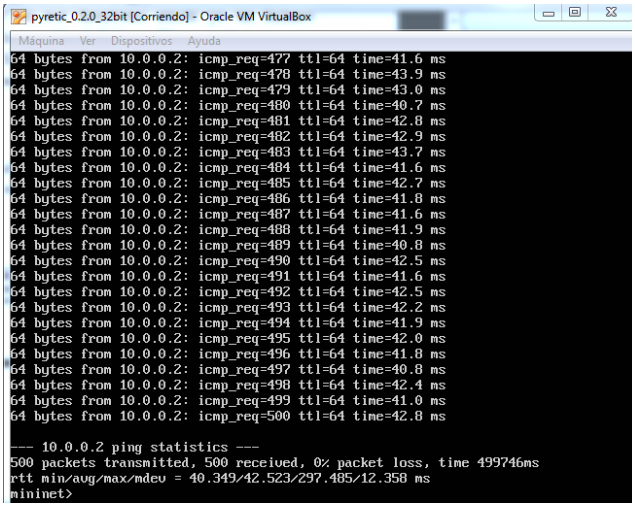
Host 3 – host 4



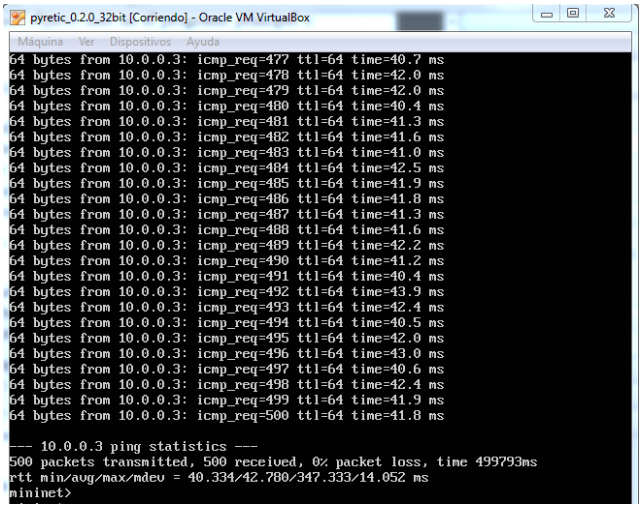
Captura de la latencia obtenida para el controlador PYRETIC

Con 500 pings

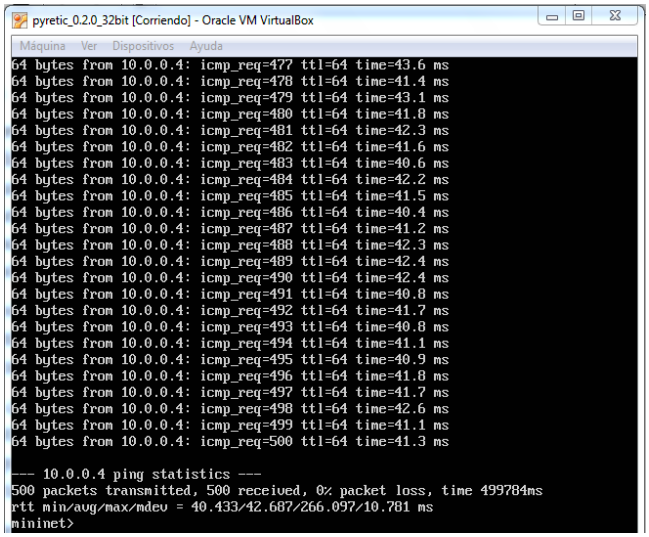
Host 1 – host 2



Host 2 – host 3

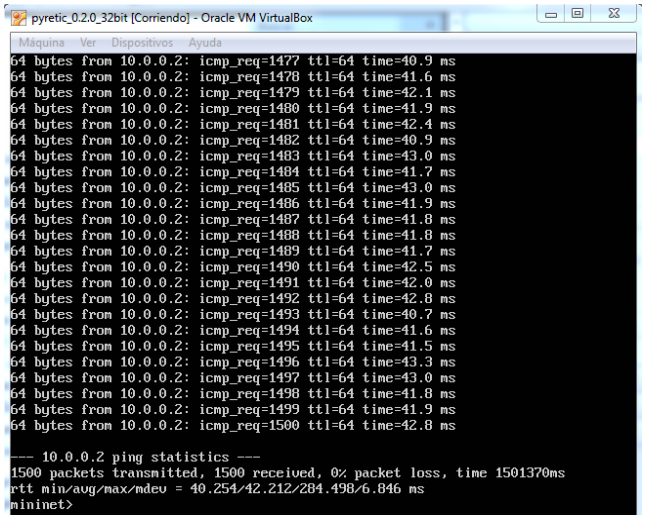


Host 3 – host 4

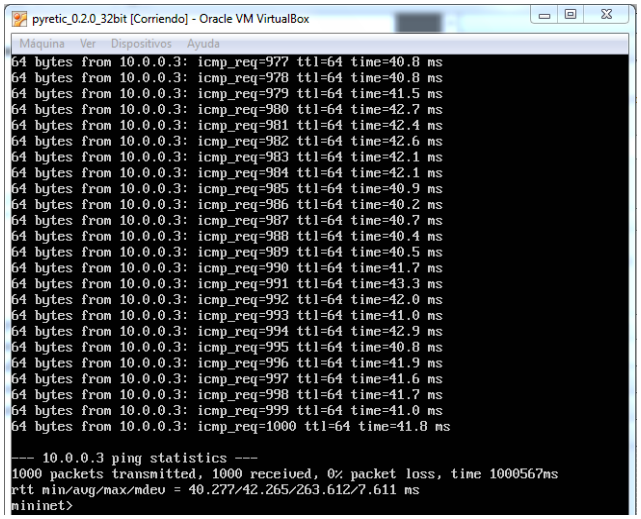


Con 1000 pings

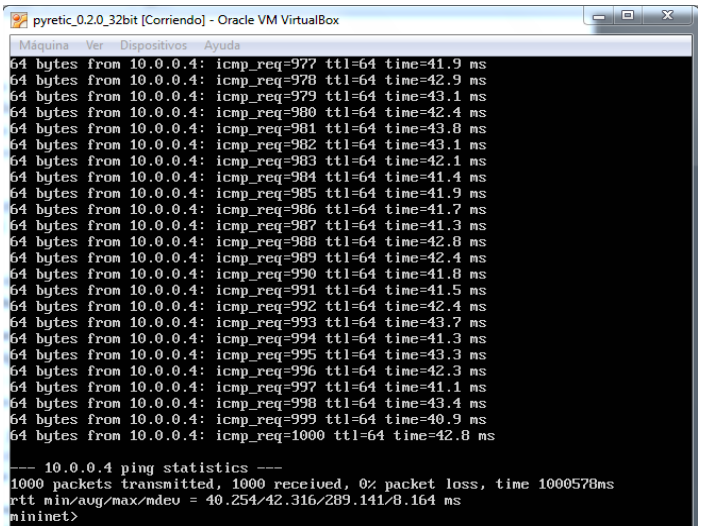
Host 1 – host 2



Host 2 – host 3

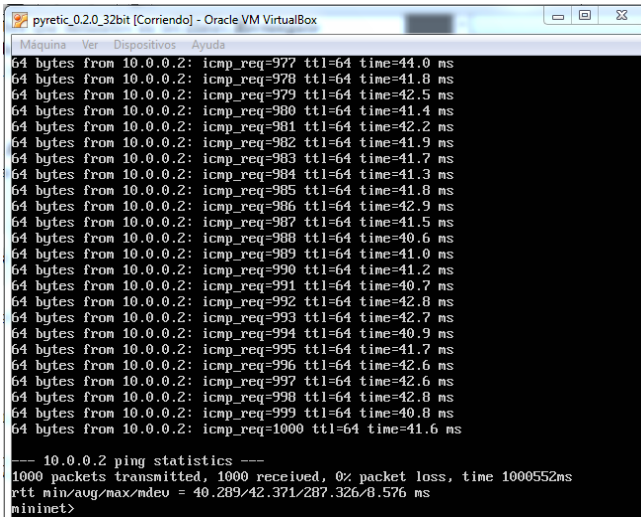


Host 3 – host 4

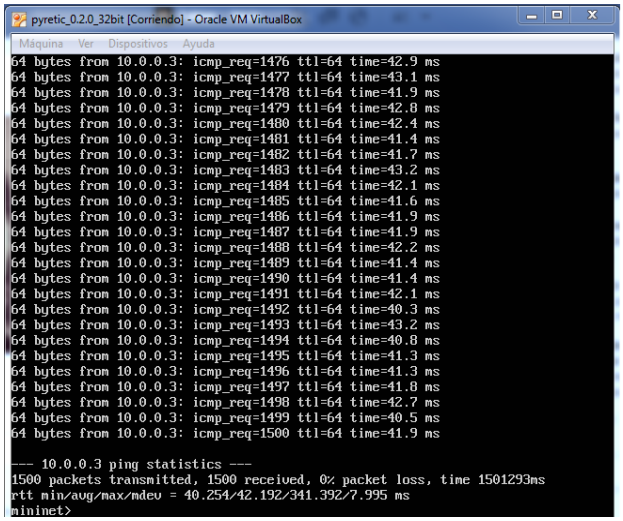


Con 1500 pings

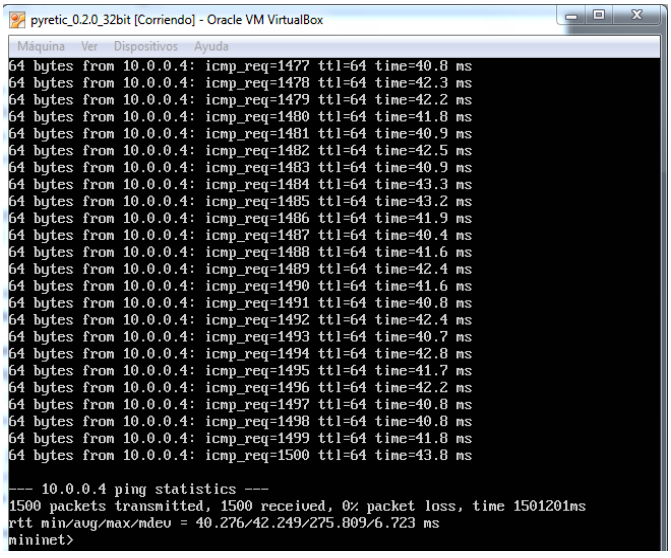
Host 1 – host 2



Host 2 – host 3



Host 3 – host 4



ANEXO N° 3: ACTUALIZACIÓN DE FIRMWARE ORIGINAL (TP - LINK) A FIRMWARE OPENWRT

La actualización de un firmware OpenWRT permite tomar al dispositivo y hacerle capaz de escuchar peticiones Openflow. Los pasos son los siguientes

1. Descargarse desde la página oficial de OpenWRT un firmware (Trunk) para la versión del dispositivo en este caso TP – LINK TL1043 ND: http://downloads.openwrt.org/snapshots/trunk/ar71xx/openwrt-ar71xx-generic-tl-wr1043nd-v2-squashfs-factory.bin

Downloads for TL-WR1043ND v1.x

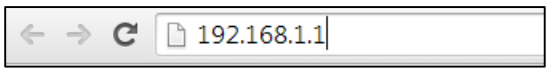
Branch	Type	Download link
Stable (Barrier Breaker)	Factory	http://downloads.openwrt.org/barrier_breaker/14.07/ar71xx/generic/openwrt-ar71xx-generic-tl-wr1043nd-v1-squashfs-factory.bin
	Upgrade	http://downloads.openwrt.org/barrier_breaker/14.07/ar71xx/generic/openwrt-ar71xx-generic-tl-wr1043nd-v1-squashfs-sysupgrade.bin
Trunk (Barrier Breaker)	Factory	http://downloads.openwrt.org/snapshots/trunk/ar71xx/openwrt-ar71xx-generic-tl-wr1043nd-v1-squashfs-factory.bin
	Upgrade	http://downloads.openwrt.org/snapshots/trunk/ar71xx/openwrt-ar71xx-generic-tl-wr1043nd-v1-squashfs-sysupgrade.bin

Downloads for TL-WR1043ND v2.x

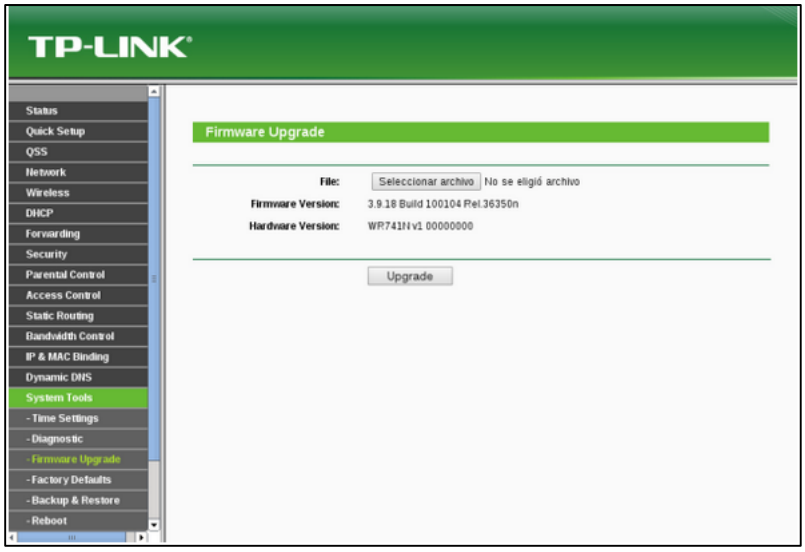
Branch	Type	Download link
Stable (Barrier Breaker)	Factory	http://downloads.openwrt.org/barrier_breaker/14.07/ar71xx/generic/openwrt-ar71xx-generic-tl-wr1043nd-v2-squashfs-factory.bin
	Upgrade	http://downloads.openwrt.org/barrier_breaker/14.07/ar71xx/generic/openwrt-ar71xx-generic-tl-wr1043nd-v2-squashfs-sysupgrade.bin
Trunk (Barrier Breaker)	Factory	http://downloads.openwrt.org/snapshots/trunk/ar71xx/openwrt-ar71xx-generic-tl-wr1043nd-v2-squashfs-factory.bin
	Upgrade	http://downloads.openwrt.org/snapshots/trunk/ar71xx/openwrt-ar71xx-generic-tl-wr1043nd-v2-squashfs-sysupgrade.bin

2. Ingresar a través de la web al router mediante un cable de datos RJ45 para poder manipular el dispositivo actual y actualizar el firmware con el que se descargó en el paso 1.

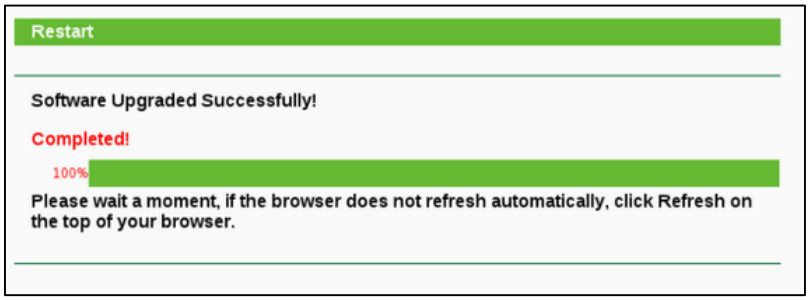
   Para ingresar via web al router se debe ingresar la dirección IP propia del dispositivo, normalmente y por defecto la dirección IP es: 192.168.1.1 o 192.168.0.1.



3. En la pantalla general del dispositivo nos dirigimos a la opción en la parte derecha llamada SYSTEM TOOLS y luego a FIRMWARE UPGRADE



4. Escogemos el archivo del firmware que nos hemos descargado del paso 1 y le damos un clic en “Upgrade”
5. Después de varios minutos aproximadamente el firmware se instalará normalmente y el dispositivo después de reiniciar manualmente constara con el firmware OpenWRT.

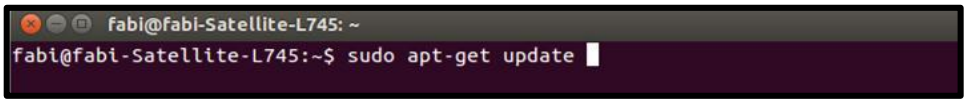


ANEXO N° 4: CREACIÓN DEL FIRMWARE OPENWRT CON OPENFLOW V. 1.0

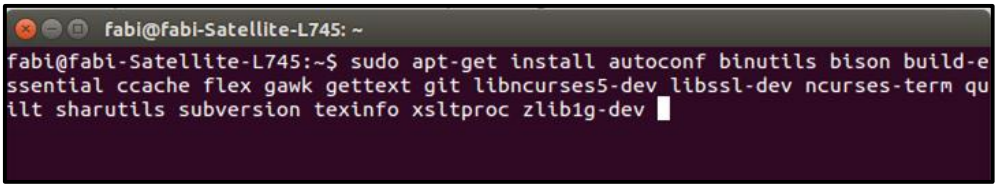
Para la creación de la imagen de extensión .bin que sirve para que escuche peticiones Openflow, se debe trabajar bajo la distribución Linux; para este caso se ha usado Ubuntu 14.04 y como dispositivo un Router-Wireless TP-LINK TL-WR1043ND v2.

1. Instalar las dependencias necesarias:

- sudo apt-get update

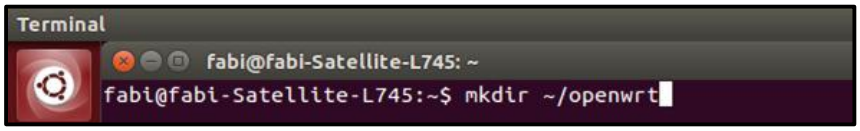


- sudo apt-get install autoconf binutils bison build-essential ccache flex gawk gettext git libncurses5-dev libssl-dev ncurses-term quilt sharutils subversion texinfo xsltproc zlib1g-dev



2. Preparar y crear el directorio donde se colocara la fuente de los archivos OpenWrt

- mkdir ~/ openwrt



- cd ~/openwrt
- svn co snv://svn.openwrt.org/openwrt/trunk

3. Actualizar e instalar los feeds (alimentadores)

- cd ~/openwrt/trunk

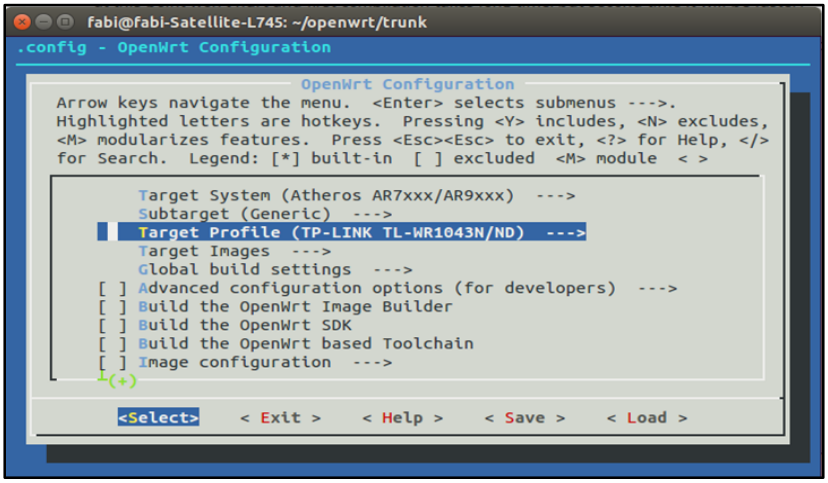
- ./scripts/feeds update -a

- ./scripts/feeds install -a

4. Configurar la creación y verificar los prerrequisitos

- cd ~/openwrt/trunk
- make menuconfig

- Seleccionar “Target System” : Atheros AR71xxx/Ar9xxx
- Seleccionar “Target Profile” : TP-LINK TL-WR1043N/ND
- Salir y guardar

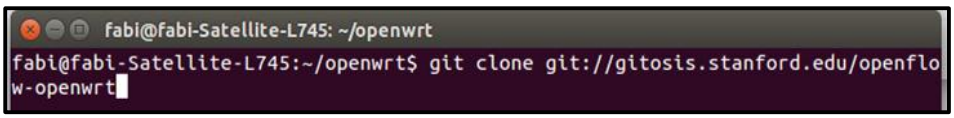


- make prereq

- make (tardará alrededor de una hora por lo general)

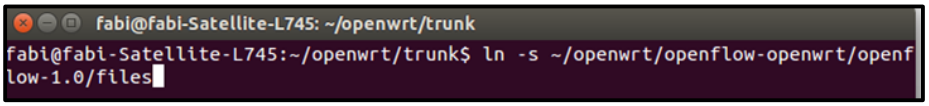
5. Prepara el directorio, descargar la fuente de OpenFlow para OpenWRT y enlazarlo a OpenWRT para la compilación

- cd ~/openwrt/
- git clone git:/gitosis.stanford.org/openflow.openwrt (si es para la versión de OpenFlow 1.0)



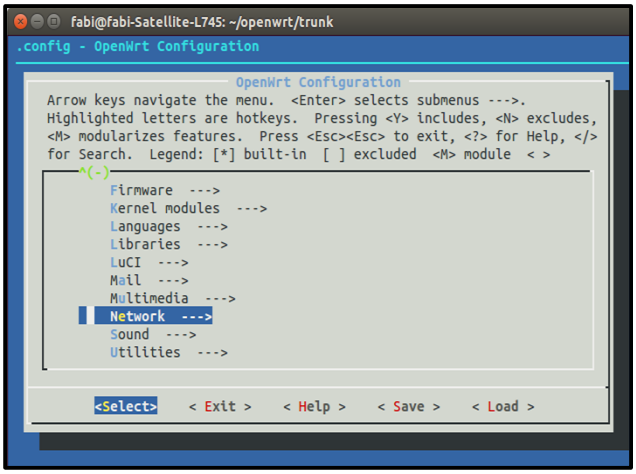
- cd ~/openwrt/trunk/package
- ln -s ~/openwrt/openflow-openwrt/openflow-1.0/

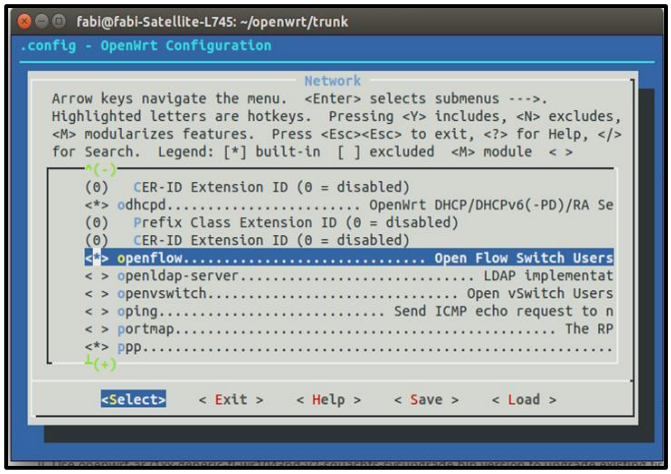
- cd ~/openwrt/trunk
- ln -s ~/openwrt/openflow-openwrt/openflow-1.0/files



6. Configurar y construir con OpenFlow

- cd ~/openflow/trunk
- make menuconfig (Se compilará el menú para la configuración de paquetes a escoger)
- Seleccionar paquete <*> "openflow" bajo la opcion "Network"

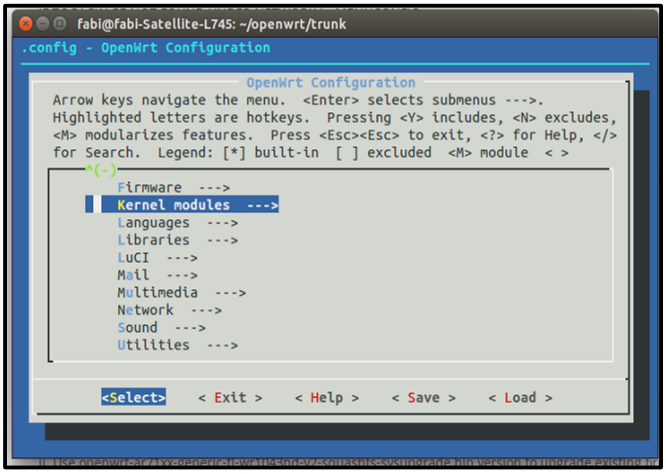


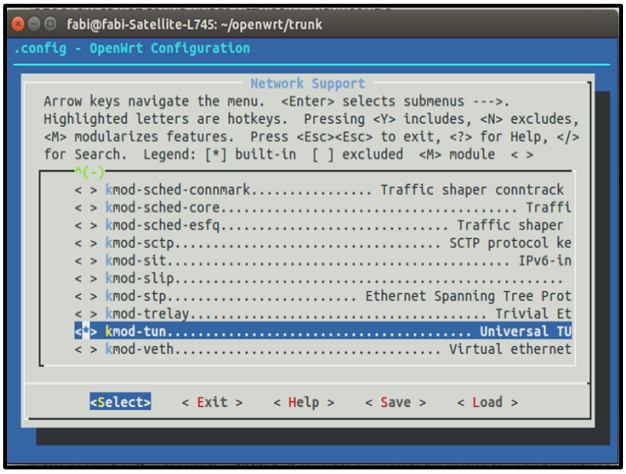


- Seleccionar paquete <*> "tc" bajo la opcion "Network"
- seleccionar <*> "kmod-tun" bajo la opcion "Kernel Modules" y "network support"

Si el paquete "tc" no se encuentra en la lista bajo la opción Network en Menuconfig:

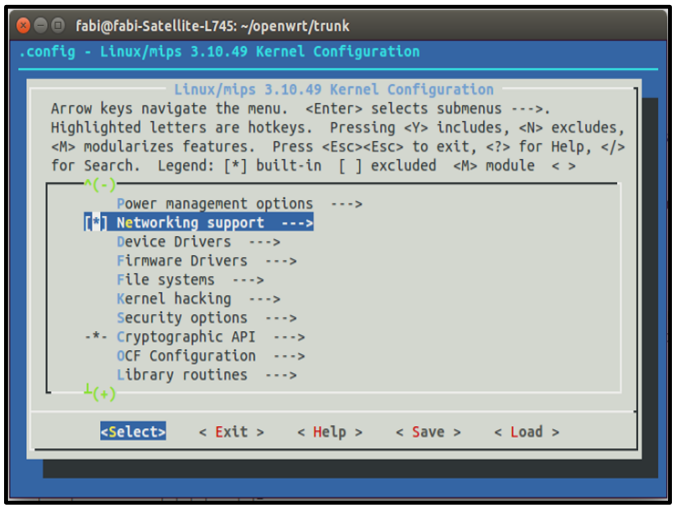
- Seleccionar <*> “kmod-sched-core” y “kmod-sched” bajo “Kernel Modules” y “network support”

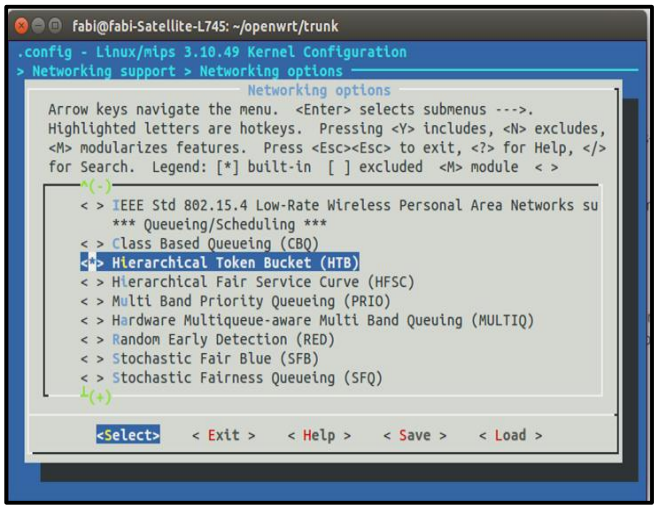




- “kmod-sched-core” es un prerrequisito para TC y puede ser instalado después manualmente.
- Salvar la configuración realizada y salir.
- Make kernel_menuconfig

Seleccionar <*> “Hierarchical Token Bucket (HTB)” bajo “Networking support” y “Networking options”





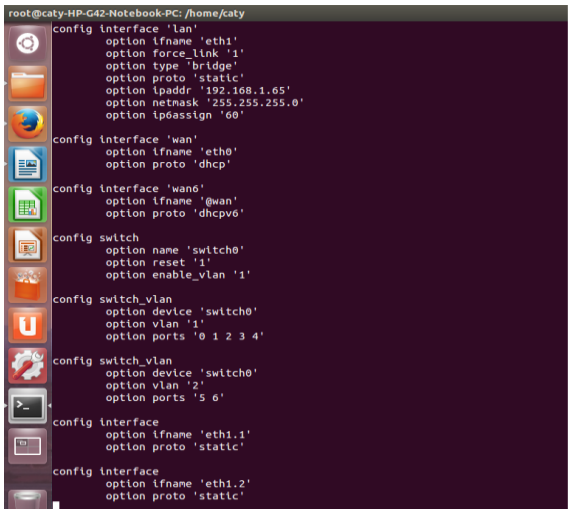
7. Realizar la construcción final

- cd ~/openwrt/trunk
- make

ANEXO N° 5: CONFIGURACIÓN DE LOS ARCHIVOS DEL ROUTER OPENWRT

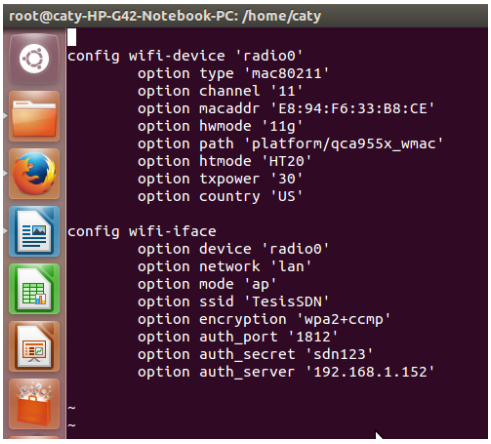
Los archivos a configurar son tres: network, wireless y openflow los mismos que se encuentran bajo el directorio *etc/config* del router.

En el archivo */etc/config/network* se configura la dirección IP del router y se definen las cuatro interfaces físicas del mismo.



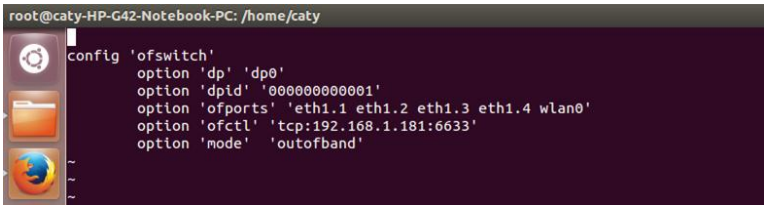
La interfaz eth1 hace referencia a la red LAN por esta razón los nombres de las interfaces del router son: eth1.1, eth1.2, eth1.3, eth1.4, la interfaz eth0 hace referencia a la red WAN.

En el archivo *etc/config/wireless* están todas las características de las configuraciones de la wireless, estas se configuran desde la interfaz gráfica o desde este archivo: nombre de la red, contraseña, tipo de encriptación, y otros datos propios de la red wifi



La configuración del archivo */etc/config/openflow* tiene varios parámetros a considerar:

- **option dp** este parámetro permite signar el nombre del datapath: **dp0**.
- **option dpid** aquí se asigna un identificador al datapath o camino de datos: **000000000001**.
- **option ofports** aquí se asignan los puertos en los cuales se va a maejar el protocolo OpenFlow: **eth1.1, eth1.2, eth1.3, eth1.4, wlan0.**
- **option ofctl** es la dirección donde está el servidor controlador y el puerto por donde este escucha: **tcp:192.168.1.181:6633**
- **option mode** esta opción permite definir si el router y el controlador están en la misma red o no
    - **outofband**: el router y el servidor controlador están en la misma red.
    - **inband**: el router y el servidor controlador se encuentra en una red diferente.

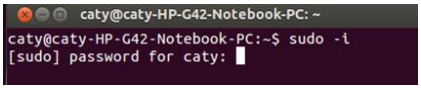


ANEXO N° 6: INSTALACIÓN DE SERVIDOR RADIUS

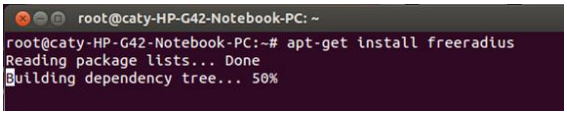
El servidor radius permite la autenticación a una red wireless a través de un cierto número de usuarios previamente registrado.

Su instalación y configuración se muestra a continuación:

1. Es necesario autenticarse como usuario root



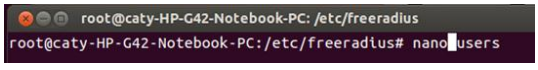
2. A continuación es necesario instalar freeradius para ello se usa el siguiente comando si no existe se instalará sino únicamente se va a actualizar



3. Una vez instalado constatamos la instalación ingresando a la carpeta /etc/freeradius y listando su contenido

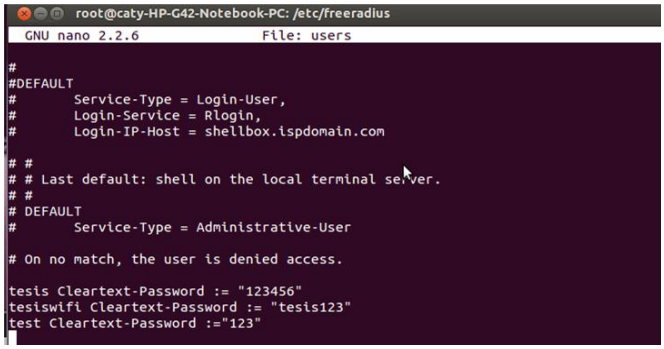
   Dentro de este listado aparecen dos archivos importantes clients.conf y users los mismos que vamos a configurar a continuación para que el servidor funcione correctamente.

4. Ahora es necesario editar el archivo users para eso ingresamos el siguiente comando

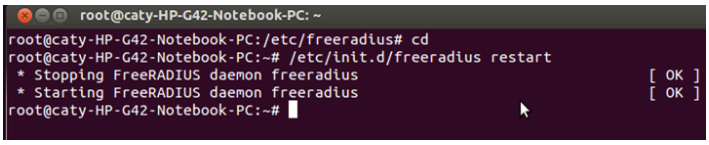


5. Dentro de este archivo se ingresan los usuarios con sus respectivas contraseñas mismas que servirán para autenticarse a la red wireless. En este caso se han

registrado tres usuarios para futuras pruebas

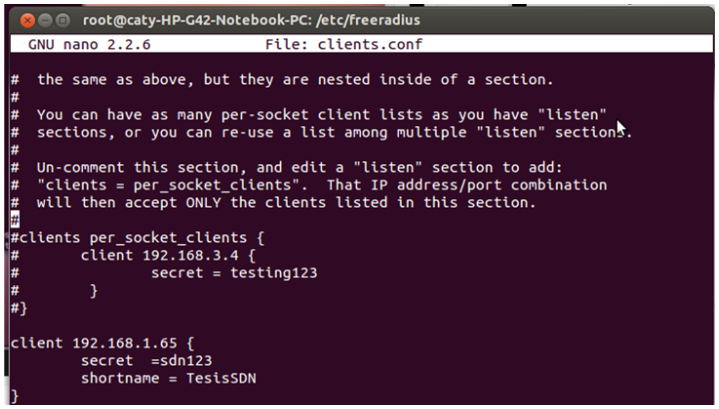


6. A continuación reiniciamos el servicio de radius de la siguiente manera



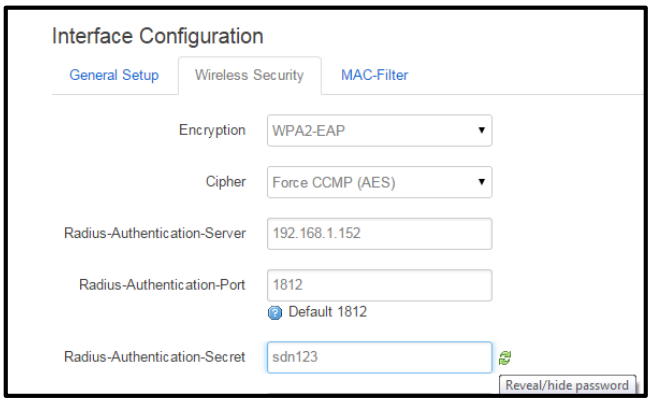
7. Ahora es necesario editar el archivo clients.conf

8. En este archivo es necesario establecer la relación entre el servidor radius y el conmutador, dentro de este archivo se ingresa la dirección del conmutador: 192.168.1.65, la contraseña de acceso al conmutador (secret) y el nombre de la red (shortname)



Después de esta configuración se vuele a reiniciar el servidor radius.

9. A continuación es necesario configurar el conmutador a través de la web; se ubica la dirección del radius: 192.168.1.152 el puerto de comunicación: 1812 y el secret que se registró en el archivo clients.conf

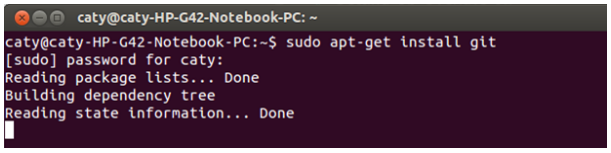


Se guardan los cambios y el servidor radius empieza a funcionar.

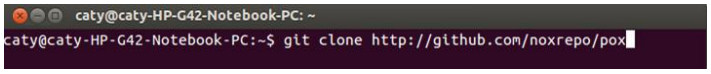
ANEXO N° 7: INSTALACIÓN DE POX

POX es el controlador que posibilita la creación del prototipo planteado para este trabajo de instalación, su instalación es rápida y sencilla y se muestra paso a paso a continuación:

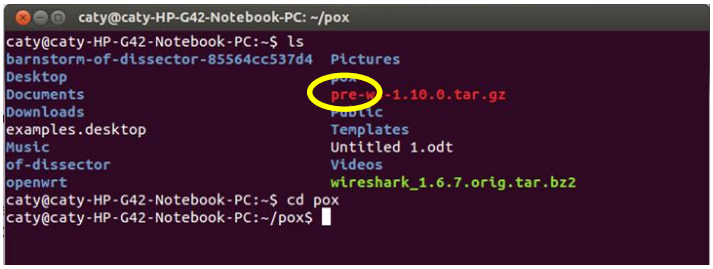
1. Es necesario la instalación de POX a través del comando git razón por la cual es necesario que este comando se pueda usar libremente en el sistema operativo para esto es necesario del siguiente comando; si existe únicamente se actualizará sino se instalará:



2. A continuación es necesario clonar el contenido de POX desde el servidor para esto se usa el siguiente comando:



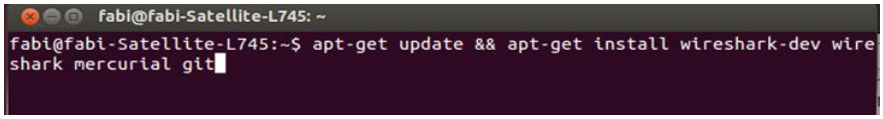
3. Finalmente es necesario constatar la existencia de POX en la PC. Al listar el contenido se puede observar que ya hay un directorio pox
    - cd pox



ANEXO N° 8: INSTALACIÓN DE WIRESHARK Y DPCTL

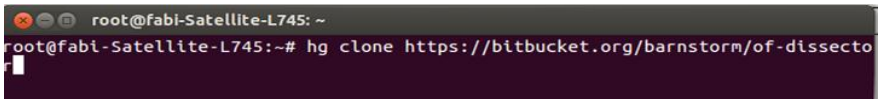
Wireshark y Dpctl son herramientas que pueden ayudar a analizar la situación de una red o solucionar problemas. Para este anexo se ha trabajado con la distribución Linux 12.02 y también se configurara a la herramienta wireshark para que pueda escuchar y manejar peticiones Openflow necesarios para esta investigación. Los pasos son los siguientes:

1. Instalar wireshark desde el repositorio de Linux.
   apt-get update && apt-get install wireshark-dev wireshark mercurial git



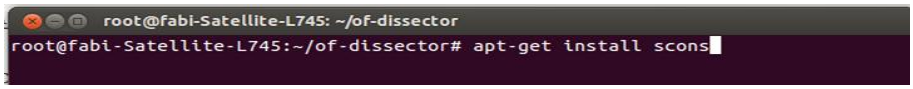
2. Descargar e instalar el paquete Openflow para wireshark

- hg clone https://bitbucket.org/barnstorm/of-dissector

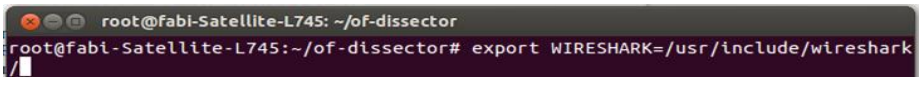


- cd of-dissector/src

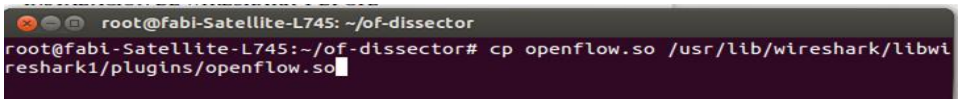
- apt-get install scons



- scons install
- export WIRESHARK=/usr/include/wireshark/



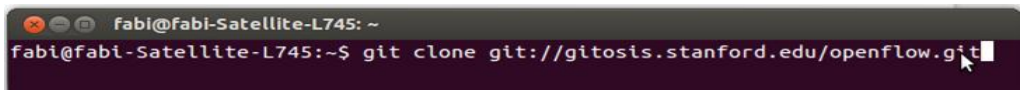
- scons install
- cp openflow.so /usr/lib/wireshark/libwireshark1/plugins/openflow.so



INSTALACION DE DPCTL

Para la instalación de Dpctl se usó de igual manera la distribución Linux 12.02 y a través de los siguientes pasos:

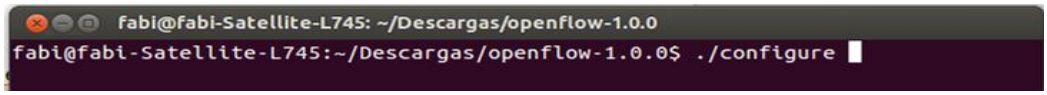
1. Descargar desde la página: git clone git://gitosis.stanford.edu/openflow.git



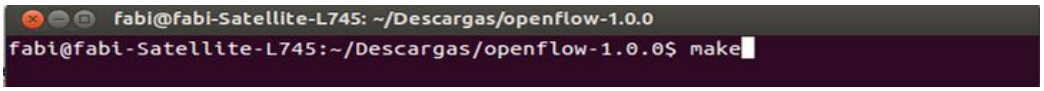
2. c

   d openflow/

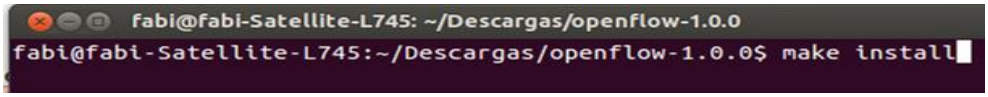
3. ./configure



4. make



5. make install

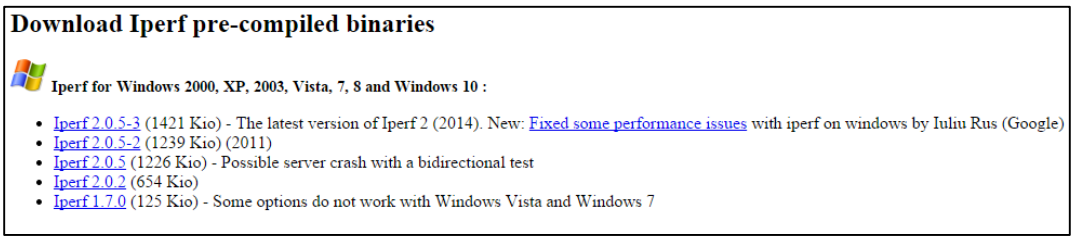


ANEXO N° 9: INSTALACIÓN SOFTWARE IPERF

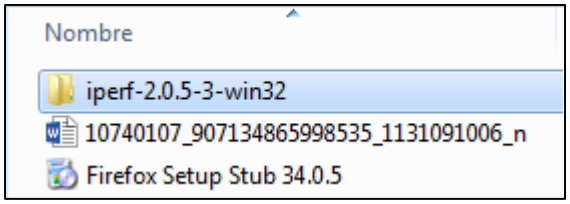
Iperf es un programa que ofrece a los usuarios valores de rendimiento que sucede en una red. La instalación de este software se lo realiza con los siguientes pasos. Cabe recalcar que las pruebas realizadas para esta investigación se la realizó desde un equipo con Windows 7.

1. Descargar el paquete de la pagina https://iperf.fr

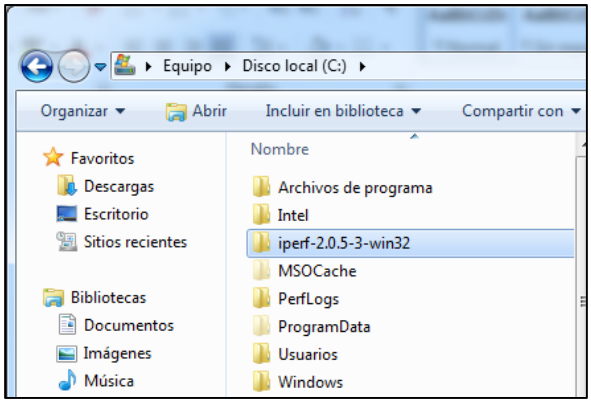
2. Escoger la última versión de Iperf para Windows: Iperf 2.0.5-3



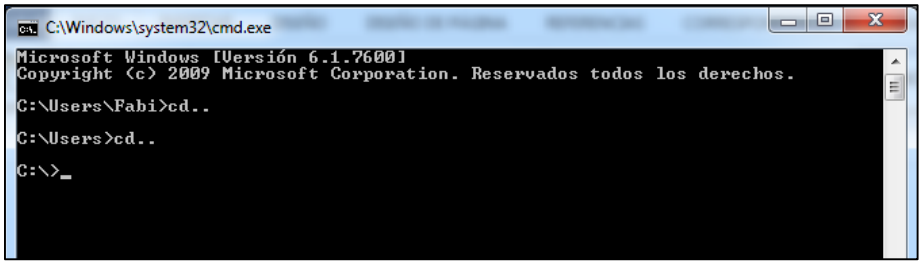
3. Descomprimir el archivo .rar o .zip en una carpeta



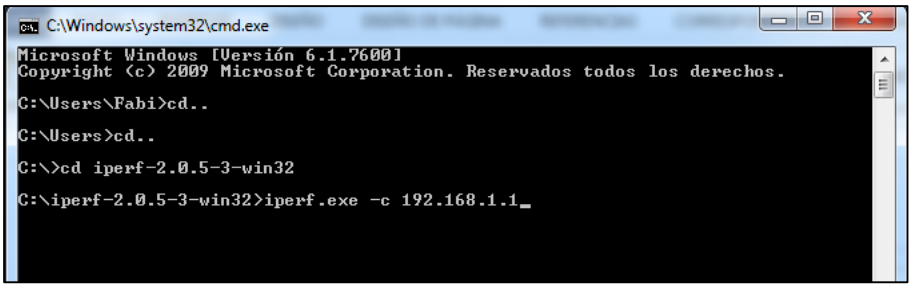
4. Colocar el directorio descomprimido en el Disco de instalación de Windows en este caso C:



5. Abrir una terminal CMD para la ejecución de Iperf y dirigirse a la raíz del sistema de archivos en este caso C:\>



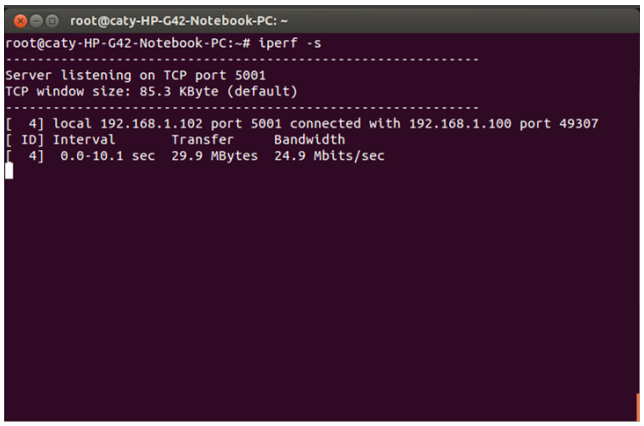
6. Ingresar a la carpeta descomprimida y ejecutar el software iperf con el comando iperf.exe



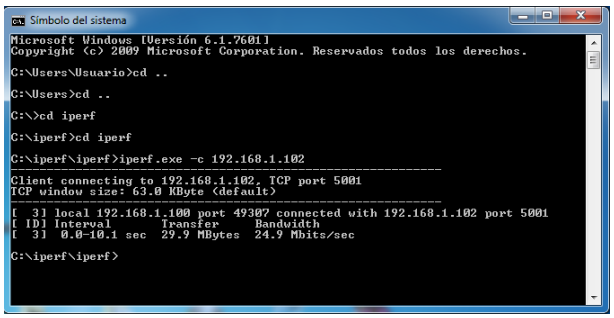
ANEXO N° 10 (IPERF GENERAL): DEL PROTOTIPO SDN Y PROTOTIPO ESPOCH

IPERF GENERAL CON EL PROTOTIPO ESPOCH

SERVIDOR

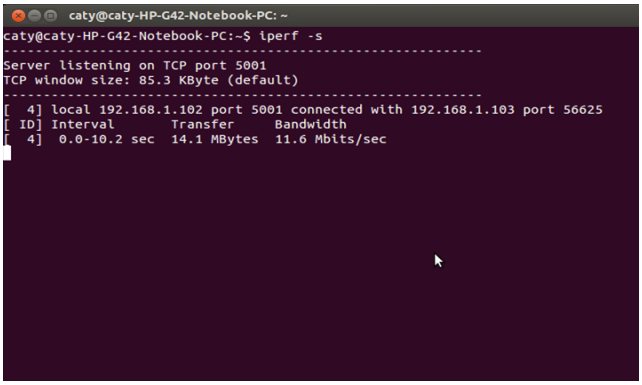


CLIENTE

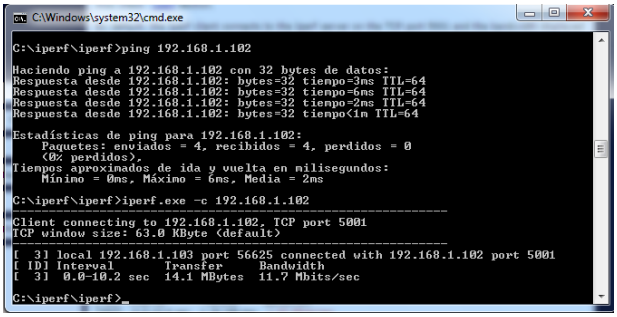


IPERF GENERAL CON EL PROTOTIPO SDN

SERVIDOR



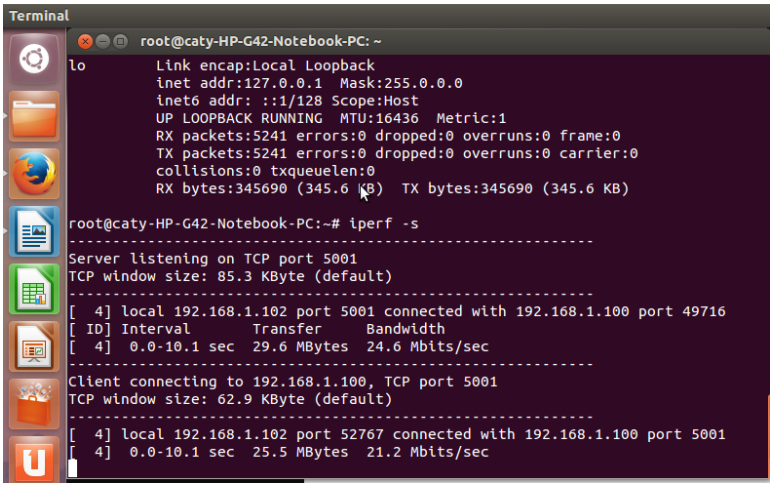
CLIENTE



ANEXO N° 10: (IPERF BIDIRECCIONAL): DEL PROTOTIPO SDN Y PROTOTIPO ESPOCH

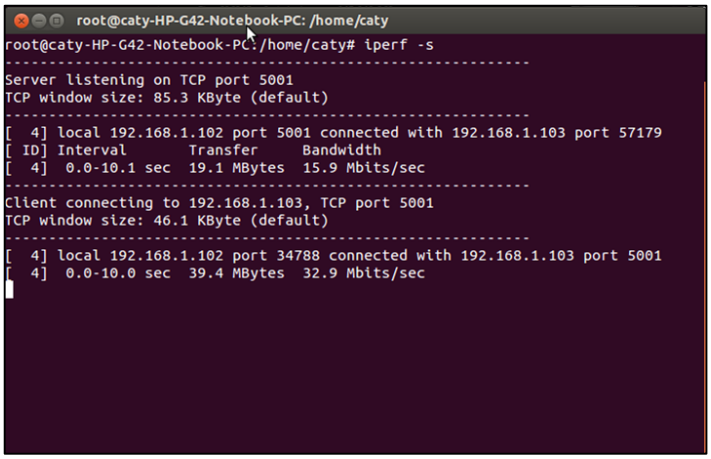
IPERF BIDIRECCIONAL SECUENCIAL PROTOTIPO ESPOCH

SERVIDOR



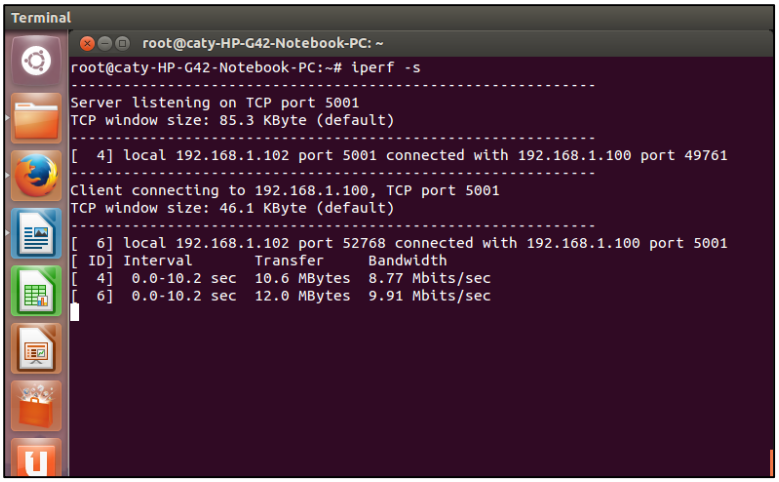
IPERF BIDIRECCIONAL SECUENCIAL PROTOTIPO SDN

SERVIDOR



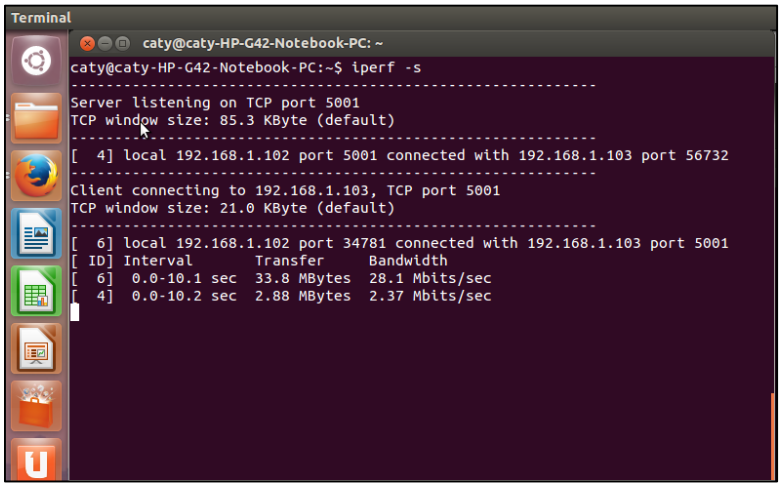
IPERF BIDIRECCIONAL SIMULTÁNEO PROTOTIPO ESPOCH

SERVIDOR



IPERF BIDIRECCIONAL SIMULTÁNEO PROTOTIPO SDN

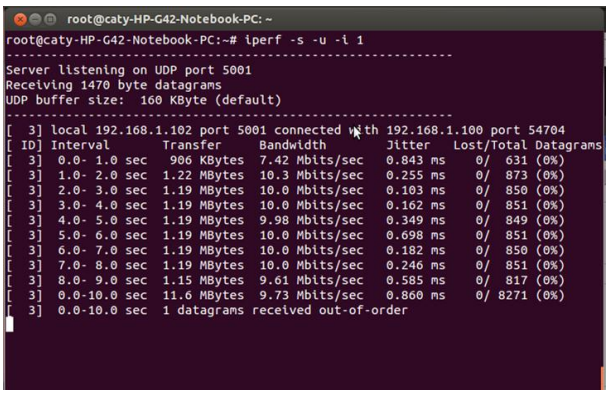
SERVIDOR



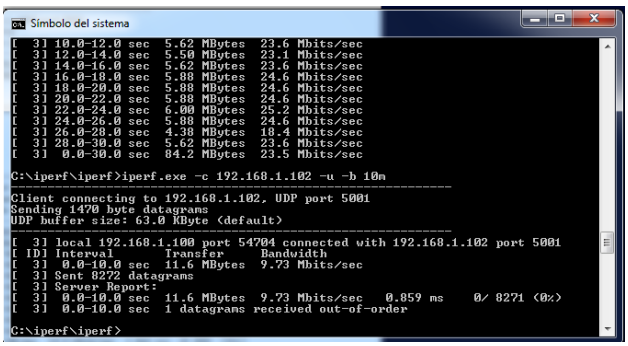
ANEXO N° 10: (JITTER): DEL PROTOTIPO ESPOCH Y PROTOTIPO SDN

JITTER DEL PROTOTIPO ESPOCH

SERVIDOR

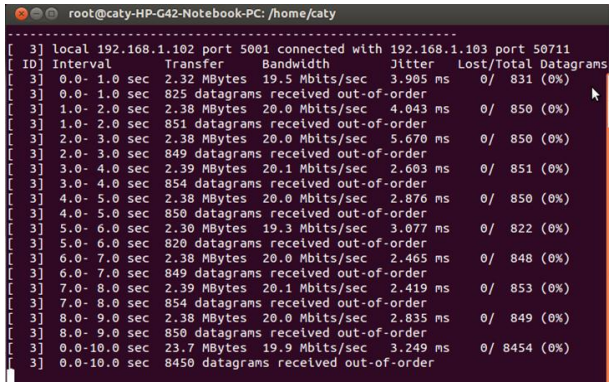


CLIENTE

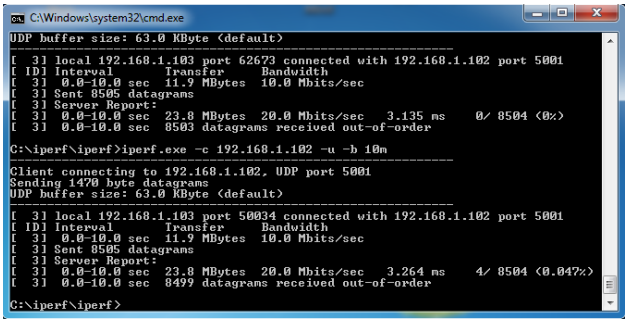


JITTER DEL PROTOTIPO SDN

SERVIDOR



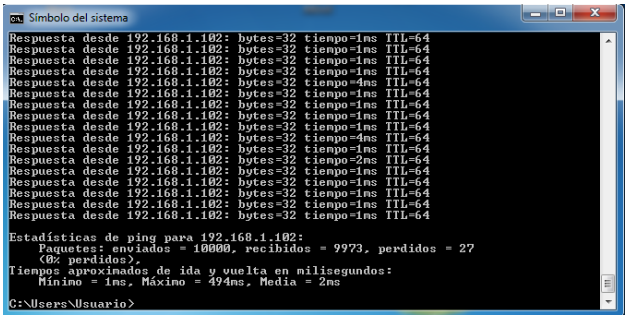
CLIENTE



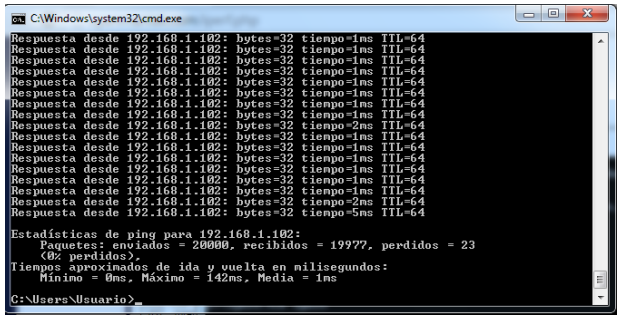
ANEXO N° 10 (LATENCIA): DEL PROTOTIPO ESPOCH Y PROTOTIPO SDN

PROTOTIPO ESPOCH

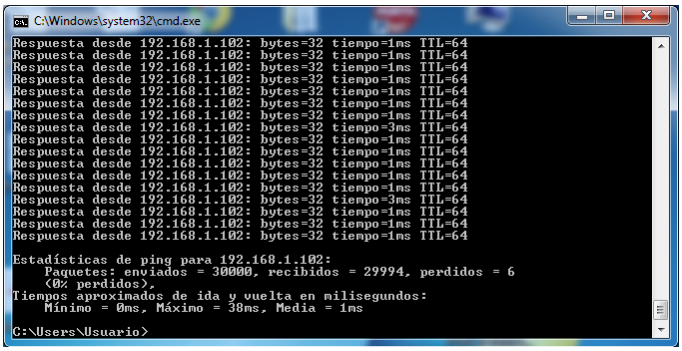
10000 PING



20000 PING

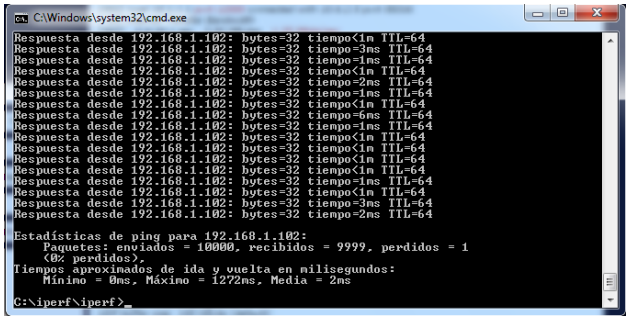


30000 PING

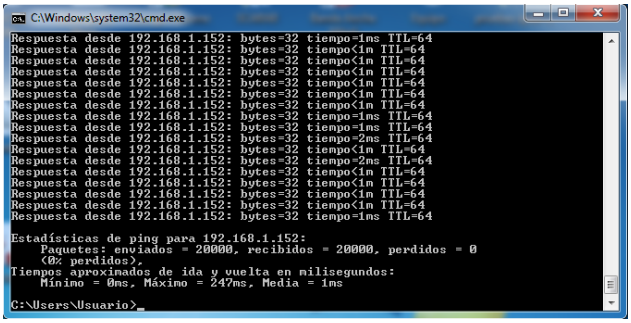


PROTOTIPO SDN

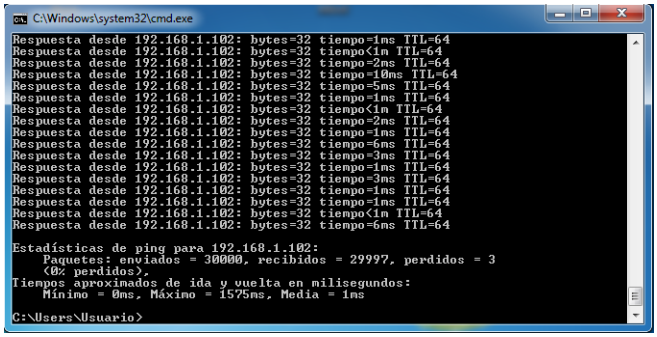
10000 PING



20000 PING



30000 PING


C:\Windows\system32\cmd.exe
Respuesta desde 192.168.1.102: bytes=32 tiempo=1ms TTL=64
Respuesta desde 192.168.1.102: bytes=32 tiempo<1m TTL=64
Respuesta desde 192.168.1.102: bytes=32 tiempo=2ms TTL=64
Respuesta desde 192.168.1.102: bytes=32 tiempo=10ms TTL=64
Respuesta desde 192.168.1.102: bytes=32 tiempo=5ms TTL=64
Respuesta desde 192.168.1.102: bytes=32 tiempo=1ms TTL=64
Respuesta desde 192.168.1.102: bytes=32 tiempo<1m TTL=64
Respuesta desde 192.168.1.102: bytes=32 tiempo=2ms TTL=64
Respuesta desde 192.168.1.102: bytes=32 tiempo=1ms TTL=64
Respuesta desde 192.168.1.102: bytes=32 tiempo=6ms TTL=64
Respuesta desde 192.168.1.102: bytes=32 tiempo=3ms TTL=64
Respuesta desde 192.168.1.102: bytes=32 tiempo=1ms TTL=64
Respuesta desde 192.168.1.102: bytes=32 tiempo=3ms TTL=64
Respuesta desde 192.168.1.102: bytes=32 tiempo=1ms TTL=64
Respuesta desde 192.168.1.102: bytes=32 tiempo=1ms TTL=64
Respuesta desde 192.168.1.102: bytes=32 tiempo<1m TTL=64
Respuesta desde 192.168.1.102: bytes=32 tiempo=6ms TTL=64
Estadísticas de ping para 192.168.1.102:
Paquetes: enviados = 30000, recibidos = 29997, perdidos = 3
(0% perdidos),
Tiempos aproximados de ida y vuelta en milisegundos:
Mínimo = 0ms, Máximo = 1575ms, Media = 1ms
C:\Users\Usuario>