Trabajo Final de Grado

Facultad de Ingeniería
Ingeniería en Telecomunicaciones

Tema:

Diagramación de una Red Definida por Software de bajo costo utilizando mini ordenadores de la marca Raspberry Pi

Gastón Alejandro Borja Perazzi

Tutor: Ingeniero Juan Galleguillo

Año 2016

Dedicatoria

A mi mama Alejandra por ser siempre la que me empujo de atrás para poder seguir avanzando de a poco y superando obstáculos tanto en la facultad como en la vida.

A mi papa Néstor por dar todo por mí, apostar en mi futuro y regalarme la posibilidad de poder enfrentar mis estudios sin otra preocupación más que la de estudiar.

Gastón Alejandro Borja Perazzi

Agradecimientos

A mis viejos por bancarme todos estos años.

A los amigos y compañeros que me regaló la vida y con los que siempre voy a estar en deuda.

A profesores y profesionales del Departamento de Ing. Electrónica y Telecomunicaciones del Instituto Universitario Aeronáutico por tantos consejos brindados, dudas aclaradas, sonrisa mañaneras y la confianza depositada en mí.

Gastón Alejandro Borja Perazzi

"Nuestro mayor miedo, no es que no encajemos... Nuestro mayor miedo es que tenemos una fuerza desmesurada, es nuestra luz y no nuestra oscuridad lo que más nos asusta, empequeñecerse no ayuda al mundo, no hay nada inteligente en encogerse para que otros no se sientan inseguros a tu alrededor, todos deberíamos brillar como hacen los niños, no es cosa de unos pocos, sino de todos, y al dejar brillar nuestra propia luz, inconscientemente damos permiso a otros para hacer lo mismo, al liberarnos de nuestro propio miedo, nuestra presencia automáticamente libera a otros..."

Marianne Williamson
"A Return to Love: Reflections on the Principles of A Course in Miracles"
(1992)

Glosario

- **ACL:** Access Control List. (Lista de control de acceso)
- **AH:** Authentication Header. (Cabecera de autenticación)
- **API:** Application Programming Interface. (Interfaz de aplicación)
- **ARP:** Address Resolution Protocol. (Protocolo de resolución de dirección)
- **BGP:** Border Gateway Protocol. (Protocolo de puerta de enlace fronteriza)
- **Framework**: Infraestructura digital. Conjunto estandarizado de conceptos prácticas y criterios para enfocar una problemática.
- **FTP:** File Transfer Protocol. (Protocolo de transferencia de archivo)
- **GPL:** General Public License.
- **GUI:** Grafical User Interfaz (Interfaz Gráfico de usuario)
- **HTTP:** Hypertext Transfer Protocol. (Protocolo de transferencia de hipertexto)
- **HTTPS:** Hypertext Tranfer Protocol Secure. (Protocolo de transferencia de hipertexto seguro)
- **ICMP:** Internet Control Message Protocol. (Protocolo de mensaje de control de internet)
- **IP:** Internet Protocol. (Protocolo de internet)
- **IS-IS:** Intermedia System to Intermedia System.
- **ISP:** Internet Service Provider. (Proveedor de internet)
- **LAN:** Local Area Network. (Red de area local)
- **LLDP:** Link Layer Discovery Protocol. (Protocolo de descubrimiento de capa de enlace)
- **MAC:** Media Access Control.
- **NetConf:** Network Configuration Protocol. (Protocolo de configuración de red)
- **NFV:** Network Function Virtualization. (Virtualización de funciones de red)
- **NIB:** Network Information Base.
- **OF:** OpenFlow.
- **OFA:** OpenFlow Application.
- **OFC:** OpenFlow Controller.
- **ONF:** La Open Network Foundation (ONF), una organización guiada por usuarios con el objetivo de fomentar el desarrollo de la SDN.
- **OOB:** Out of Band.

- **OSPF:** Open Shortest Path First.
- **OVS:** Open vSwitch.
- **Pipeline:** Cadenas de procesos conectados.
- **QoS:** Quality of Service. (Calidad de servicio)
- **SDN:** Redes definidas por software o SDN (del inglés Software Defined Network)
- **SNMP:** Simple Network Management Protocol.
- **SO:** Sistema Operativo.
- **SSH:** Secure Shell.
- **STP:** Spanning Tree Protocol.
- **TCP:** Transmission Control Protocol. (Protocolo de control de transmisión)
- **Threads:** Tareas.
- **TLS:** Transport Layer Security. (Seguridad en capa de transporte)
- **TTL:** Time to Live. (Tiempo de vida)
- **VLAN:** Virtual Local Area Network. (Red de area local virtual)
- **VM:** Virtual Machine. (Máquina Virtual)
- **VoIP:** Voice over IP.
- **WAN:** Wide Area Network.

Tabla de ilustraciones

Índice

1. Introducción

Las redes de comunicación actuales enfrentan problemas que se van volviendo más críticos con el paso de los años, y hace tiempo se está buscando soluciones que se adapten tanto a las necesidades de los clientes como a las de las grandes empresas.

Dentro de estos problemas podemos encontrar la gran cantidad de datos que se necesitan enviar por la red, la necesidad de seguridad avanzada, mayor eficiencia en el procesado de paquetes para no generar cuellos de botellas en los enrutadores ni colisiones, propiedades de escalabilidad acordes al crecimiento de la población y más incógnitas para las cuales las redes actuales resultan obsoletas.

Todas estas exigencias que son un desafío para casi todos los dispositivos de red actuales en aspectos como la capacidad, el cambio de patrones de tráfico que circula por la red, la calidad de servicio y el advenimiento de aplicaciones de alto rendimiento basadas en virtualización, es decir, el despliegue acelerado de las redes de alto rendimiento nos muestran las limitaciones de los dispositivos de conectividad actuales, obligándonos a buscar una estructura de control más predecible y flexible.

Ya hace varios años que las grandes empresas de comunicación están trabajando en el desarrollo de redes definidas por software o SDN, que sean altamente programables y, por medio de aplicaciones, permitan la automatización de la red y su flexibilidad ante las necesidades referentes a seguridad, calidad de servicio (QoS) y enrutamiento, que sean tan programables y manejables como el administrador lo considere necesario.

El switch virtual está tomando el papel del corazón de las redes de grandes centros de datos, redes universitarias, etc. Dado que la virtualización es una de las herramientas más usadas por las empresas para hacer un uso eficiente de los recursos, lo que les permite obtener respuestas económicas y completas a los problemas de networking.

La *Open Networking Foundation* (ONF)[1], es una de las principales fundaciones encargadas del desarrollo de las Redes Definidas por Software, buscando que la industria del

[1] Open Networking Foundation, Disponible en: https://www.opennetworking.org

networking desarrolle redes que provean funcionalidades menos costosas y más sencillas de manejar, mediante la separación del plano de control del plano de datos que es la principal característica de esta arquitectura de red.

A pesar de ser una solución a muchos problemas que poseen las redes actuales, presenta una serie de desventajas a la hora de entrar al mercado, una de las más importantes es el precio de los dispositivos que conforman esta red.

Los switch "OpenFlow" o switch de nueva generación tienen un costo elevado y no siempre su uso hace rentable este gasto. Si consideramos una red grande, en donde el gasto del producto se amortice rápidamente, y se utilice el switch en la totalidad de su rendimiento, resulta ser una inversión muy buena. Pero si se desea usar esta tecnología en redes pequeñas, en donde no usaremos todas las capacidades que nos brindan estos switches, la relación costo-beneficio se vuelve en nuestra contra y resulta poco rentable el gasto en estos dispositivos.

Debido a que las redes se están volviendo la parte crítica de la infraestructura de las empresas, hogares e instituciones educativas se plantea este Trabajo Final de Grado titulado "***Diagramación de una Red Definida por Software de bajo costo utilizando mini ordenadores de la marca Raspberry Pi***" donde se hace uso de hardware libre y de costo reducido combinándolo con las nuevas técnicas de virtualización para crear un switch de nueva generación de "bajo costo", que pueda cumplir las funciones de los dispositivos de altos precios y marcas reconocidas, con menor rendimiento pero lo suficientemente alto para cubrir todas las necesidades de las redes pequeñas.

2. Redes Definidas por Software

2.1. Características

Esta nueva arquitectura de red presenta una característica clara que es la separación de las funciones de control y gestión de las funciones de "forwarding", dando como resultado una red dinámica, manejable, centralizada, adaptable y rentable, ideal para las necesidades de las empresas y clientes de hoy en día.

Al entender este concepto de la separación del plano de control y el plano de datos, podemos imaginar a la red como un gran conjunto de dispositivos de conmutación (switch, routers, etc.), conectados entre sí por líneas de transmisión y, separado de estos, la capa de control por encima compuesta por un controlador que es el "cerebro" de la red y toma las decisiones sobre los paquetes que circulan por ella.

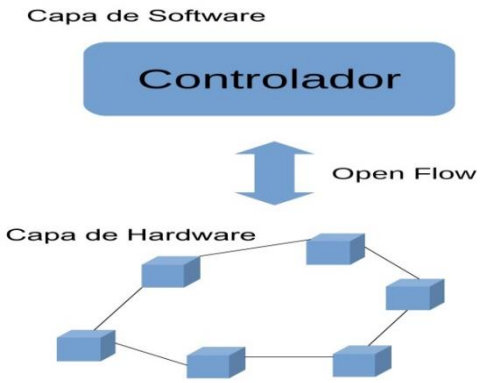

Ilustración 1: Visión básica SDN [Fuente: Propia]

Dentro de las características que presenta esta nueva arquitectura, y que sacan ventaja a las redes convencionales, se encuentran:

- **Gestión centralizada:** Permite gestionar la totalidad de la red desde un único punto debido a que toda la inteligencia de la red se encuentra concentrada en el controlador. Este tiene una visión global de la red y de todos los elementos que se encuentran en ella. Si se desea realizar algún cambio en la red, solo con informárselo al controlador ya tendremos el problema resuelto. Nos ofrece soluciones con respecto a problemas que

tienen las redes tradicionales, eliminando fallas de seguridad, cuellos de botellas, errores de configuración, etc. que traen consigo los múltiples puntos de decisión.

- **Agilidad:** Se puede reservar ancho de banda para servicios especiales o QoS (Calidad de Servicio) con mayor rapidez y adaptarse a los cambios y prioridades de la red. La ventaja de esto es que, programando el controlador con anterioridad, podemos ir cambiando en "caliente" estas características y que la red vaya mutando a medida que van entrando paquetes nuevos con diferentes prioridades a la hora de ser transportados.

- **Bajos costos:** Se considera esto al pensar en redes grandes que al no tener que optar por la homogeneidad de productos y tener un proveedor genérico, abaratan los costos de la red. Muchos de los controladores son open-source y presentan las mismas o más características que los controladores propietarios, por ende, no se debe pagar licencias para poder tener control sobre la red.

- **Programación automática:** Software Defined Networks permite a los administradores, por medio de programas automatizados que ellos mismos pueden escribir ya que no depende de software propietario, gestionar y optimizar los recursos de la red de forma rápida y dinámica. Esto se debe a que se pueden ligar balanceadores de carga, firewalls o cualquier tipo de aplicación para mantener control sobre la red a los controladores por medio de API's. Estos programas analizan constantemente la red y van informando al controlador sobre el estado de la misma y los requerimientos de cada paquete o enlace para que este mantenga un rendimiento constante y libere al administrador de la red de esta tarea.

- **Fácilmente programable**: Brinda la posibilidad de programar o configurar el plano de control con mayor facilidad al estar separado del plano de red. La mayoría de los controladores responden a lenguajes de programación muy usados como Python o Java, lo que hace que configurarlos y manejarlos resulte muy cómodo, y al estar separado del plano de datos, se pueden realizar cambios sin cortar el funcionamiento de la red.

- **Escalabilidad:** Con esta arquitectura, es muy fácil proyectar una red que crezca en gran medida sin perder ninguna de sus características. Con solo conectar el nuevo switch al controlador, este se encargara de su configuración y de su puesta a punto para que realice sus funciones y monitoree su rendimiento. Un solo controlador puede administrar muchos switches al mismo tiempo y se pueden usar más de un controlador en la red para tener redundancia por si uno llega a fallar.

- **Basado en estándares abiertos y proveedores neutrales:** Como se implementa a través de estándares abiertos, simplifica el diseño de la red y las operaciones porque las instrucciones son proporcionadas por el controlador, en lugar de múltiples dispositivos, proveedores específicos o protocolos. Ya no se depende de los protocolos o estándares a los que uno estaba restringido cuando se adquiere un switch Cisco, HP, Juniper, etc. Ahora uno es capaz de crear sus propios protocolos y decidir el futuro de los paquetes de la red de acuerdo a las necesidades actuales.

- **Seguridad:** Al estar la red centralizada por el controlador, no se corre el riesgo de poseer agujeros de seguridad en las configuraciones de los switches. Solo hay un punto de acceso a la configuración de la red y es el controlador.

Todas estas características que poseen las SDN y no las redes tradicionales nos llevan a pensar que en un futuro, no muy lejano, esta arquitectura será ampliamente aceptada por todos los sectores del mercado de telecomunicaciones, favoreciendo tanto a los clientes que buscan velocidad, seguridad, y robustez en sus conexiones, como a las empresas que buscan brindar un servicio de alta calidad y confiabilidad con el menor costo de operación y de inversión posible.

2.2. Arquitectura lógica de las SDN

2.2.1. Introducción

En las redes de datos tradicionales, los equipos de red cuentan con dos elementos principales, uno es el plano de control y el otro es el plano de datos. El primero es el encargado de tomar las decisiones en el enrutamiento, control y seguridad de los paquetes, el segundo cumple la función de transmitir los datos de usuario. En el plano de datos los encargados de la tarea de enrutamiento son los protocolos como BGP, OSPF, IS-IS, etc. que han ido

evolucionando con el tiempo para poder cumplir con las exigencias que proponen las evoluciones en los datos trasmitidos por la red, pero cada vez se exigen más y más tareas a este plano como movilidad, QoS, entre otros y no siempre cumplen con las expectativas.

Software Defined Networks aparece como una solución a esas exigencias que los protocolos actuales no pueden satisfacer y esto lo logra usando software para el plano de control.

Esta arquitectura de red se divide en tres partes, la más baja es la capa de datos o hardware, la capa del medio es la capa de control y la capa superior es la capa de aplicación.

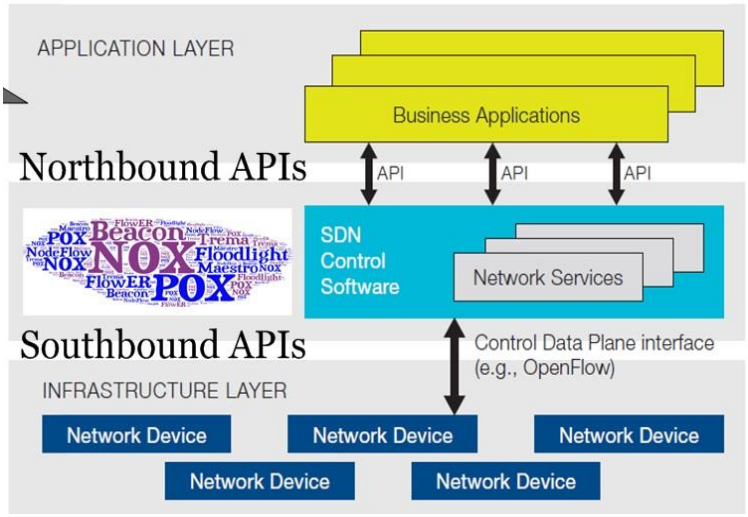

Ilustración 2: Arquitectura lógica SDN [Fuente: www.docplayer.es]

2.2.2. Capa de Hardware

Esta capa está compuesta por el conjunto de switches, routers, hosts, medios de transmisión, etc. que pueda contener el sistema. Aquí es donde se los paquetes se encaminan y viajan a los distintos puntos de la red. Los elementos principales de esta capa son los switches OpenFlow, capaces de comunicarse con la capa de control por medio de Southbound APIs y así obtener la información necesaria para poder encaminar los paquetes que circulan por la red. Estos switches reciben constante actualización en sus tablas de flujos por parte del controlador y utilizan esta información para "matchear" los paquetes entrantes y decidir la acción a realizar con él. Si el switch no sabe qué hacer con el paquete, esto quiere decir que no encuentra coincidencias en su tabla de flujo, lo envía, por medio de la southbound API, hacia el controlador, donde será analizado y se devolverá al switch junto con una nueva entrada de flujo

para que la próxima vez que llegue un nuevo paquete con estas características el dispositivo de switcheo sepa qué hacer con él.

2.2.3. Capa de Control

Esta capa contiene la inteligencia de la red, aquí se realizan las tomas de decisiones sobre qué hacer con los paquetes que circulan por la red, calidad de servicio, seguridad, etc.

Para poder comunicarse con el switch, el administrador de red necesita programar uno o más controladores para que concentre todas las decisiones que se deberán tomar para que la red sea eficiente, y satisfaga las necesidades del usuario. Estos controladores son el cerebro de la red, son ellos quienes concentran la gestión, seguridad, control, etc. de la topología. Pueden estar instalados en un servidor, una computadora personal o en cualquier dispositivo que tenga las características necesarias para que funcione eficazmente. Hay muchos controladores de donde elegir, algunos son de código abierto y otros son propietarios.

2.2.3.1. Controladores de código abierto

Ilustración 3: Controlador NOX

- **NOX [2]:** Es uno de los primeros controladores creados para redes OpenFlow, fue escrito por la empresa Nicira, y desde su lanzamiento bajo la licencia GPL fue uno de los más usados para desarrollos e investigación en este campo. Se trata de una plataforma para la creación de aplicaciones de control de red y algunas de sus características son:

  - Proporciona soporte y una API para OpenFlow 1.0. (No funciona con OpenFlow 1.3)
  - Las reglas para el tráfico son escritas en C++.
  - Está dirigido a las recientes distribuciones de Linux

[2] NOX, Disponible en: https://github.com/noxrepo/nox

- Incluye componentes que le permiten realizar algunas acciones como: descubrimiento de la topología, aprendizaje de conmutación y el análisis de redes extendidas de conmutación.
- NOX proporciona una API de alto nivel para OpenFlow, así como a otras funciones de control de red.

A pesar de sus características, este controlador no es muy usado en la actualidad, su mayor defecto es que no es compatible con versiones de OpenFlow mayores a 1.0 y su bajo rendimiento en comparación con controladores más nuevos.

Ilustración 4: Controlador POX

- **POX [3]:** Desarrollado a partir de NOX, este controlador OpenFlow es uno de los más usados en la actualidad, siendo uno de los más fáciles de entender, muy completo en sus componentes y la posibilidad de poder programar el control de la red en el lenguaje de programación Python. A pesar de que no posee soporte para versiones mayores a OpenFlow 1.0 (Igual que NOX), es muy usado para realizar desarrollos e investigaciones en el campo de las SDN y es usado como punto de partida para muchos otros controladores que salieron luego. Algunas de sus características son:

  - Interfaz OpenFlow basada en Python.
  - Los componentes muestran la selección de ruta de acceso, detección de la topología, entre otras.
  - Está dirigido específicamente a Linux, Mac OS y Windows.
  - Interfaz gráfica y herramientas de visualización similares a las de NOX.

[3] POX, Disponible en: https://github.com/noxrepo/pox

Ilustración 5: Controlador Floodlight

- **FloodLight [[4]]:** Este controlador es capaz de trabajar tanto con switch virtuales como físicos, que sean capaces de trabajar con el protocolo OpenFlow. Basado en código abierto, bajo la licencia Apache y escrito en JAVA provee una forma de poder modificar el comportamiento de la red. Posee un módulo principal que es el encargado de escuchar los paquetes OpenFlow y realizar la distribución de eventos y módulos secundarios que agregan funciones extras a las del módulo principal como detección de hosts, determinación de la topología, etc.

  Algunas de las características de este controlador son:

  - Puede trabajar tanto con switches virtuales como con switch físicos, siempre y cuando soporten el protocolo OpenFlow.
  - No se necesitan grandes requerimientos para hacerlo funcionar.
  - Está desarrollado por una gran comunidad abierta de desarrolladores que participan activamente para reparar errores, realizar investigaciones y buscar mejoras.
  - Está diseñado para soportar grandes rendimientos.
  - Soporta versiones mayores a OpenFlow 1.0
  - Curva de aprendizaje baja comparado con otros controladores.

[4] Project Floodlight, Disponible en: http://www.projectfloodlight.org/floodlight/

Ilustración 6: Controlador Beacon

- **Beacon [[5]]:** Es un controlador OpenFlow rápido, multiplataforma, modular, basado en Java, compatible con la programación basada en threads y eventos. Algunas de las características más relevantes son:

  - **Estable:** Beacon ha sido utilizado en muchos proyectos de investigación y algunas implementaciones de prueba. Actualmente es capaz de soportar 100 switches virtuales y 20 switches físicos en redes de datos experimentales.
  - **Multiplataforma:** Está escrito en Java y se ejecuta en muchas plataformas, desde servidores Linux hasta teléfonos Android.
  - **Código abierto:** Esta licenciado bajo una combinación de la licencia GPL11v.2 y la Licencia de la Universidad de Stanford.
  - **Dinámico:** Los paquetes de código en Beacon pueden ser inicializados, detenidos, actualizados e instalados en tiempo de ejecución, sin interrumpir otros paquetes.
  - **De rápido desarrollo:** Ya que es fácil de descargar y correr. Java y Eclipse simplifican el desarrollo y la depuración de sus aplicaciones.
  - **Rápido:** Debido a su característica de multiproceso.
  - **Interfaz gráfica de usuario:** Opcionalmente incorpora al servidor web Jetty Enterprise y un framework extensible con una interfaz de usuario personalizada.
  - **Amplia documentación:** La documentación de Beacon incluye tutoriales y guías para ayudar a conseguir un funcionamiento productivo de inmediato, además de un foro de usuarios activos.

[5] Beacon, Disponible en: https://openflow.stanford.edu/display/Beacon/Home

Ilustración 7: Controlador ODL

- **OpenDayLight [6]:** Este controlador es un proyecto de código abierto que tiene su sede en la Fundación Linux pero es fruto de la colaboración entre varias empresas, entre ellas Cisco, Dell e Intel. La meta de este proyecto es acelerar la adopción de las redes definidas por software y crear una fundación sólida para la Virtualización de Funciones de Red (NFV). Este software también está escrito en Java. Sus características son:

  - Escrito en Java y multiplataforma.
  - Soporta distintas southbound APIs como OpenFlow (En todas sus versiones), BGP-LS, SNMP, etc.
  - Capaz de adaptarse a una variedad muy grandes de topologías.
  - Es el controlador más ampliamente documentado.
  - Continuamente actualizado y con soporte permanente.
  - Cuenta con interfaz gráfica de usuario web con posibilidad de realizar cambios en los switches muy fácilmente.

Ilustración 8: Controlador Trema

- **Trema [7]:** Es un software completo y fácil de usar para el desarrollo de controladores OpenFlow que permite definir las reglas de control mediante los lenguajes de programación Ruby y C.

  Posee bibliotecas y módulos que funcionan como interfaz con los switches. Una gran característica es que posee un emulador de red OpenFlow y no es necesario disponer de

[6] OpenDayLight, Disponible en: https://www.opendaylight.org/
[7] Trema, Disponible en: https://trema.github.io/trema/

switches físicos o virtuales para poder probar las aplicaciones del controlador. Algunas características de este controlador son:

- Permite ser ejecutado en una red virtualizada, es decir, permite ser desarrollado en una computadora simple, con el uso de máquinas virtuales y sin necesidad de switches físicos.
- Permite especificar y construir topologías virtuales arbitrarias.
- El controlador desarrollado en una red virtualizada puede ser perfectamente implementado en una red real.
- Soporta GNU/Linux y ha sido probado en los siguientes entornos:
  - Ruby 1.8.7 (1.9.x no está soportada al momento).
  - Ubuntu (de 10.04 en adelante).
  - Debian GNU / Linux 6.0 (i386/amd64).

Ilustración 9: Controlador Ryu

- **Ryu [8]:** Es un controlador basado en componentes para SDN que nos ofrece un framework amplio y fácil de utilizar. Provee componentes de software con API's bien definidas que le facilitan la tarea a los desarrolladores de crear nuevas aplicaciones de gestión y control de red. Soporta varios protocolos para la gestión de dispositivos de red como OF, NetConf, OF-Config, etc. Con respecto a OpenFlow, Ryu soporta completamente las versiones 1.0, 1.1, 1.2, 1.3, 1.4 y 1.5 y las extensiones de Nicira. Todo el código es gratuito y disponible bajo la licencia Apache 2.0. Como dato adicional, "Ryu" en japonés significa "flujo".

En esta tabla podemos comparar algunas de las características que presentan los controladores nombrados.

[8] Ryu, Disponible en: https://osrg.github.io/ryu/

Controlador	FloodLight	POX	Beacon	NOX	Trema	Ryu	ODL
Facilidad de instalación	FÁCIL	FÁCIL	REGULAR	FÁCIL	FÁCIL	FÁCIL	REGULAR
Simplicidad en el manejo de funcionalidades	FÁCIL	FÁCIL	REGULAR	FÁCIL	DIFÍCIL	REGULAR	DIFÍCIL
Facilidad de expansión	FÁCIL	FÁCIL	REGULAR	DIFÍCIL	FÁCIL	FÁCIL	REGULAR
Soporte de los desarrolladores	BUENO	BUENO	REGULAR	MALO	MALO	BUENO	BUENO
Módulos para el desarrollo de aplicaciones	BUENO	BUENO	REGULAR	MALO	REGULAR	BUENO	BUENO
Bien documentado	SÍ	SÍ	SÍ	NO	SÍ	SÍ	SÍ
Manejo de interfaz web	SÍ	NO	SÍ	NO	NO	SÍ	SÍ

Tabla 1: Comparación de controladores Open-Source [Fuente: Propia]

2.2.3.2. *Controladores propietarios*

Estos son controladores diseñados por empresas para ser usados principalmente con switches OpenFlow de la misma marca, pero también pueden ser usados con dispositivos de otros fabricantes, suelen poseer algunas características que los controladores Open-source no poseen y son más amigables con el usuario, teniendo un interfaz de usuario más intuitivo.

Ilustración 10: Controlador HP VAN

- **HP VAN SDN [9]:** El controlador de Hewlett Packard, Virtual Applications Networks (VAN), provee un punto unificado de control para una red OpenFlow, simplificando la gestión, orquestación y la provisión de la red. Permite el desarrollo servicios de red basada en aplicaciones y brinda APIs para que programadores puedan innovar soluciones para cualquier red en particular. Está diseñado para operar en campus, centro de datos, y redes de proveedores de servicios. Algunas características de este controlador son:

  - Plataforma de clase empresarial que permite el desarrollo de un gran número de innovaciones en la red.

[9] HP VAN, Disponible en: http://h17007.www1.hp.com

- Soporta desde OpenFlow versión 1 hasta la versión 1.3
- Soporta más de 50 switches físicos
- APIs abiertas para la interacción de desarrolladores externos
- Arquitectura de controlador extensible, escalable y resiliente.

Ilustración 11: Logo NEC

- **ProgrammableFlow PF6800 [10]:** Este controlador de la empresa japonesa NEC (Nippon Electric Company) está en el centro del tejido de red basado en OpenFlow. Proporciona un punto de control para las redes físicas y de gestión para las redes virtuales y físicas. El controlador se considera programable, así como estandarizado. Se integra tanto con OpenStack, como con Microsoft System Center Virtual Machine Manager para una gestión de red y orquestación añadidas. El controlador también incluye la tecnología de red de inquilino virtual de NEC, que permite redes aisladas, con varios inquilinos.

Ilustración 12: Logo Nuage

- **El Controlador de Servicios Virtualizados (VSC)[11]:** Perteneciente a Nuage Networks, permite la visión completa de una red por inquilino y las topologías de servicio, mientras externaliza plantillas de servicios de la red definidos a través del Directorio de Servicios Virtualizados de Nuage Networks. El directorio de servicios es un motor de políticas que utiliza el análisis y las reglas de red para autorizar permisos basados en roles. El VSC envía mensajes usando esas reglas a la plataforma de conmutación y ruteo virtual de Nuage. La plataforma detecta ya sea la creación o supresión de una máquina virtual y luego le pregunta al controlador SDN si hay una política instalada para ese inquilino. Si existe una regla, la conectividad de red se establece inmediatamente.

[10] ProgrammableFlow PF6800, Disponible en: https://www.necam.com/sdn/doc.cfm?t=PFlowController
[11] El Controlador de Servicios Virtualizados (VSC), Disponible en: http://www.nuagenetworks.net/

Ilustración 13: Controlador NSX

- **El controlador NSX [12]:** Una creación de VMWare, se considera un sistema de gestión de estado distribuido que controla las redes virtuales y superpone los túneles de transporte. Es el punto de control central para todos los switches lógicos dentro de una red. El controlador mantiene la información de las máquinas virtuales, hosts, switches lógicos y VXLANs, mientras usa APIs en dirección norte para hablar con las aplicaciones. Cuando se trabaja con el controlador, las aplicaciones comunican lo que necesitan, y el controlador programa todos los switches virtuales bajo el control NSX en dirección sur para cumplir con esos requisitos. El controlador podría ejecutarse de dos maneras dentro de NSX: ya sea como un clúster apartado de máquinas virtuales en un entorno vSphere, o en los aparatos físicos para aquellos con hipervisores mixtos.

Como ya se vio más arriba, el controlador es el núcleo de estas redes, es la parte más importante, el encargado de la "inteligencia" de la red. Él es quien toma las decisiones como la distribución de recursos, decisiones sobre paquetes que no coinciden con las tablas de flujo de los switches, creación, modificación, o eliminación de entradas de flujo, ejecuta las instrucciones que le proporcionan las distintas aplicaciones, básicamente, centraliza el funcionamiento de la red que es la característica básica de las SDN.

2.2.3.3. Clasificación de los controladores.

Según el tipo de red en el que se encuentre, se lo puede clasificar en distintos modos de funcionamiento:

- **Emplazamiento:** Se separa en dos configuraciones, una es centralizada, donde un solo controlador maneja a toda la red y otra en donde hay un controlador para un conjunto de dispositivos de una red.
- **Por flujo:** Donde cada entrada define un flujo en particular o donde una entrada define un conjunto de flujos.

[12] El controlador NSX, Disponible en: http://www.vmware.com/ar/products/nsx

- **Comportamiento:** Puede ser reactivo, donde el controlador va a agregar una nueva entrada de flujo cuando llegue un paquete al switch que no tenga coincidencia con ninguna entrada, o proactivo, donde el controlador completa la tabla de flujo antes de que lleguen flujos nuevos. Cuando es reactivo, cada paquete nuevo que llegue al switch va a tener un tiempo de retardo que se corresponde a su procesamiento en el controlador y a la implementación de una nueva entrada de flujo, y si se pierde la conexión entre ambos se corta, el switch no podrá conmutar paquetes de flujos nuevos, limitando su conectividad. En el modo proactivo, no hay retardo y no se pierde conectividad si se corta la comunicación entre el controlador y el switch.

Un controlador puede ser, desde un simple software en una computadora, encargado de gestionar las tablas de flujos de los dispositivos de una red, quedando en mano de un solo administrador encargado de la gestión, hasta múltiples administradores de red que pueden gestionarla por medio de distintas cuentas y con la posibilidad de poder configurar distintos conjuntos de flujos de la red.

2.2.4. Capa de aplicación

Esta capa es donde los desarrolladores pueden realizar sus peticiones al controlador sobre el comportamiento de la red. Aquí se automatiza el monitoreo de la red mediante aplicaciones que los desarrolladores y los encargados de las red pueden crear o modificar de algunos ya creados. Esta capa se comunica con el controlador por medio de las NorthBound APIs y el controlador es el encargado de enviar las nuevas modificaciones a los switches de la red para que se adapten al comportamiento que se desea. Dentro de las aplicaciones SDN que podemos usar se encuentran [13]:

- **Enrutamiento adaptativo**: Tradicionalmente el enrutamiento conlleva una compleja implementación y convergencia lenta. Con SDN podemos crear un balanceador de carga y distintas aplicaciones que monitoreen la red en busca de optimizaciones en tiempo real. Al tener información constante sobre el estado de la red es mucho más rápido y fácil poder adaptarla a las necesidades del momento.

[13] Trabajo Final de Grado de David Andrés Serrano Carrera, “Redes Definidas por Software (SDN): OpenFlow”, Universidad Politécnica de Valencia.

- **Itinerancia sin interrupciones**: La transferencia o handover al hacer uso de dispositivos móviles hace necesario prever un servicio continuo. Con SDN, las redes entre diferentes portadores con diferentes tecnologías tienen un plano de control común. Hay diferentes propuestas de transferencia como Hoolock en redes Wi-Fi y WiMAX u Odin para WLANs de empresa.

- **Mantenimiento de la red**: Herramientas de configuración típicas como traceroute o tcpdump no son una solución para mantener una red extensa de forma automática, ya que tienden al error humano. Al tener una visión global de la red y un control centralizado de la configuración, SDN permite nuevas herramientas de diagnóstico como ndb u OFRewind.

- **Seguridad de la red**: Las redes tradicionales utilizan cortafuegos o servidores proxy para proteger las redes físicas, siendo su implementación una tarea pesada para el administrador. SDN permite analizar patrones de tráfico para posibles problemas de seguridad como ataques de denegación de servicio, guiar paquetes sospechosos a sistemas de prevención de intrusión (IPS), modificar reglas de reenvío para bloquear tráfico, o dar privacidad a los usuarios con ejemplos como AnonyFlow.

- **Virtualización de la red**: Lo que se pretende es permitir la existencia en una infraestructura compartida de múltiples arquitecturas de red heterogéneas. Normalmente, lo que se hace es separar la red física en múltiples instancias virtuales y asignarlas a diferentes usuarios, controladores o aplicaciones, mediante túneles o etiquetas VLAN y MPLS, convirtiéndose en una tarea compleja. Con SDN, la configuración se realiza en el controlador con plataformas como libNetVirt, una librería de virtualización de red o FlowVisor, colocando un proxy transparente para filtrar mensajes de control según la red virtual.

- **Cloud Computing:** Los centros de datos en las redes para ‘cloud computing’ necesitan algunas características, como escalabilidad, independencia de la localización para abastecer recursos dinámicos o diferenciación de QoS. La conmutación virtual es usada para la comunicación entre máquinas virtuales en el mismo host e implementada en SDN como Open vSwitch.

2.2.5. Southbound API

En una arquitectura SDN, el Interfaz de Programación de Aplicaciones (API) Southbound es usado para la comunicación entre el controlador SDN y los switches y routers de la red. Puede ser OpenSource o propietario. OpenFlow es el interfaz estándar definido por la ONF para la comunicación del plano de control con el de datos, permitiendo el acceso directo y la manipulación de las tablas de flujos de los dispositivos de red tanto físicos como virtuales. Aparte de OpenFlow existen otras southbound APIs como [14]:

- **Border Gateway Protocol (BGP):** Utilizado para intercambiar información de enrutamiento entre hosts de gateways en una red de sistemas autónomos. Se busca una utilidad en las SDN hibridas.
- **NetConf:** Es un protocolo de gestión de red de la Internet Engineering Task Force (IETF), es una forma segura de configurar un firewall, switch, router u otro dispositivo. Fue incorporado recientemente por la ONF y su uso se está volviendo obligatorio para la configuración de dispositivos compatibles con OF.
- **Protocolo de Presencia y Mensajería Extensible (XMPP):** Tiene uso en la mensajería instantánea y detección de presencia en la línea. Funciona entre o en los servidores y facilita la operación en tiempo real. Busca ser una alternativa a OF en SDN hibridas y su función sería la de la de asistir al controlador para la distribución de información de plano de control a los puntos finales de servidor.
- **Protocolo de Gestión de Base de Datos OpenvSwitch (OVSDB):** Es un protocolo de configuración OF destinado a administrar las implementación OpenvSwitch.

2.2.6. Northbound API

La Northbound API es usada para comunicar el controlador SDN con los servicios y aplicaciones corriendo fuera de la red. Se usa para facilitar la innovación y permitir una automatización y orquestación eficiente de la red para alinearse con las necesidades de las distintas aplicaciones que corren en la capa superior. Para este interfaz no hay un protocolo estandarizado y se está trabajando en la actualidad en ello. Muchos expertos en la materia consideran que la clave del éxito en SDN no se encuentra en OpenFlow sino en la Northbound API esto se debe a dos diferencias notables.

[14] Researchgate, Disponible en: https://www.researchgate.net/

- **Las Northbound API no dependen del hardware:** no está atada al tiempo de desarrollo o innovación del hardware, si observamos la figura podemos ver que las dos puntas del interfaz une productos de software. En cambio, por ejemplo OpenFlow, posee versiones recientes que no todos los switches pueden soportar.

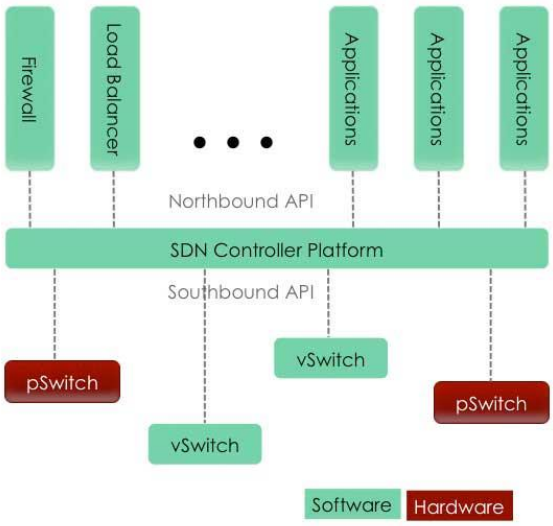

Ilustración 14: Ubicación de las Northbound APIs [Fuente: ProjektberichtSDN.pdf]

- **La mayor parte del valor de las SDN se creara y se capturara en la Capa de Aplicación:** Serán las aplicaciones como firewalls, balanceadores de cargas, softwares de seguridad, etc., los que trabajaran para modificar la red según las necesidades de los usuarios, estas peticiones serán enviadas al controlador que modificara las configuraciones de los switches para que trabajen de la manera deseada.

Por esto es que muchas empresas ya están trabajando para proponer un estándar para este interfaz, como así también trabajan muchos proyectos Open-Source en conseguir la misma meta.

2.3. Protocolo Openflow

Es una tecnología de switching y se define como “un protocolo emergente y abierto de comunicaciones que permite a un servidor de software determinar el camino de reenvió de paquetes que debería seguir una red de switches” según la Open Networking Foundation. Su propósito es ser el comunicador entre el controlador y los distintos dispositivos de la red, logrando que la red se vea como un todo y sea el controlador el que tome las decisiones sobre

los paquetes que viajan en ella, dejándole a los dispositivos de encaminamiento la función de reenvío solamente. Este protocolo, como ya dijimos, define la comunicación entre el switch Openflow y el controlador. El switch establece una comunicación con el controlador, usando un determinado puerto, normalmente el 6633, e iniciará una comunicación TCP estándar. Cuando se establecen las comunicaciones, cada lado envía un mensaje de HELLO junto con la versión de OpenFlow más alta que puede soportar, y así se negocia la versión mínima entre la que se envió y la que se recibió en los dispositivos. Si se soporta la versión negociada, la conexión será iniciada, sino se enviará un mensaje de HELLO-FAILED y la conexión será finalizada.

Podemos encontrar tres mensajes distintos en el protocolo OpenFlow [15]:

- **Mensaje del controlador al Switch:** Es iniciado por el controlador y su objetivo es conocer y/o actuar sobre el estado actual del switch. Dentro de este mensaje podemos encontrar cuatro tipos distintos:
  - **Modificar estado:** Añade, modifica o elimina entradas en tablas de flujo y fija características en los puertos.
  - **Lectura de estado:** Sirve para consultar al switch sobre las estadísticas del tráfico (se usa la parte de las estadísticas de las entradas de flujo).
  - **Enviar paquete:** Indica que envié paquetes por un puerto específico luego de agregar una nueva entrada de flujo a la tabla del switch.
  - **Notificaciones:** El controlador lo utiliza para asegurarse de que los objetivos de un mensaje se han cumplido o para recibir notificaciones por operaciones finalizadas.
- **Mensajes asíncronos:** Son mensajes enviados por los switch hacia el controlador. Existe cuatro tipos de mensajes distintos:
  - **Entrada de paquete:** Se envía al controlador cuando algún paquete no encuentra coincidencia con alguna entrada de la tabla de flujo del switch o la acción asociada con esa entrada es reenviar el paquete al controlador.
  - **Modificación de flujo:** Se envía cuando se ha agregado, eliminado o cambiado con éxito una entrada en la tabla de flujo del switch.
  - **Estado de puertos:** Se envía cuando se modifica el estado de algún puerto del switch.
  - **Error:** El switch informa al controlador si existen problemas con estos mensajes.

[15] Wikipedia, OpenFlow. Disponible en: http://es.wikipedia.org/wiki/Openflow

- **Mensajes síncronos:** Son enviados en cualquier dirección sin solicitud previa. Existen tres tipos distintos:
  - **Hello:** Mensaje intercambiado durante la duración de la negociación de conexión.
  - **Echo:** Es utilizado por cualquiera de los dos dispositivos para verificar el ancho de banda, la latencia, o, simplemente, la conexión entre los dos dispositivos. Una petición Echo request es respondida por un mensaje Echo reply desde el destino al origen.
  - **Mensaje de proveedor:** Esta planeado para futuras versiones de OpenFlow y consta de mensajes para ofrecer funcionalidades adicionales a los switch.

Está claro que el protocolo OpenFlow está totalmente pensado para la implementación de las SDN y se presenta en la actualidad como el primer estándar desarrollado para las redes definidas por software. A pesar de sus características, la Open Networking Foundation, organización desarrolladora y promotora de Software Defined Networks, sigue trabajando para poder agregarle más beneficios para mejorar la interacción entre el controlador y los Switch OpenFlow.

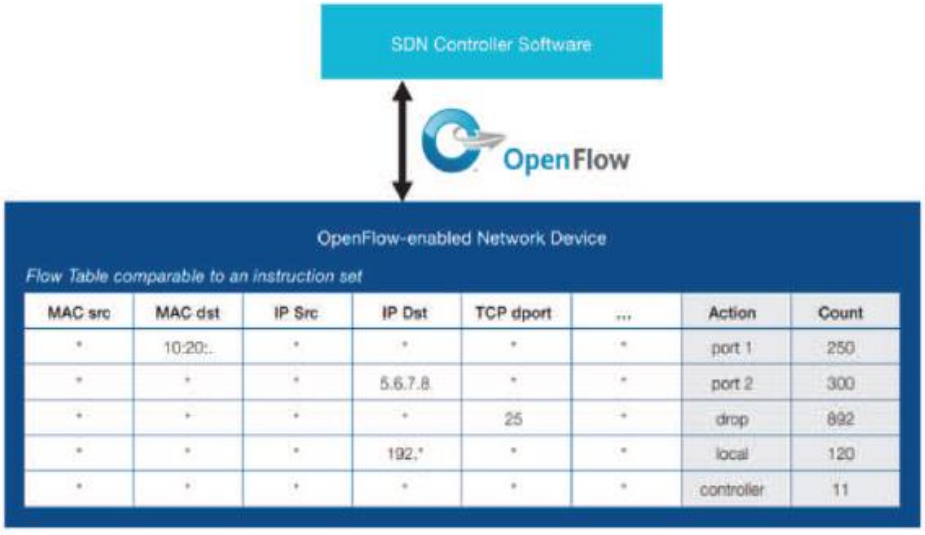

Ilustración 15: Tabla de flujo OpenFlow [Fuente: MaldonadoHidalgoDiegoArmando2014.pdf]

2.4. Switch OpenFlow

El switch OpenFlow cuenta con tres partes esenciales para su funcionamiento:

- **Tablas de flujo**: Indican como debe ser procesado cada paquete que entre en el switch.

Cada switch puede poseer una o más tablas de flujo utilizadas para decidir cómo re direccionar los flujos entrantes. Una tabla de flujo consiste en entradas de flujo. Cada entrada de flujo está compuesta por seis partes:

- **Campos de comparación:** Usados para ser comparados con los paquetes entrantes. Para trabajar con *OpenFlow* un *switch* debe soportar como mínimo, en su procesamiento en *pipeline*, los campos de comparación requeridos. Estos campos requeridos deben ser soportados por lo menos en una tabla de flujo.

In Port	VLAN ID	Ethernet			IP			TCP	
		SA	DA	Type	SA	DA	Proto	Src	Dst

Ilustración 16: Campos de comparación de tablas de flujo [Fuente: MaldonadoHidalgoDiegoArmando2014.pdf]

- **Prioridad:** Prioridad de la entrada de flujo. Si el paquete tiene coincidencias con múltiples entradas de flujo solo la entrada con mayor prioridad será seleccionada. Pudiendo esta enviar el paquete hacia otra tabla de flujo para que se realicen otras acciones, o bien direccionar el paquete de acuerdo a las configuraciones que tenga esa entrada.

- **Contadores:** Usados con propósitos de estadística, para darle seguimiento al número de paquetes, bytes, errores, etc. de cada flujo, tabla, puerto, etc. En la siguiente tabla se enlistan todos los contadores que posee OpenFlow y que pueden ser activables, algunos son esenciales y otros son opcionales.

Contadores	Bits	
Tabla de flujo		
Reference Count (active entries)	32	Requerido
Packet Lookups	64	Opcional
Packet Matches	64	Opcional
Entrada de flujo		
Received Packets	64	Opcional
Received Bytes	64	Opcional
Duration (seconds)	32	Requerido
Duration (nanoseconds)	32	Opcional
Puerto		
Received Packets	64	Requerido
Transmitted Packets	64	Requerido
Received Bytes	64	Opcional

Transmitted Bytes	64	Opcional
Receive Drops	64	Opcional
Transmit Drops	64	Opcional
Receive Errors	64	Opcional
Transmit Errors	64	Opcional
Receive Frame Alignment Errors	64	Opcional
Receive Overrun Errors	64	Opcional
Receive CRC Errors	64	Opcional
Collisions	64	Opcional
Duration (seconds)	32	Requerido
Duration (nanoseconds)	32	Opcional
Cola		
Transmit Packets	64	Requerido
Transmit Bytes	64	Opcional
Transmit Overrun Errors	64	Opcional
Duration (seconds)	32	Requerido
Duration (nanoseconds)	32	Opcional
Grupo		
Reference Count (ow entries)	32	Opcional
Packet Count	64	Opcional
Byte Count	64	Opcional
Duration (seconds)	32	Requerido
Duration (nanoseconds)	32	Opcional
Bucket de Grupo		
Packet Count	64	Opcional
Byte Count	64	Opcional
Métrica		
Flow Count	32	Opcional
Input Packet Count	64	Opcional
Input Byte Count	64	Opcional
Duration (seconds)	32	Requerido
Duration (nanoseconds)	32	Opcional
Banda de métrica		
In Band Packet Count	64	Opcional
In Band Byte Count	64	Opcional

Tabla 2: Contadores OpenFlow [Fuente: Propia]

- **Instrucciones:** Especifican las acciones a tomar de acuerdo al flujo al que pertenece el paquete. Son instrucciones para modificar un set de acciones o procesamiento en pipeline. Cada entrada de flujo contiene un conjunto de instrucciones que son ejecutados cuando un paquete coincide con esta. Estas

instrucciones provocan cambios en el paquete, conjunto de acciones y/o procesamiento en *pipeline*.

Entre las instrucciones definidas se encuentran:

- **Apply-Actions:** Con este campo se pueden realizar acciones específicas de forma inmediata sin alterar el conjunto de acciones del paquete. Dentro de estas acciones hay tres que son las principales:
  - ✓ Reenviar paquete a través de un puerto, o a otra tabla de flujo.
  - ✓ Encapsular el paquete y enviarlo al controlador para decidir qué hacer con él. Esto normalmente sucede cuando llega un paquete de un flujo nuevo.
  - ✓ Eliminar paquete.

- **Clear-Actions:** Elimina todas las acciones en el *conjunto de Acciones* de forma inmediata.

- **Write-Actions:** Agrega las acciones especificadas al *Set de Acciones* actual.

- **Write-Metadata:** Escribe el valor del metadato enmascarado en el campo de metadato.

- **Goto-Table:** Señala hacia cual tabla de la pipeline se debe mandar el paquete, el ID de la tabla nueva debe ser mayor a la actual. Las entradas de flujos de la última tabla en el procesamiento en *pipeline* no pueden incluir esta instrucción. En los switches donde solo se posea una tabla de flujo esta instrucción no se define.

- **Meter**: Dirige el paquete a la métrica especificada donde el mismo puede ser descartado de acuerdo a la configuración y el estado de dicha métrica.

El set de instrucciones asociados a una entrada de flujo no puede contener más de una instrucción de cada tipo. Las instrucciones son ejecutadas en el orden especificado en la lista anterior. Un *switch* puede rechazar una entrada de flujo si no puede ejecutar las instrucciones en la misma. Una tabla de flujo también puede o no soportar todas las instrucciones.

- **Tiempo de espera:** Límite de tiempo o tiempo de inactividad para que una entrada de flujo expire. Este tiempo de vida se puede modificar para que el flujo no expire o para cambiar el tiempo con el cual el controlador lo pone por defecto. Distintos controladores poseen tiempos de espera estándar diferentes entre sí.

- **Cookie:** Tipo de datos cuyo valor es elegido por el controlador. Usado por el mismo para filtrar estadísticas de flujo, modificación de un flujo y eliminación de un flujo. No se usa al procesar paquetes.

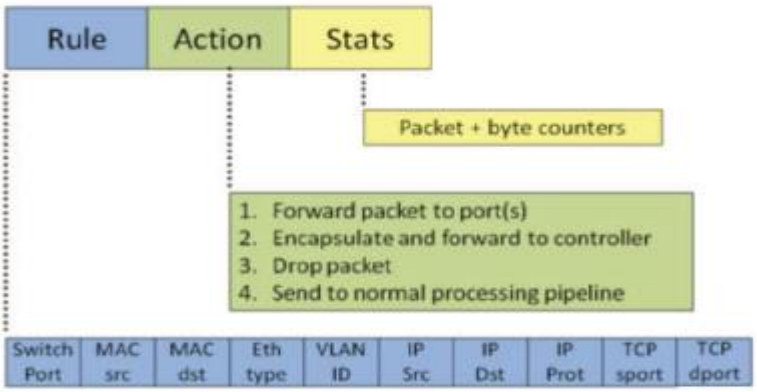

Ilustración 17: Campos de una entrada de flujo [Fuente: ProjektberichtSDN.pdf]

- **Canal especial o seguro:** Es el medio de comunicación entre el switch y el controlador. Por medio de este canal, y de distintos protocolos (El más utilizado ampliamente es OpenFlow pero no es el único), el switch puede enviar paquetes hacia el controlador, recibir configuraciones, y enviar datos de control al controlador, mientras que este último lo utiliza para crear, modificar y eliminar entradas de flujo en las tablas del switch, devolver paquetes analizados, obtener datos de la red, etc. Como se dijo antes, el protocolo más usado es OpenFlow, el cual es un protocolo de comunicación para unir la capa de hardware con la capa de control y que será explicado más adelante en profundidad. Ya que este canal puede ser vulnerable a ataques o robo de información, generalmente se lo codifica por medio del protocolo de seguridad TLS (Transport Layer

Security), este genera una comunicación segura entre el cliente y el servidor. Algunos controladores no soportan este protocolo por lo que utilizan TCP (Transmission Control Protocol).

- **Controlador:** El switch OpenFlow necesita del controlador para que este le agregue las entradas de flujo y saber qué hacer con los paquetes entrantes. Sin un controlador, el switch no tendrá ninguna información con la que matchear los paquetes que le llegan y los desechara, por lo que no circularan paquetes dentro de la red.

Cada Switch OpenFlow puede poseer múltiples tablas de flujo y cada tabla de flujo contiene múltiples entradas de flujo. El procesamiento en *pipeline* de OpenFlow define como los paquetes interactúan (se comparan) en las tablas de flujo. El paquete entrante se compara en la primera tabla de flujo, si no existen coincidencias, se actualiza el número de la tabla de flujo y se comparan con la n-ésima tabla del *pipeline*, hasta encontrar una coincidencia, después de la cual se ejecutara un conjunto de acciones sobre el paquete y se enviara al correspondiente destino, caso contrario se tienen dos opciones que dependen de la configuración del switch, se puede enviar al controlador o se descartara el paquete.

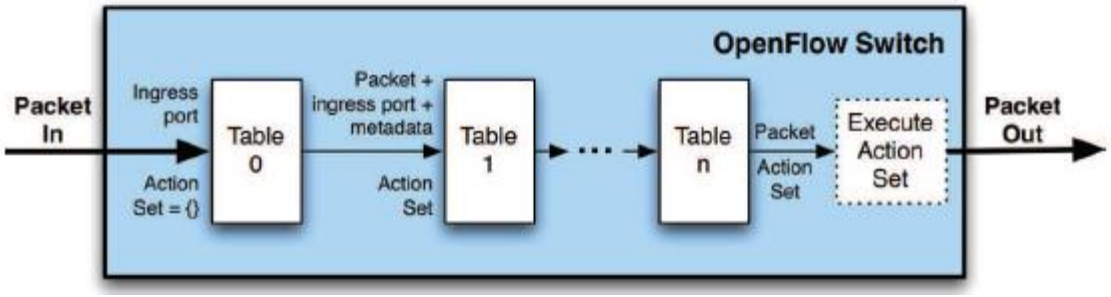

Ilustración 18: Flujo de paquetes a través de la pipeline [Fuente: ContrerasPardoCarlosAlberto2014.pdf]

En resumen, y como se puede ver en la siguiente figura, procesado del paquete es el siguiente:

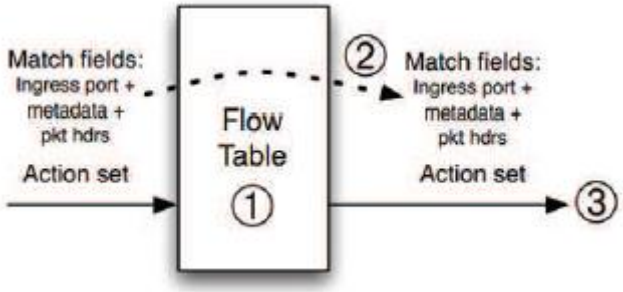

Ilustración 19: Procesamiento del paquete en la tabla de flujo [Fuente: ContrerasPardoCarlosAlberto2014.pdf]

- **1)** El encabezado del paquete entrante se compara con las entradas de flujo de la tabla de flujo y se elige la coincidencia con mayor prioridad.
- **2)** Cuando el paquete ha sido "*Matcheado*" se aplican distintas instrucciones:
  - Modificación del paquete y actualización de los campos de comparación.
  - Actualización del conjunto de acciones.
  - Actualización del metadato que determina la coincidencia del flujo.
- **3)** Envío del dato coincidente y su correspondiente conjunto de acciones a la siguiente tabla de flujo.

Los switch OpenFlow o de nueva generación se pueden separar en dos grupos, los switches OpenFlow-Only y los Switches Híbridos.

- **Switch OpenFlow-Only:** Estos dispositivos contienen una o más tablas de flujos y, a su vez, cada tabla de flujo contiene múltiples entradas de flujo. Es indispensable que disponga de una tabla de flujo como mínimo, y, mientras menos tablas de flujo posea, más simple será el procesamiento de paquetes. Este switch solo funciona con un controlador conectado, no es capaz de realizar las tareas de un switch convencional. El controlador se encarga de rellenar sus tablas de flujos con entradas para poder empezar a trabajar en el procesado de paquetes y así tener una red funcional. Si por alguna razón el controlador se desconecta, este switch no tendrá la capacidad de seguir realizando sus funciones una vez que las entradas de flujo sean eliminadas por su tiempo de vida.

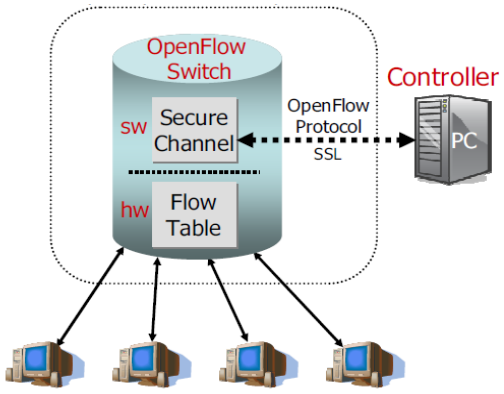

Ilustración 20: Arquitectura del switch OpenFlow-Only [Fuente: ContrerasPardoCarlosAlberto2014.pdf]

- **Switch Hibrido:** Estos equipo son capaces de realizar el direccionamiento Ethernet tradicional y, también, el direccionamiento por medio del protocolo OpenFlow. Estos

dispositivos tienen la capacidad de trabajar como los switches convencionales, haciendo switching Ethernet de capa 2, separación en VLANs, STP, etc. Mientras que también posee la capacidad de conectarse a un controlador que modifique sus entradas de flujo por medio del protocolo OpenFlow y así tener un control más granular de lo que se mueve por la red, pudiendo hacer tareas de capa 2, capa 3 o routing, QoS, etc.

3. Tecnologías actuales – Estado del Arte

3.1 MININET

3.1.1. Introducción

Con el fin de promocionar el uso de SDN, aprender sus características, desarrollar nuevos protocolos, y favorecer su enseñanza a las nuevas generaciones, en la Universidad de Stanford se desarrolló Mininet, un emulador de red virtual, con un conjunto de switch, routers, links, controladores, host, etc. Corriendo en un simple núcleo Linux, brindando la posibilidad de acceder a cada dispositivo individualmente, cambiando sus características a nuestro gusto. También nos da la posibilidad de modificar los enlaces que unen estos dispositivos, cambiando su retardo, su ancho de banda, etc.

3.1.2 Características y limitaciones

Para poder tener una experiencia más real aun. Entre sus puntos fuertes se encuentran [16]:

- Es muy veloz, dado que crear una red virtual toma tan solo unos pocos segundos.
- Posibilidad de crear diferentes tipos de redes, desde dos host con un solo controlador, hasta centros de datos, o redes para una universidad o ciudad.
- Se puede combinar con cualquier programa que corra en Linux para sacarle más provecho, por ejemplo, podemos monitorizar los paquetes OpenFlow con Wireshark.
- Permite modificar a gusto las tablas de flujo de los switches de la red virtual, adaptándola a nuestras necesidades. Característica básica de las SDN.
- Posibilidad de correr Mininet en una computadora de escritorio, Notebook, máquina virtual, etc.
- Su uso es muy fácil y se pueden crear distintos tipos de topologías mediante scripts en Python.

[16] Mininet, Disponible en : http://mininet.org/

- Es un proyecto Open-Source, por lo cual, cualquiera puede acceder a su código y modificarlo, arreglar errores, agregar características, etc.

A pesar de sus buenas características, no es perfecto y tiene sus limitaciones:

- Como cualquier virtualización, comparte recursos con la máquina que la aloja.
- Los host creados no pueden acceder a internet ya que no se puede hacer NAT al exterior.

Es una herramienta potente para la enseñanza, investigación y desarrollo de las SDN.

Su uso es bastante básico, con solo configurarla y poner a correr la máquina virtual, se pueden crear redes virtuales. Desde una red básica con el comando **sudo mn** o **sudo mn --topo minimal**, que consta de un controlador, un switch y dos hosts.

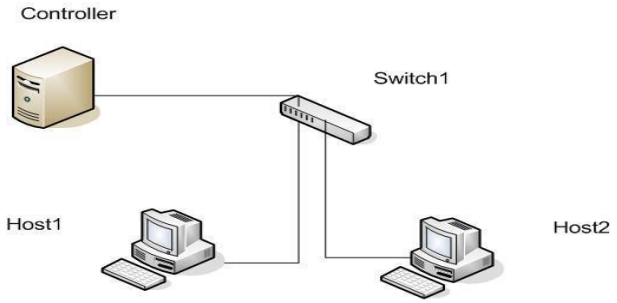

Ilustración 21: Topología mínima [Fuente: Propia]

Hasta una red de las dimensiones que queramos con los parámetros de host, switch, controlador y links que queramos.

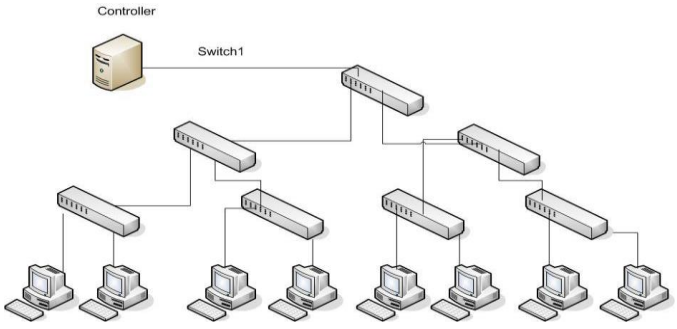

Ilustración 22: Topología árbol [Fuente: Propia]

3.1.3. MiniEdit

El propio entorno de Mininet posee múltiples bloques de aplicaciones que complementan la funcionalidad de este. Uno de ellos es MiniEdit, un entorno gráfico y mucho más amigable para la creación de redes virtuales, pensado para un ambiente educativo.

Para ejecutarlo, habrá que cerrar todos los procesos creados en la prueba anterior mediante *exit* (salir de la red creada por Mininet), *killall controller* (terminar el proceso del controlador) y sudo *mn –c* (para 'limpiar' cualquier elemento creado en la red anterior). Ahora, desde una terminal conectada mediante SSH a la máquina virtual se ejecuta:

- ***sudo ~/mininet/examples/miniedit.py***

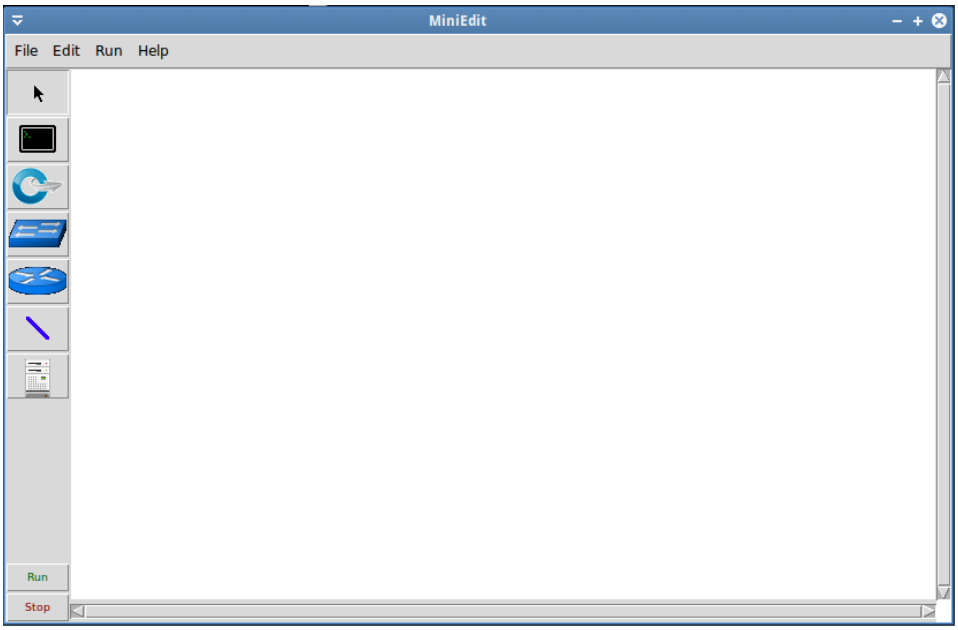

Ilustración 23: GUI Miniedit [Fuente: Propia]

Desde la sencilla interfaz que se presenta, se puede observar una serie elementos para construir una red en la parte derecha de la ventana, teniendo los siguientes, por orden de aparición:

- **Flecha de selección**: Podemos mover los elementos de la red.
- **Host**: Nos brinda la posibilidad de crear un host. Si se mantiene apretado el botón derecho sobre cada uno de los hosts creados se puede acceder a sus propiedades y modificarlas.
- **Switch**: Creamos un conmutador OpenFlow Open VSwitch por defecto que luego puede ser cambiado en sus propiedades.
- **Switch tradicional**: Crea un switch de capa 2 tradicional que aprende la información de la red de manera normal, y funciona de forma independiente. Una de sus características es que posee, por defecto, el protocolo Spanning Tree desactivado y se corre peligro de bucles.
- **Router tradicional**: Se crea un enrutador básico que funciona como un host con el reenvío IP activado. No puede ser configurado desde la interfaz gráfica.

- **Enlace de red:** Se crea un link, solo con hacer click en un dispositivo y arrastrar el mouse hasta otro distinto. También, al igual que Mininet operado por línea de comandos, se nos permite modificar las propiedades de los enlaces, solo se debe mantener apretado el click derecho sobre el link.
- **Controlador**: Se pueden añadir muchos tipos de controlador, aunque por defecto se crea el controlador de referencia en OpenFlow, que implementa el comportamiento de un conmutador 'learning' de capa 2. En el menú de Propiedades se puede configurar las propiedades del controlador.
- **Ejecutar / Parar**: Botones para empezar o terminar la simulación. Cuando está en ejecución, aparecen otras propiedades en cada elemento al clicar con el botón derecho.

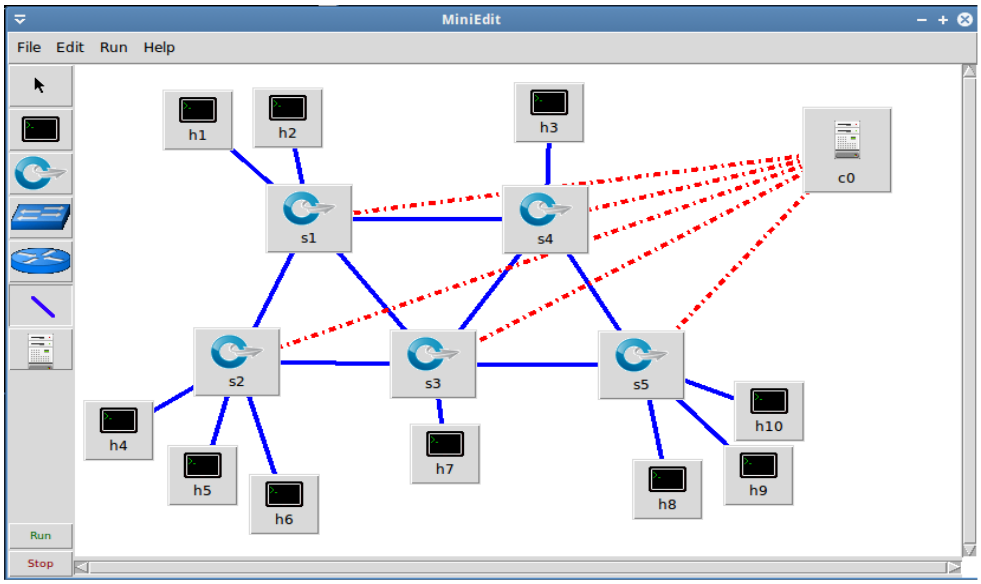

Ilustración 24: Topología en Miniedit [Fuente: Propia]

Y acá podemos ver una topología creada en miniedit, con 5 switches OpenFlow, un controlador, y 10 hosts.

3.2 Wireshark

3.2.1. Introducción

Antes conocido como Ethereal, es un analizador de protocolos utilizado para realizar análisis y solucionar problemas en redes de comunicaciones, para desarrollo de software y protocolos, y como una herramienta didáctica. Cuenta con todas las características estándar de un analizador de protocolos de forma únicamente hueca.

La funcionalidad que provee es similar a la de tcpdump, pero añade una interfaz gráfica y muchas opciones de organización y filtrado de información. Así, permite ver todo el tráfico que pasa a través de una red (usualmente una red Ethernet, aunque es compatible con algunas otras) estableciendo la configuración en modo promiscuo. También incluye una versión basada en texto llamada tshark.

Permite examinar datos de una red viva o de un archivo de captura salvado en disco. Se puede analizar la información capturada, a través de los detalles y sumarios por cada paquete. Wireshark incluye un completo lenguaje para filtrar lo que queremos ver y la habilidad de mostrar el flujo reconstruido de una sesión de TCP.

3.2.2. Características

Algunas de sus características son [17]:

- Inspección profunda de miles de protocolos, con mucho más siendo adheridos todo el tiempo.
- Captura en vivo y análisis offline.
- Multiplataforma, corre en Windows, Linux, OS X, Solaris, FreeBSD, NetBSD y muchos otros.
- Los mejores filtros para separar protocolos.
- Análisis de VoIP
- Lee/Escribe diferentes formatos de archivos de capturas: tcpdump (libpcap), Pcap NG, Catapult DCT2000, Cisco Secure IDS iplog, Microsoft Network Monitor, etc.

3.2.3. Implementación de MININET y Wireshark

Se demuestra la sinergia de estas dos tecnologías en la investigación y desarrollos de las SDN con un ejemplo muy simple. Utilizaremos Mininet para crear un pequeña red virtual con una topología árbol, como controlador haremos uso de uno de los controladores Open-Source vistos más arriba que es POX, y luego analizaremos los paquetes OpenFlow que recorren la red con el software Wireshark.

[17] Wireshark, Disponible en: https://www.wireshark.org/

Lo primero es crear la red virtual con Mininet, en este caso se utilizara la topología tipo árbol (tree) con tres niveles.

Esto se logra escribiendo en el termina el comando:

- ***sudo mn --topo=tree,3 --link tc,bw=100 --controller=remote,port=6633***

Con esto se crea una red tipo árbol, con tres niveles, los enlaces tienen una velocidad de 100Mbits, y con un controlador remoto al cual se accede por el puerto 6633.

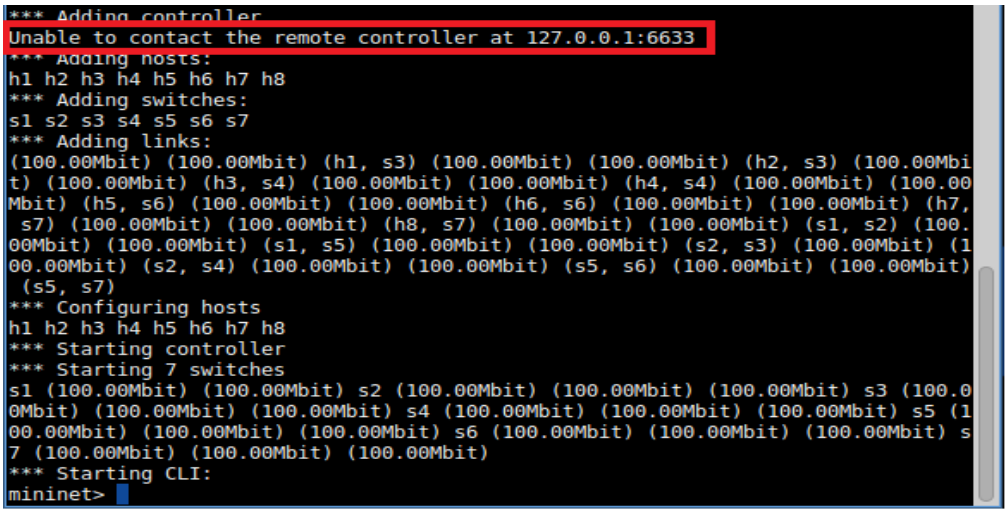

Ilustración 25: Creación de la red [Fuente: Propia]

Lo primero que notamos es que los switches no pueden conectarse con el controlador remoto, dado que todavía no dimos de alta el controlador.

Como no tiene un controlador que le indique que hacer con los paquetes que circulan por la red, si yo intento realizar un ping desde el host número 1 al host número 8, no se podrá realizar. Esto se hace con el comando:

- ***h1 ping –c4 h8***

Con esto envió cuatro paquetes ICMP de 64 bytes del host 1 al host 8 esperando respuesta a cada uno de ellos, pero como no hay controlador estos paquetes nunca llegan a destino.

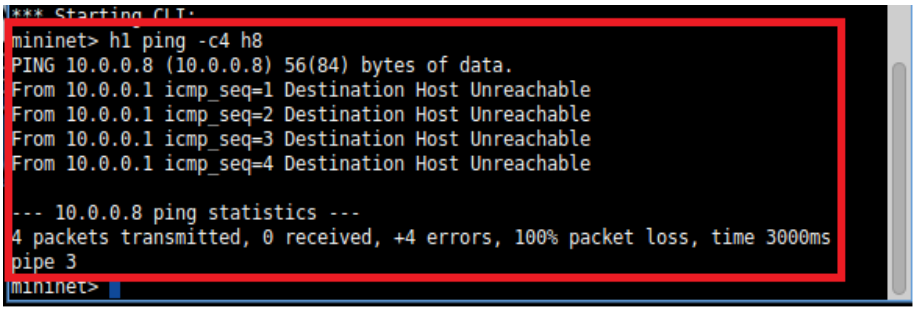

Ilustración 26: Ping rechazado [Fuente: Propia]

Ahora conectamos el controlador POX que armara una tabla de flujo en cada uno de los switches indicando que tienen que hacer con los paquetes que reciben. Esto se logra ingresando en otro terminal el comando:

- ***sudo ./pox.py forwarding.l2_learning***

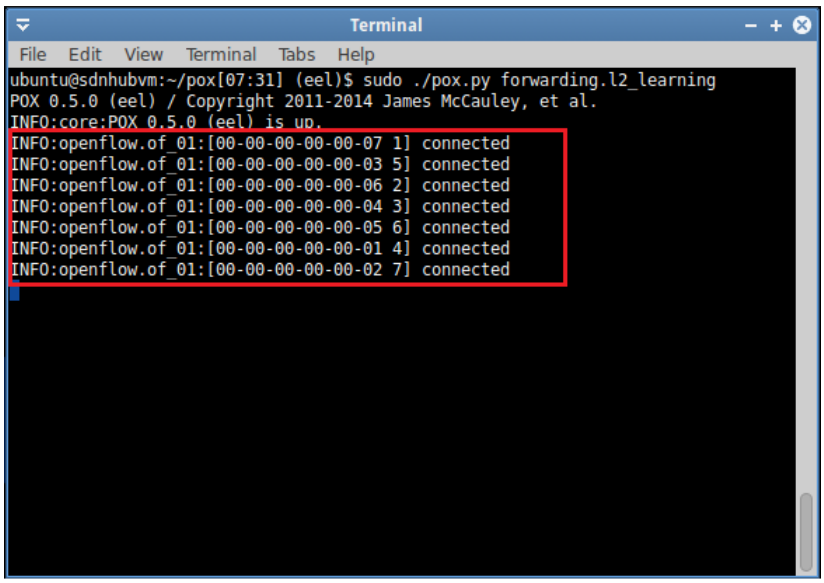

Ilustración 27: Ejecución del controlador [Fuente: Propia]

Este es el comando más básico del controlador POX, se pueden agregar módulos para agregar funciones extra que se verán más adelante. Podemos ver como detecta los 7 switches de nuestra topología.

Ahora si probamos realizar nuevamente los cuatro ping del host 1 al host 8 veremos que la operación se realiza con éxito y además podemos apreciar el retardo que experimenta el primer paquete con respecto a los siguientes.

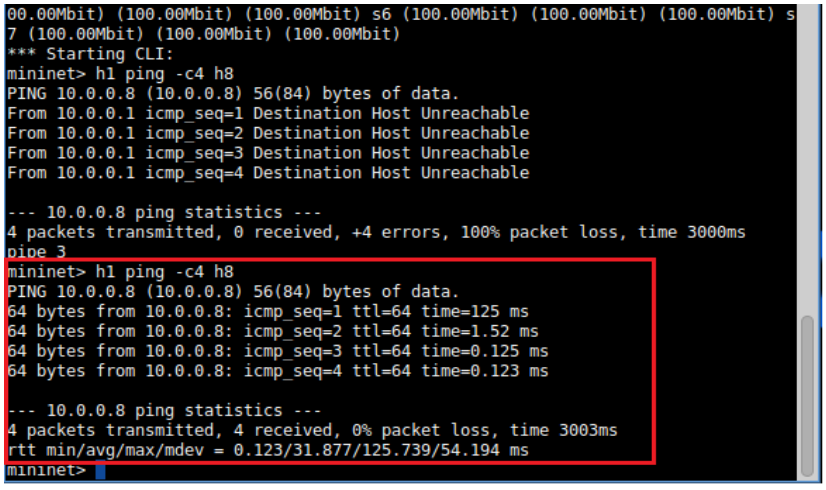

Ilustración 28: Ping con el controlador [Fuente: Propia]

También, como se dijo antes, se puede utilizar el software Wireshark para ver las comunicaciones entre el controlador y los switches de la red.

Se pueden ver los paquetes Hello, negociación de versión, configuraciones, etc.

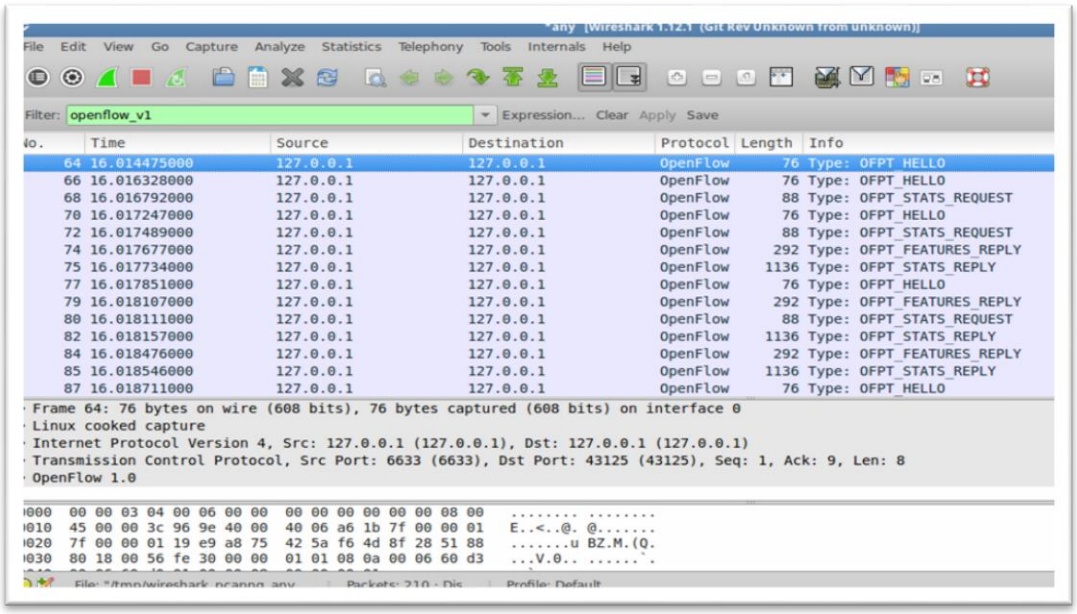

Ilustración 29: Primitivas de OpenFlow [Fuente: Propia]

Donde podemos analizar muchas cosas, por ejemplo, todos los paquetes tienen una cabecera común que nos indica que estamos trabajando con la versión 1.0 de OpenFlow y podemos ver en la pestaña “OpenFlow 1.0” sus características:

- **Versión:** Indica la versión del protocolo, en este caso es 1.0 pero el más nuevo es el 1.3.
- **Type:** Tipo del paquete, en el caso de la imagen se trata de un paquete *Hello* que se identifica con el número 0, el *Set_Config* se identifica con el número 9.
- **Length:** Longitud del paquete incluyendo la cabecera.
- **Transaction ID:** Identificación de la transacción asociada al paquete. Las parejas *Request-Reply* tienen el mismo identificador.

Podemos apreciar los distintos mensajes que intercambian los switches y el controlador al verse por primera vez.

- **Hello:** Se envía desde el controlador al switch y viceversa, en este mensaje se intercambian los números de versiones que soporta cada uno. Podemos identificar cual switch está hablando con el controlador por medio del campo de los puertos destino y fuente del paquete.

- **Set_Config:** El controlador pide que el switch le envíe el tiempo de expiración de los flujos que posee.
- **Features_Request y Features_Reply:** El controlador envió el mensaje Feature_Request al switch que no posee ningún dato después de la cabecera OpenFlow. El switch le responde al controlador con un mensaje Feature_Reply en el que le informa sobre los puertos que posee disponibles, velocidades, tablas soportadas y acciones.

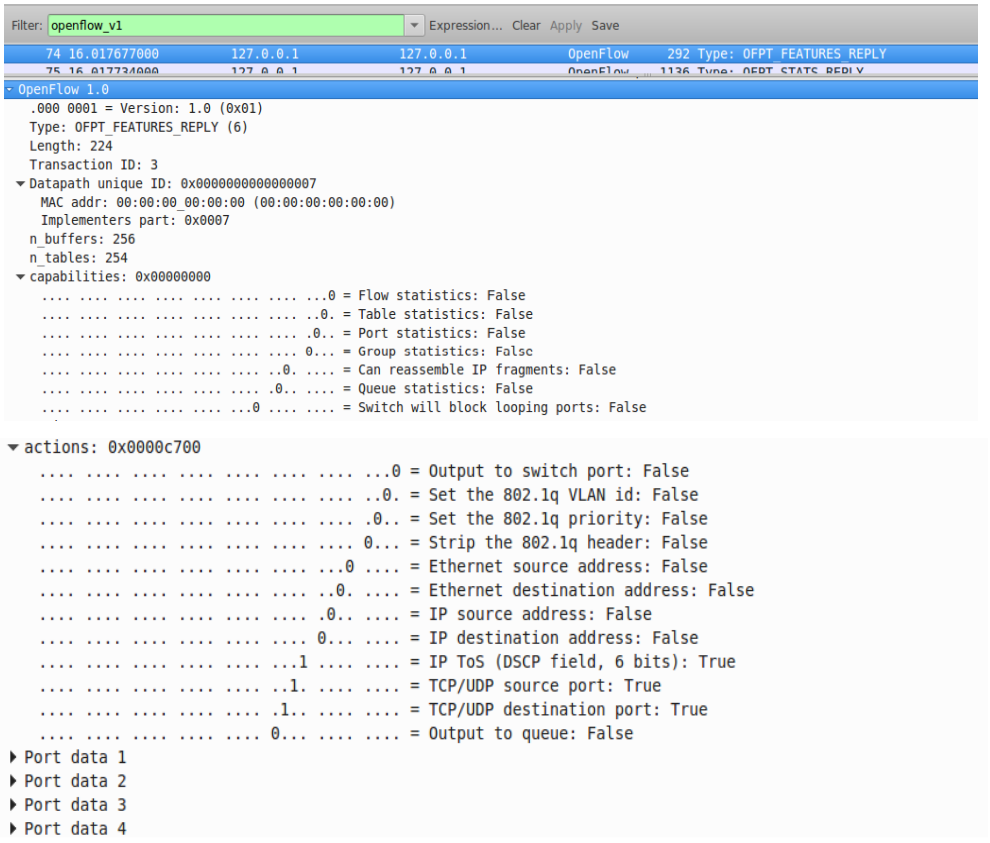

Ilustración 30: Paquete Features_Reply [Fuente: Propia]

3.3. Open vSwitch

3.3.1. Introducción

Abreviado OVS, es un software de código abierto, diseñado para ser utilizado como un switch virtual en entornos de servidores virtualizados. Es el encargado de reenviar el tráfico entre diferentes máquinas virtuales (VMs) en el mismo host físico y también reenviar el tráfico entre las máquinas virtuales y la red física. Está bajo licencia Apache 2.0.

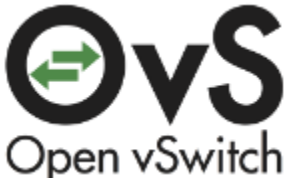

Ilustración 31: Logo Open vSwitch [Fuente: openvswitch.org]

Open vSwitch es un software multicapa para switches, cuyo objetivo es la implementación de una plataforma de calidad que soporta interfaces de gestión estándar y exponga las funciones de forwarding de forma programable. Podemos decir, que Open vSwitch es una de las implementaciones más populares de OpenFlow. Está bien adaptado para funcionar como un switch virtual en ambientes implementados con máquinas virtuales. Además de exponer interfaces estándar de control y visibilidad con la capa de red virtual, fue diseñado para soportar una distribución a través de múltiples servidores físicos. Open vSwitch soporta numerosas tecnologías de virtualización basadas en Linux.

3.3.2. Características

La versión actual de Open vSwitch es compatible con las siguientes características [[18]]:

- Modelo estándar de VLAN 802.1Q con puertos troncales y de acceso.
- **NetFLow** y **sFlow**: posibilidad de sacar netfllow (paquetes del protocolo NEtFlow que sirven para capturar información sobre el tráfico IP) y sFlow (paquetes del protocolo sFlow diseñado para monitorización de interfaces, dispositivos Wireless y del equipo anfitrión) entre máquinas virtuales, que pueden ser capturados y analizados. Detectar cualquier anomalía de la red, malware, etc.
- **OpenFlow** 1.0 o extensiones posteriores.
- **SPAN** (Switch port Analyzer) y **RSPAN** (Remote SPAN): Nos permite hacer mirroring de un puerto o incluso de un bridge entero, es decir, enviar una copiar de los paquetes vistos en un puerto del switch (o VLAN entera) a una conexión para el monitoreo de red en un puerto del switch y de esta forma poder llevar a cabo el diagnóstico de errores en la red.
- **LACP**, **Bonding** (IEE 802.1 AX-2008): permite balancear el tráfico entre varios enlaces, LACP es a nivel de switch exterior (trunking) y el bonding a nivel de máquina virtual.
- **STP**, del inglés Spanning Tree Protocol (IEEE 802.1D-1998): es un protocolo de red de nivel 2 del modelo OSI. Su función es la gestión de bucles en topologías de red. Los bucles

[18] Open vSwitch, Disponible en: http://openvswitch.org/

ocurren cuando hay rutas alternativas hacia un mismo destino, necesarias para proporcionar redundancia y ofrecer mayor fiabilidad a la red. Entonces los dispositivos de interconexión de nivel de enlace reenvían indefinidamente las tramas broadcast y multicast, al no existir un campo TTL (tiempo de vida) en las tramas de la capa 2, creando así un bucle infinito que consume el ancho de banda. STP calcula una única ruta libre de bucles entre los dispositivos de la red pero manteniendo los enlaces redundantes desactivados como reserva, con el fin de activarlos en caso de fallo.

- Políticas de tráfico por puerto, firewalling.
- **QoS**: Para definir niveles de calidad de servicio como disponibilidad, ancho de banda, ratio de error, latencia, priorizar tráfico, etc.
- Protocolos de túnel Múltiples (**GRE, VXLAN, IPsec, GRE y VXLAN sobre IPsec**): GRE Generic Routing Encapsulation es un protocolo de túnel desarrollado por Cisco Systems que puede encapsular una variedad de protocolos de capa de red dentro de enlaces virtuales punto a punto a través del protocolo IP.
- Soporte para **Bidirectional Forwarding Detection** (BFD)
- Soporte para **HFSC qdisc**: es un algoritmo que permite priorizar el tráfico en una red basándose en QoS (calidad de servicio) y CBQ (Class-based quequing o Encolamiento Basado sobre Clases) que permite que el tráfico este organizado por clases y así compartir el ancho de banda (shaping).
- Selección de herramientas para el **monitoreo de enlaces 802.1ag CFM**, estándar para redes de puentes virtuales y metropolitanas, que define una serie de protocolos y prácticas para operaciones, administración y mantenimiento para puentes y redes locales (LAN).

3.3.3. Composición de OVS

Los principales componentes son:

- **Ovs-vswitchd:** un demonio que implementa el switch, con un módulo del *kernel* de Linux para conmutación basada en flujos.
- **Ovsdb-server:** un servidor de base de datos ligero que ovs-vswitchd consulta para obtener su configuración.

- **Ovs-brcompatd:** un demonio que permite a ovs-vswitchd actuar como un reemplazo momentáneo del bridge de Linux.
- **Ovs-dpctl:** una herramienta para configurar el módulo del *kernel* del switch.
- **Ovs-vsctl:** una utilidad para consultar y actualizar la configuración de ovsvswitchd.
- **Ovs-appctl:** una utilidad que envía comandos para ejecutar los demonios de Open vSwitch.
- **Ovsdbmonitor:** una herramienta GUI para la visualización remota de las

Open vSwitch también proporciona algunas herramientas:

- **Ovs-controller:** un controlador OpenFlow básico.
- **Ovs-ofctl:** una utilidad para consultar y manejar *switches* y controladores OpenFlow.
- **Ovs-pki:** una utilidad para crear y administrar la clave pública de la infraestructura de *switches* OpenFlow.

Open vSwitch puede ser portado a diferentes sistemas operativos y plataformas de hardware. Además se ejecuta en servidores físicos y soporta la administración remota de una manera que hace que sea más fácil para los desarrolladores de las plataformas de virtualización. Open vSwitch proporciona dos protocolos abiertos que están especialmente diseñados para la gestión remota en ambientes de redes virtualizadas: OpenFlow, que expone estados de reenvío basado en flujos, y el protocolo de gestión de OVSDB, que expone el estado del puerto del switch.

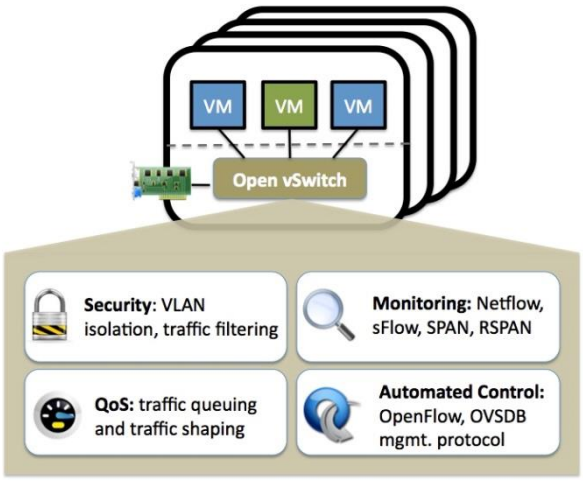

Ilustración 32: Características OVS [Fuente: openvswitch.org]

4. Desarrollo del prototipo

4.1. Introducción

Este trabajo surgió de la necesidad de encontrar alguna solución a algunos problemas que presentan las SDN cuando se las aplica a redes de pequeña escala o para uso didáctico. Esta problemática ya fue expuesta al comienzo de este trabajo.

Se optó por utilizar tecnología de bajo coste como también software y hardware libre para no tener problemas con licencias y poder realizar modificaciones al proyecto para adaptarlo a las necesidades presente.

4.2. Tecnologías utilizadas

- **Raspberry PI [19]:** Es un ordenador de placa reducida de bajo coste desarrollado en el Reino Unido con el objetivo de fomentar las enseñanzas de las ciencias de la computación en las escuelas.

  En este proyecto se utilizaran dos de estas placas. Para la construcción del switch OpenFlow se utilizara la *Raspberry Pi 2 Model B* que cuenta con estas características:

  - Un CPU ARM Cortex-A7 de cuatro núcleos a 900 MHz.
  - 1GB de memoria RAM.
  - 4 Puertos USB
  - 40 pins GPIO (General Purpose Imput/Output)
  - Puerto HDMI
  - Puerto Ethernet
  - Jack combinado de audio y video de 3.5mm
  - Interfaz para cámara
  - Interfaz para display
  - Slot para micro SD

[19] Raspberry, Disponible en: https://www.raspberrypi.org/

Ilustración 33: Raspberry Pi 2 Model B [Fuente: www.raspberrypi.org]

La segunda placa utilizada será una *Raspberry Pi Model B,* un modelo más antiguo que el utilizado para construir el switch pero con suficientes prestaciones para poder funcionar como el controlador de nuestra red SDN. Sus características son:

- Un CPU ARM1176JZF-S a 700 MHz.
- 512MB de memoria RAM.
- 2 Puertos USB
- 26 pins GPIO (General Purpose Imput/Output)
- Puerto HDMI
- Puerto Ethernet
- Jack de audio y video separados
- Interfaz para cámara
- Interfaz para display
- Slot para tarjeta de memoria SD

Ilustración 34: Raspberry Pi Model B [Fuente: www.raspberrypi.org]

Estas dos placas serán las partes principales de nuestro proyecto, una funcionando como un conmutador OpenFlow y la otra siendo el cerebro de la red.

- **Adaptadores USB-Ethernet:** Dado que la Raspberry Pi 2 model B solo cuenta con un puerto Ethernet es necesario utilizar adaptadores para poder disponer de más puertos donde conectar los hosts que utilizaremos para las pruebas. Se escogieron adaptadores de marcas distintas, uno es de marca RISUTA y el otro de marca MX7. La razón de que sean de distintos proveedores es que los adaptadores de la misma marca suelen traer las mismas direcciones MAC y no nos sirve para este proyecto.

- **Hosts:** Para la creación de la red se utilizaran dos computadoras como hosts. La primera presenta estas características importantes:
  - Marca: Bangho
  - Procesador: Intel® Core™ I3-2330M CPU @ 2.20 GHz 2.20 GHz
  - Memoria RAM: 6 GB
  - Puertos Ethernet: 1 puerto de 10/100/1000 Mbps
  - Sistema Operativo: Windows 7 Profesional 64bits

  El segundo host posee las siguientes características importantes:

  - Marca: Bangho
  - Procesador: Intel® Core™ I7-4702MQ CPU @ 2.20 GHz x8
  - Memoria RAM: 8 GB
  - Puertos Ethernet: 1 puerto de 10/100/1000 Mbps
  - Sistema Operativo: Ubuntu 14.04 LTS 64bits

- **Controlador:** Para este proyecto se eligió el controlador POX por su facilidad de uso, su rendimiento, la cantidad de bibliografía y documentación que posee y sus complementos adicionales que lo transforman en una buena opción para redes de pequeña escala, además de ser OpenSource.

Ilustración 35: Controlador POX

- **Software Raspberry [20]:** El sistema operativo utilizado para las placas será la distribución *Raspian Jessie* descargada desde la página oficial de Raspberry.

- **Periféricos:** contaremos con mouse, teclados, monitores y demás objetos para poder facilitar el manejo de los dispositivos del proyecto.

También utilizaremos otros elementos para realizar comparaciones entre este proyecto y las redes SDN con switches adquiridos por la facultad.

[20] Raspian Jessie, Disponible en: https://www.raspberrypi.org/downloads/raspbian/

- **Switch OpenFlow:** El switch a utilizar será de la marca *Hewlett-Packard*, más precisamente el switch HP Aruba 2920-24G, que posee estas características:

  - **Puertos:** 20 puertos RJ-45 10/100/1000 con detección automática, 4 puertos RJ-45 10/100/1000 con doble función y 2 ranuras para módulos adicionales de 10 Gbps.

  - **Memoria y Procesador:** Tri Core ARM1176 a 625 MHz con 512 MB de SDRAM con tamaño de buffer para paquetes de 11,25 MB.

  - **Latencia:** 10 Mb< 9 µs, 100 Mb< 3,3 µs y 1 GB< 3,3 µs.

Ilustración 36: Switch HP 2920-24G [Fuente: hp.com]

El proyecto se dividió en 2 partes, la primera consistió en la programación del controlador POX en la Raspberry PI Model B, que servirá como cerebro de nuestra red SDN, y la segunda parte, la más complicada, es la que abarca la programación del switch OpenFlow en la placa Raspberry PI 2 Model B.

4.3. Desarrollo del controlador

Como ya se dijo más arriba, utilizaremos el controlador POX para poder gestionar la red, debido a su baja complejidad, facilidad de uso y los pocos recursos que necesita.

Nos serviremos de un router Mikrotik RB751 para conectarnos a internet y poder descargar los archivos necesarios.

Una vez asegurados la conexión a internet, iniciamos la instalación. Abrimos el terminal y como primer paso colocamos los comandos:

- ***$ Sudo apt-get update***
- ***$ Sudo apt-get upgrade***

Recordando que la distribución que estamos usando para esta Raspberry es la Raspbian que es una distribución de GNU/Linux basada en Debian.

Estos dos comandos no tienen nada de especial, son comúnmente usados en cualquier sistema operativo basado en Linux. El comando *Sudo* se utiliza para brindar permisos de seguridad de otros usuarios (usualmente del usuario *root)* al usuario actual. a*pt-get* es la herramienta que utiliza Debian y sus derivados (Ubuntu incluida), para gestionar los paquetes instalables disponibles en los repositorios. Finalmente llegamos a *update* y *upgrade*, el primero actualiza las listas de paquetes disponibles y sus versiones, pero no instala o actualiza paquetes. Con el segundo lo que hacemos es una actualización de nuestro sistema con todas las posibles actualizaciones que pudiera haber, es decir no sólo actualiza nuestro sistema operativo sino que también las aplicaciones que están contenidas en los repositorios. Esto es solo para tener el sistema operativo totalmente actualizado y listo para empezar con la instalación del programa.

La mejor forma de trabajar con POX es con el repositorio *git*, por lo que vamos a clonar la carpeta a nuestra Raspberry.

- ***$ Sudo git clone http://github.com/noxrepo/pox***

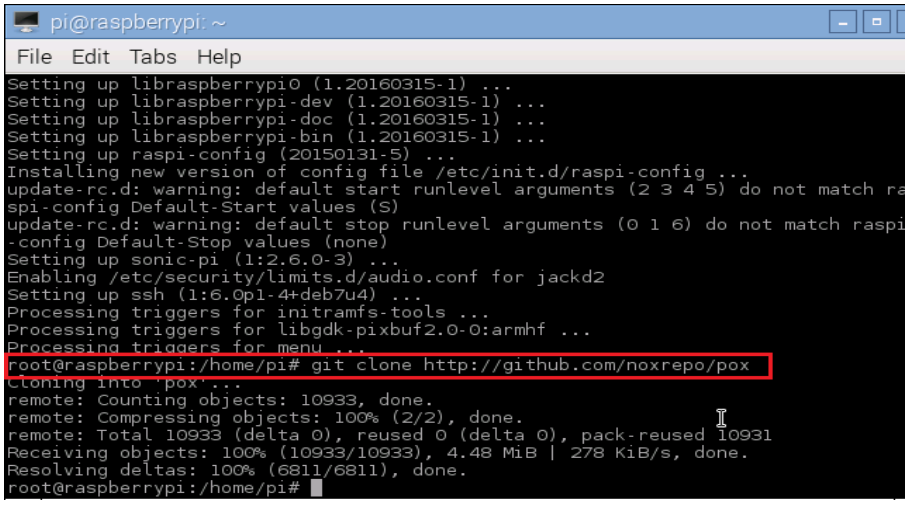

Ilustración 37: Clonación de carpeta en Git [Fuente: Propia]

Pox posee varias ramas o *branch* con las que trabajar, y cada una de ellas esta ordenada alfabéticamente y poseen nombres de peces.

- ***angler (Juio 2011 - Marzo2013)***
- ***betta (Hasta Mayo 2013)***
- ***carp (Hasta Octubre 2013)***

- ***dart (Hasta Julio 2014)***
- ***eel (...)***

Nosotros usaremos la distribución dart que es la rama estable más nueva. Para ello utilizaremos los siguientes comandos.

- ***$ cd pox***
- ***/pox $ git checkout dart***

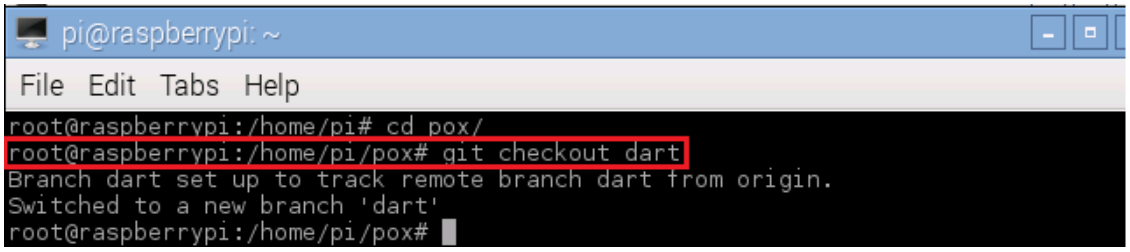

Ilustración 38: Cambio de rama [Fuente: Propia]

Con esto realizado ya podemos trabajar con el controlador POX en nuestra raspberry.

4.3.1. Prueba

Los comandos principales para invocar POX son:

- ***Pox.py***
- ***Debug-pox.py***

El primero corre el controlador bajo circunstancias normales, y el segundo se usa cuando se intentan resolver problemas.

Aparte de estos dos comandos principales, POX cuenta con muchos componentes útiles para distintos tipos de comportamientos. Por ejemplo:

- ***Forwarding.hub:*** Simplemente instala una "wild card" en cada tabla de flujo de cada switch y los convierte en simples hubs Ethernet.
- ***Py:*** Este componente causa que POX inicialice un intérprete Python interactivo que es útil para debugueo y experimentos interactivos.
- ***Openflow.webservice:*** Es un simple servicio web JSON-RPC para interactuar con OpenFlow.

Hay muchos componentes más y se van creando nuevos y modificando antiguos diariamente.

El comando que utilizaremos para nuestro controlador será:

- ***./pox.py forwarding.l2_learning openflow.discovery openflow.spanning_tree –no-flood –hold-down host_tracker info.packet_dump samples.pretty_log log.level –DEBUG***

Donde:

- ***Forwarding.l2_learning:*** Este componente hace que el switch OpenFlow actúe como un learning switch Ethernet común. Aprende la dirección MAC y matchea todos los campos en la cabecera, entonces este instala múltiples flujos en la red para cada para de direcciones MAC. Por ejemplo, diferentes conexiones TCP resultaran en diferentes flujos instalados.
- ***Openflow.discovery:*** Usa mensajes LLDP (Link Layer Discovery Protocol) enviados y recibidos desde el switch OpenFlow para descubrir la topología de la red. También detecta cuando un enlace de la red se levanta o se cae. Esta información es usada por otros componentes.
- ***Openflow.spanning_tree:*** Este componente es requerido en casos donde la topología de la red tiene loops. Este trabajo con el openflow.discovery para construir una visión de la topología de la red y construir un spanning tree deshabilitando flujos en puertos del switch que no están en el árbol. Algunos componentes, como este, toman argumentos para ellos mismos. Estos se colocan seguido del nombre del componente y empezados con dos guiones medios (--).
  - ***--no-flood:*** Deshabilita flujos en todos los puertos tan pronto como el switch es conectado, en algunos puertos son habilitados luego.
  - ***--hold-down:*** Previene alteraciones de control de flujo hasta que un ciclo de descubrimiento completo se complete (y, por lo tanto, todos los enlaces han tenido la oportunidad de ser descubiertos).

  Estos dos componentes son usados para asegurar que ningún paquete sea enviado en la red antes de que el componente cree el spanning tree.

  Responderá a cambios en la topología de la red. Si un enlace se rompe y existe un enlace alternativo, este mantiene la conectividad en la red creando un nuevo árbol que permite flujos en los puertos conectados al enlace alternativo.

  Cuando se usa spanning tree también se usa un componente de direccionamiento que cree flujos que tengan un valor de timeout seteado. En este caso usamos forwarding.l2_learning.

- ***Host_tracker:*** Intenta llevarle la pista a los hosts de la red. Examina mensajes recibidos por POX y aprende direcciones MAC e IP de los hosts de la red. En este ejemplo se basa en paquetes que llegan del controlador. El direccionamiento de paquetes en la red debe ser reactivamente y por eso se necesita un componente como *forwarding.l2_learning*.
- ***Info.packet_dump:*** Este componente mostrara en la consola log información acerca de los paquetes de datos recibidos por POX desde el switch. Ayuda a ver como el switch interactúa con POX sin correr tcpdump.
- ***Log.level:*** Permite al usuario de POX especificar la cantidad de detalles que pueden ver en la información log producida por POX.
    - ***--DEBUG:*** Es el nivel más detallado.
- ***Samples.pretty_log:*** Da formato a los mensajes log para proveer una salida atractiva y fácil de leer de los mensajes log en la consola POX.

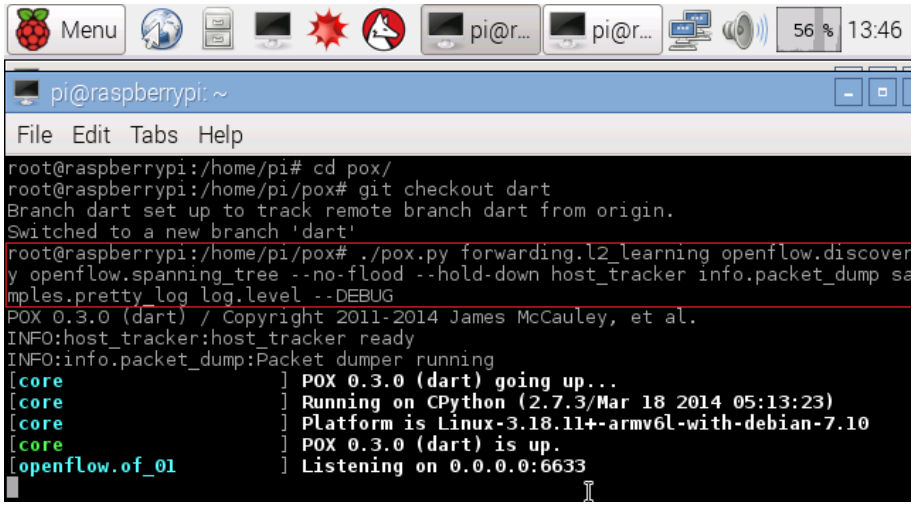

Ilustración 39: Controlador activo [Fuente: Propia]

Con todo esto ya hemos conseguido que el controlador POX funcione dentro de la raspberry pi.

Como vemos en este ejemplo, en donde creamos una red virtual gracias a Mininet y Miniedit, y configuramos los switches virtuales para que se conecten a la raspberry. Vemos como reconoce los tres switches de la topología, y empieza a hacer el descubrimiento de la topología y a evitar los loops con el spanning tree.

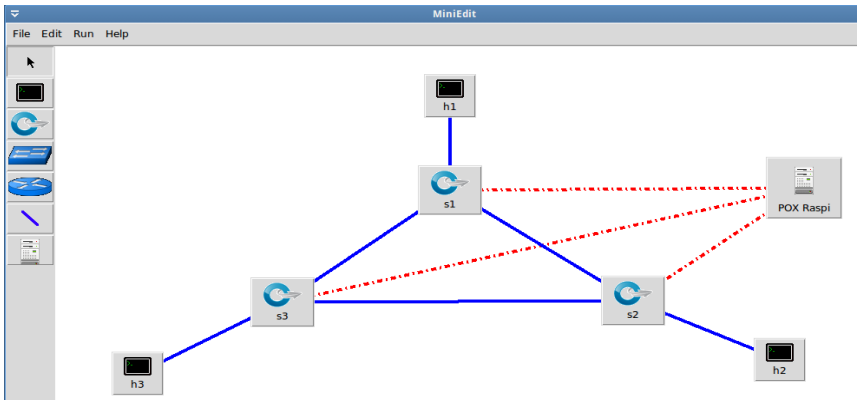

Ilustración 40: Topología en Mininet [Fuente: Propia]

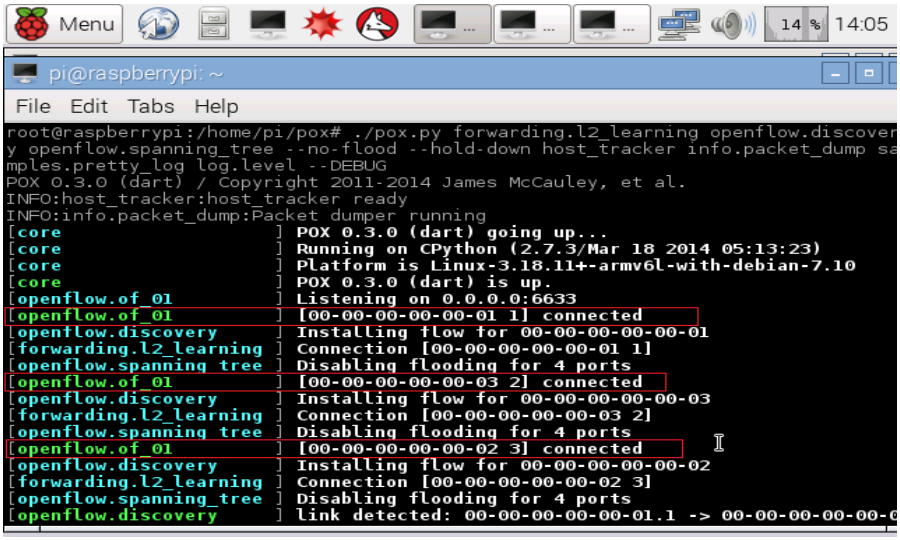

Ilustración 41: Detección de los tres switches [Fuente: Propia]

4.4. Desarrollo del switch

4.4.1. Instalación de OpenvSwitch

Al igual que en el desarrollo del controlador, comenzamos la instalación con estos dos comandos:

- ***$ Sudo apt-get update***
- ***$ Sudo apt-get upgrade***

Una vez actualizado nuestro dispositivo empezamos con programación del switch. Nos decantamos por usar Open vSwitch, sus características fueron explicadas más arriba. Para esto necesitamos instalar los paquetes y dependencias necesarias. Existen dos maneras de poder

realizar la instalación, una es clonando el repositorio Git del OVS y otro es descargándolo de su página oficial, este último será el método utilizado en este proyecto.

- ***$ Sudo wget http://openvswitch.org/releases/openvswitch-2.3.1.tar.gz***
- ***$ Sudo tar -xvzf openvswitch-2.3.1.tar.gz***

"wget" es una herramienta libre que nos permite descargar contenido de servidores web mediante protocolos HTTP, FTP y HTTPS. En este caso descargamos el paquete del servidor web de Open vSwitch. Luego, con el comando "tar" lo que hacemos es desempaquetar todos los archivos contenidos en el archivo descargado. Tener en cuenta que no es un archivo comprimido, sino que es un archivo en donde se han empaquetados muchos archivos para que sea más fácil su transferencia. El comando tar se acompaña de funciones adicionales, en este caso tenemos cuatro funciones.

- **-x:** Extrae archivos de un contenedor.
- **-v:** Modo verbose.
- **-z:** Comprime o descomprime mediante gzip.
- **-f:** Especifica el nombre del contenedor.

Antes de compilar debemos instalar algunas dependencias.

- ***$ Sudo apt-get install python-simplejson python-qt4 libssl-dev python-twisted-conch automake autoconf gcc uml-utilities libtool build-essential pkg-config***

Donde:

- **Python-simplejson:** es una sencilla, rápida, completa y extensible biblioteca para codificar y decodificar elementos JSON para Python.
- **Python-qt4:** es un conjunto de herramientas para crear aplicaciones con interfaz gráfica. Es una mezcla entre Python y la biblioteca Qt.
- **Libssl-dev:** Este paquete es parte de la implementación de protocolos criptográficos SSL y TLS para asegurar comunicaciones por internet del Proyecto OpenSSL.
- **Python-twisted-conch:** Es una implementación cliente/servidor del protocolo SSH usando el framework twisted.
- **Automake y autoconf:** Son herramientas de programación que producen programas "makefiles.ini" portables para el uso de *make* usado en la compilación de software.
- **Gcc:** Es un conjunto de compiladores creados por el proyecto GNU.
- **uml-utilities**: Es una modificación del núcleo Linux para que funcione sobre su propia interfaz de llamadas al sistema. De este modo, un núcleo compilado para la

arquitectura *um* puede operar como un proceso de usuario más de otro núcleo Linux que hace las veces de anfitrión.

- **Libtool:** Es una herramienta de programación GNU proveniente del sistema de construcción para GNU usada para crear bibliotecas de software portables..
- **build-essential:** Se trata de un paquete que contiene una lista informativa de los paquetes que se consideran esenciales para la creación de paquetes *Debian.*
- **pkg-config:** Es un software que provee una interfaz unificada para llamar bibliotecas instaladas cuando se está compilando un programa a partir del código fuente.

Un problema es la elección de los archivos de cabecera de kernel correctos para la compilación del módulo del kernel. Para esta versión de kernel no hay una versión exacta, pero se puede instalar el kernel 3.12.35 que tendrá un buen funcionamiento.

- ***$ Sudo apt-get install linux-headers-3.12-1-rpi***

Una vez realizados estos pasos, se procedió a la compilación de los archivos de Open vSwitch eligiendo la cabecera de kernel recién instalada.

- ***# cd openvswitch -2.3.1***
- ***/openvswitch -2.3.1# bash boot.sh***
- ***/openvswitch -2.3.1# ./configure --with-linux=/lib/ modules /3.12 -1-rpi/build***
- ***/openvswitch -2.3.1# make***
- ***/openvswitch -2.3.1# make install***

Con estos comandos lo que hacemos primero es entrar en la carpeta openvswitch-2.3.1, luego, dentro de la carpeta, iniciamos el *script* boot.sh que viene dentro del paquete de instalación y es el primer paso para la instalación de nuestro switch. Los tres comandos que siguen son esenciales para la instalación de un software desde fuente y son *configure, make* y *make install.*

- ***Configure:*** Este script es el encargado de conseguir que todo esté listo para la instalación del software. Se asegura de que todas las dependencias para el resto del proceso y la instalación del software estén disponibles, y descubre todo lo que se necesita saber sobre el uso de esas dependencias. Los programas de Unix a menudo son escritos en lenguaje C por lo que se va a necesitar un compilador para la

instalación. En estos casos este script se encargara de averiguar si el sistema operativo posee un compilador de C, averigua como se llama y donde se encuentra.

- ***Make:*** Una vez que *configure* hizo su trabajo podemos invocar *make* para construir el software, esto hace correr una serie de tareas descriptas en el archivo *makefiles* para terminar de construir el software desde su código fuente.
- ***Make install:*** Con los dos comandos anteriores el software ya está construido y listo para correr, este comando lo que hará será copiar el programa construido, sus librerías y documentación a su locación correcta. Esto quiere decir que el binario del programa será copiado a la carpeta */PATH*, el manual y la documentación será copiado a la carpeta */MANPATH,* y todos los otros archivos de los cuales dependa serán guardados en sus correspondientes carpetas.

Una vez completado estos pasos lo que se necesita es cargar el módulo de kernel y dar de alta los archivos de configuración. Esto tendría que hacerse cada vez que la raspberry se enciende pero se configuro para que se cargue el modulo cada vez que se enciende la raspberry automáticamente.

- ***/openvswitch -2.3.1# cp datapath/linux/openvswitch.ko /lib/module/3.12 -1-rpi/***
- ***/openvswitch -2.3.1# depmode -a***
- ***/openvswitch -2.3.1# modprobe openvswitch***
- ***/openvswitch -2.3.1# echo "openvswitch" >> /etc/modules***

Para finalizar la instalación dos archivos de configuración deben ser creados, uno es el archivo de configuración para el *deamon* del vSwitch (*ovs-vswitchd*) y el otro es la base de datos que contendrá la configuración del switch (*ovsdb-server*), por ejemplo que interfaz está conectado al switch virtual.

- ***/openvswitch -2.3.1# touch /usr/local/etc/ovs-vswitchd.conf***
- ***/openvswitch -2.3.1# mkdir –p /usr/local/etc/openvswitch***
- ***/openvswitch-2.3.1# ovsdb-tool create /usr/local/etc/openvswitch/config.db \ vswitchd/vswitch.ovsschema***

Al principio estos archivos no contienen nada, pero se van a ir llenando automáticamente y su contenido va a poder ser alterado gracias a las herramientas de línea de comando que presenta OpenvSwitch.

Creamos un *script* para que inicie el *deamon* dentro de nuestra Raspberry más fácilmente y con un solo comando.

Con todo esto nuestra instalación está completa y podemos verificarlo ingresando el *script* escrito antes.

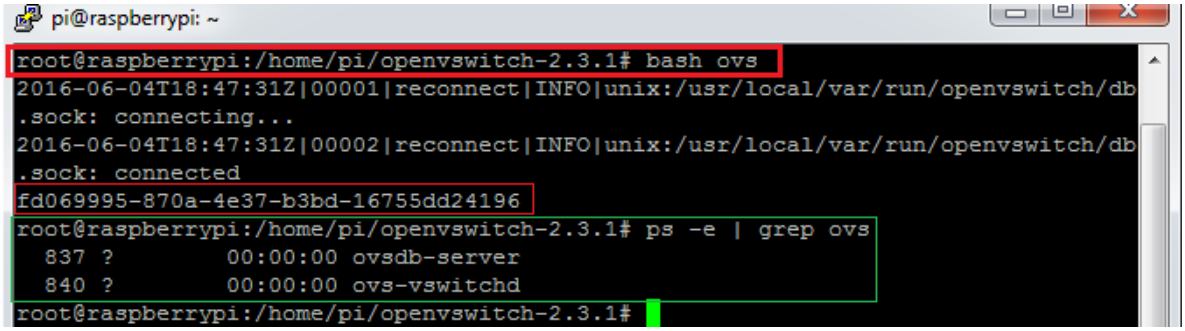

Ilustración 42: Open vSwitch activo [Fuente: Propia]

Como podemos ver, al ingresar nuestro *script* (en nuestro caso se llama "ovs") vemos que se conecta a la base de datos y que, también marcado en rojo, no devuelve el ID del switch virtual. Más abajo, con el comando "ps" que muestra todos los procesos activos, podemos apreciar cómo están en funcionamiento los dos procesos, tanto la base de datos como el switch.

4.4.2. Adaptador USB-Ethernet

Nuestra Raspberry solo cuenta con un puerto Ethernet, para utilizarlo como un switch OF es necesario que cuente con, por lo menos, tres puertos, uno para conectar el controlador, que será el propio de la placa, y otros dos para conectar los hosts. Para esto utilizaremos dos adaptadores USB-Ethernet conectados a un Hub USB por comodidad.

Ilustración 43: Adaptadores USB-Ethernet [Fuente: Propia]

Como estos adaptadores son externos a la Raspberry, debemos de configurar la placa para que los reconozca y podamos trabajar con ellos. Lo que se hace es crear dos reglas para que la placa los reconozca, como si fueran drivers.

Estas reglas son creadas en el archivo */etc/udev/rules.d/70-persistent-net.rules*

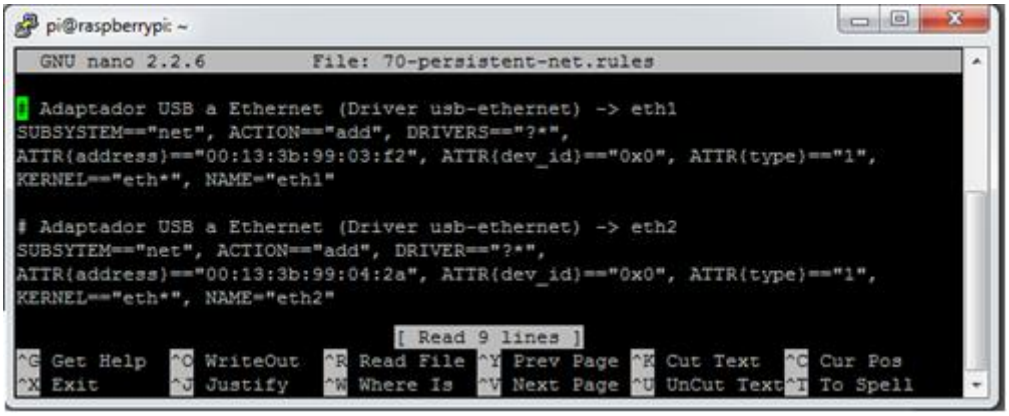

Ilustración 44: Drivers para adaptadores [Fuente: Propia]

4.4.3. Configuración del switch

Ahora hay que configurar el switch con los comandos de Open vSwitch. El comando principal será *ovs-vsctl* que es el más importante en OVS para la configuración y gestión del switch.

En general, el concepto para levantar el switch virtual es crear un *bridge* entre todas las interfaces de un dispositivo que están participando en la gestión de la red OF. El comportamiento de este *bridge* es entonces determinado por las reglas de flujo dadas.

- ***/openvswitch-2.3.1# ovs-vsctl add-br br0***
- ***/openvswitch-2.3.1# ovs-vsctl add-port br0 eth1***
- ***/openvswitch-2.3.1# ovs-vsctl add-port br0 eth2***

Como primer paso se crea el *bridge* virtual, generando automáticamente una interfaz de *bridge* virtual con el nombre “br0”. Y luego se agregan las dos interfaces necesarias “eth1” y “eth2”. El interfaz “eth0” no se agrega porque va a quedar aislado del resto de la red porque será nuestro canal seguro, por donde conectaremos el switch con el controlador.

Con esto levantamos las tres interfaces, le configuramos una dirección IP al *bridge* y dejamos las otras interfaces sin configurar, solo las damos de alta.

- ***/openvswitch-2.3.1# ifconfig br0 192.168.1.11 netmask 255.255.255.0 up***
- ***/openvswitch-2.3.1# ifconfig eth0 up***
- ***/openvswitch-2.3.1# ifconfig eth1 up***
- ***/openvswitch-2.3.1# ifconfig eth2 up***

Le asignamos al *bridge* la dirección IP 192.168.1.11, y configuramos las otras interfaces de acuerdo a nuestra red. Podemos hacer permanente (y lo haremos) a esta configuración modificando el archivo */etc/network/interfaces.*

- ***/openvswitch-2.3.1# nano /etc/network/interfaces***

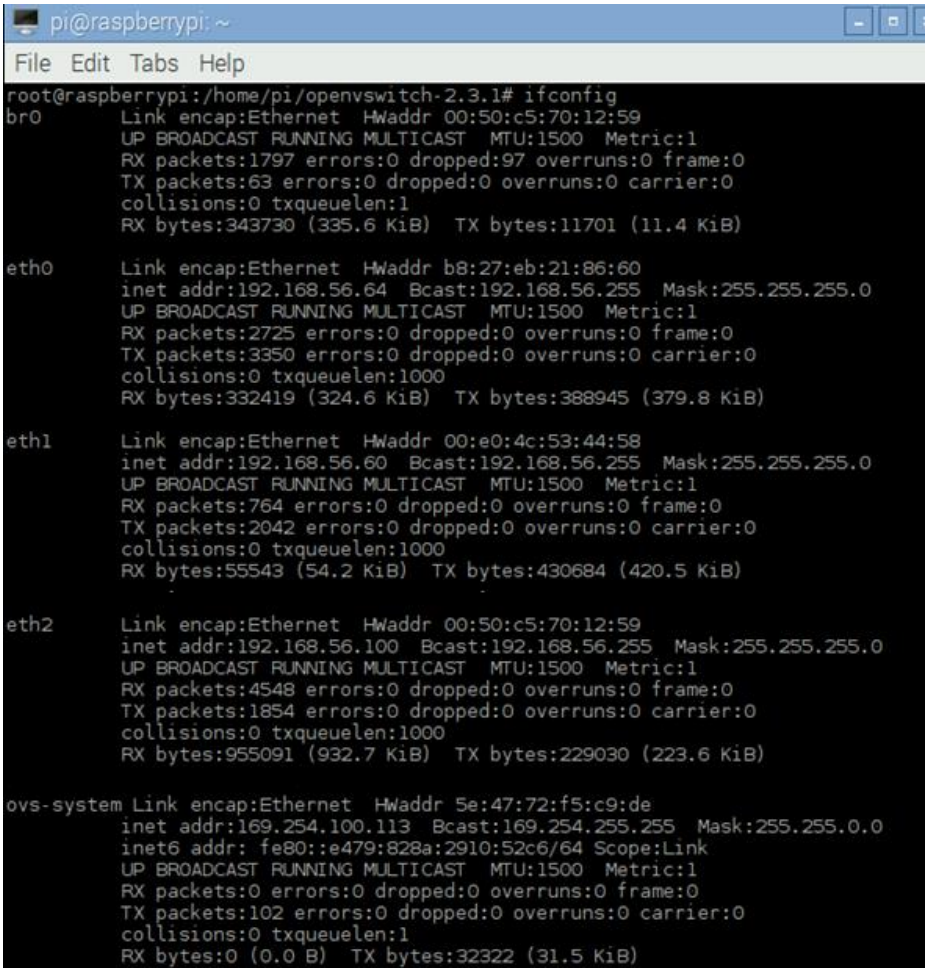

Ilustración 45: Interfaces [Fuente: Propia]

Finalmente, dos configuraciones son necesarias para terminar.

- ***/openvswitch-2.3.1#ovs-vsctl set-controller br0 192.168.56.68/24***
- ***/openvswitch-2.3.1#ovs-vsctl set-fail-mode br0 secure***

El primer comando le dice al switch que el controlador se ubicara en la dirección IP "192.168.56.68". El segundo comando está referido a un mecanismo "fallback" dentro de OpenvSwitch. Este mecanismo de seguridad tiene dos opciones.

- **Standalone:** Cuando el switch no oye al controlador después de tres intentos de reconexión, este mecanismo configura el *bridge* para que el switch funcione de forma tradicional, como un MAC-Learning switch, y así no cortar el tráfico de la red hasta que la conexión sea restablecida.
- **Secure:** Este es el mecanismo que vamos a usar, si el switch no detecta al controlador en tres intentos de reconexión seguirá utilizando las reglas de flujo antes puestas hasta que llegue a su máximo tiempo de vida y sean eliminadas. Una vez borradas las reglas de flujo, el switch no tendrá posibilidades de realizar direccionamiento de paquetes hasta que un controlador sea conectado.

Una vez que la configuración este completa, podemos ver la configuración de nuestro switch con el comando.

- ***#ovs-vsctl show***

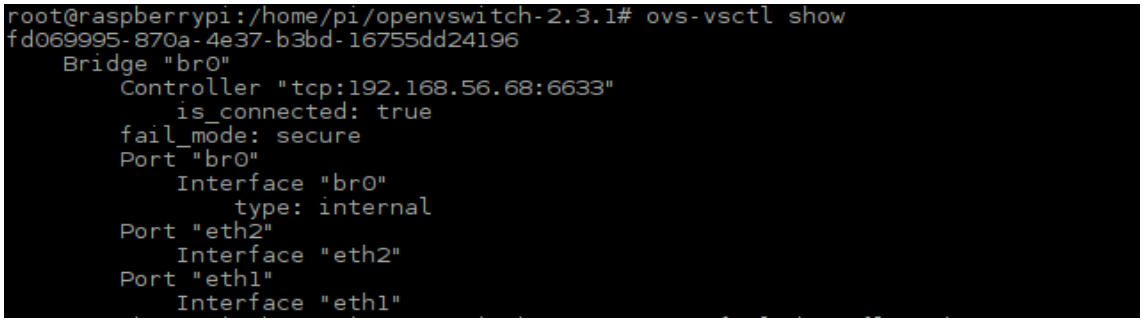

Ilustración 46: Información del switch [Fuente: Propia]

Donde podemos ver el Datapath ID, el nombre del bridge, la dirección IP del controlador, la configuración fallback, las interfaces agregadas y automáticamente creadas, y la interfaz virtual de *bridge* interna.

Otra de las herramientas que nos ofrece OpenvSwitch es el comando *ovs-ofctl*, este comando nos ayuda para el monitoreo y la administración del switch OF. Si lo combinamos con distintos argumentos podemos obtener distintas características del switch.

- ***#ovs-ofctl dump-ports br0***

```
root@raspberrypi:/home/pi/openvswitch-2.3.1# ovs-ofctl dump-ports br0
OFPST_PORT reply (xid=0x2): 3 ports
  port LOCAL: rx pkts=1853, bytes=353823, drop=97, errs=0, frame=0, over=0, crc=
0
           tx pkts=63, bytes=11701, drop=0, errs=0, coll=0
  port  1: rx pkts=1012, bytes=71060, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=2354, bytes=460192, drop=0, errs=0, coll=0
  port  2: rx pkts=4836, bytes=978299, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=2126, bytes=249651, drop=0, errs=0, coll=0
root@raspberrypi:/home/pi/openvswitch-2.3.1#
```

Ilustración 47: Puertos [Fuente: Propia]

Con este comando podemos observar propiedades de los puertos de nuestro switch. Se puede observar el local port, y los puertos 1 y 2 y la cantidad de paquetes, bytes, etc.

- ***#ovs-ofctl dump-flows br0***

```
root@raspberrypi:/home/pi/openvswitch-2.3.1# ovs-ofctl dump-flows br0
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=12.483s, table=0, n_packets=12, n_bytes=888, idle_timeout=
10, hard_timeout=30, idle_age=1, priority=65535,icmp,in_port=2,vlan_tci=0x0000,d
l_src=00:90:f5:c5:12:e3,dl_dst=1c:87:2c:45:9d:13,nw_src=192.168.56.70,nw_dst=192
.168.56.110,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:1
 cookie=0x0, duration=12.474s, table=0, n_packets=12, n_bytes=888, idle_timeout=
10, hard_timeout=30, idle_age=1, priority=65535,icmp,in_port=1,vlan_tci=0x0000,d
l_src=1c:87:2c:45:9d:13,dl_dst=00:90:f5:c5:12:e3,nw_src=192.168.56.110,nw_dst=19
2.168.56.70,nw_tos=0,icmp_type=0,icmp_code=0 actions=output:2
root@raspberrypi:/home/pi/openvswitch-2.3.1#
```

Ilustración 48: Flujos [Fuente: Propia] [Fuente: Propia]

Con este comando se nos muestran los flujos que posee el switch en ese momento, en este caso, como estabamos realizando un “ping” de una pc a otra se pueden ver los dos flujos, uno para el mensaje ICMP de ida y uno para el mensaje ICMP de regreso.

Se pueden ver tambien caracteristicas de los flujos como el tiempo de vida, puertos de entrada, direcciones MAC, acciones, prioridad, etc.

- ***#ovs-ofctl dump-desc br0***

```
root@raspberrypi:/home/pi/openvswitch-2.3.1# ovs-ofctl dump-desc br0
OFPST_DESC reply (xid=0x2):
Manufacturer: Nicira, Inc.
Hardware: Open vSwitch
Software: 2.3.1
Serial Num: None
DP Description: None
root@raspberrypi:/home/pi/openvswitch-2.3.1#
```

Ilustración 49: Descripción del switch [Fuente: Propia]

Con este comando podemos observar la descripción de nuestro switch OF. Vemos que el hardware es un Open vSwitch y que su fabricante es Nicira, Inc.

- ***#ovs-ofctl show br0***

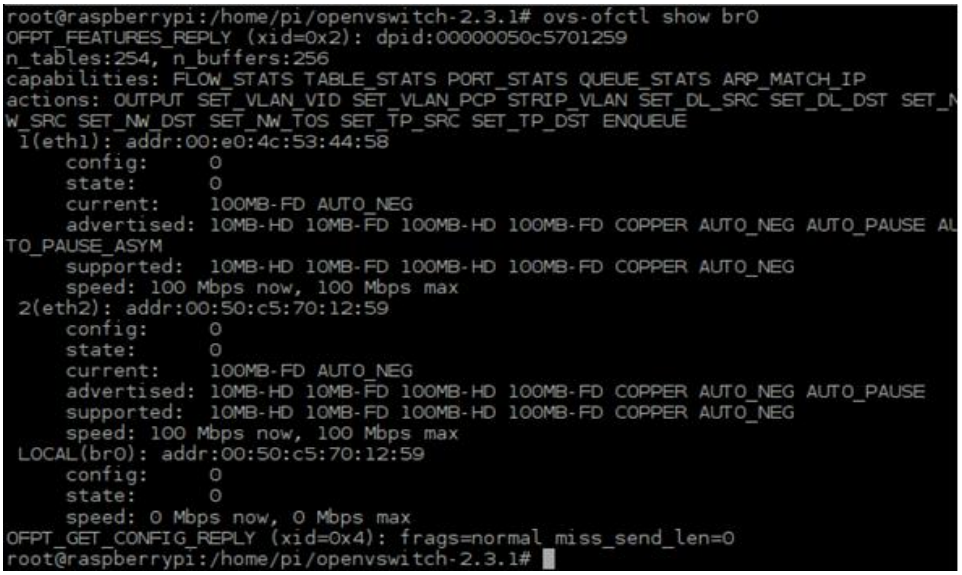

Ilustración 50: Información detallada del switch [Fuente: Propia]

Acá se puede ver de manera más detallada información de los puertos del switch, velocidades, estados del puerto, etc.

Una última configuración necesaria es la del puerto que conectara el switch con el controlador, este puerto no es OpenFlow y estará fuera de la red.

Para que haya conexión debemos agregar una ruta al switch que indique como se deben conectar.

- ***# ip route add 192.168.56.68/32 dev eth0***

Ahora podrán habrá comunicación entre las dos interfaces y vemos como se agregó la ruta al switch

- ***# route –n***

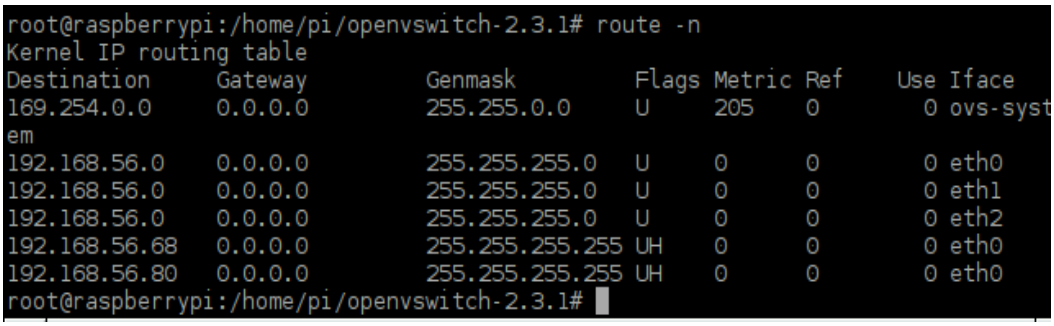

Ilustración 51: Rutas del switch [Fuente: Propia]

Con esto terminamos la configuración y ya podemos armar la red de nuestra prueba que quedara de la siguiente manera.

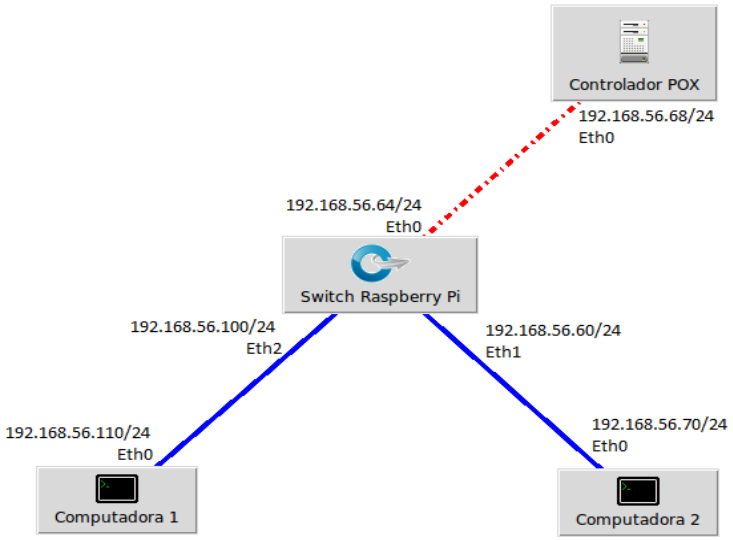

Ilustración 52: Topología del prototipo [Fuente: Propia]

4.5. Diagramación de la red completa

Ya tenemos el controlador instalado en una Raspberry y el switch funcionando en la otra placa, ya estamos en condiciones de diagramar la red completa para nuestras pruebas.

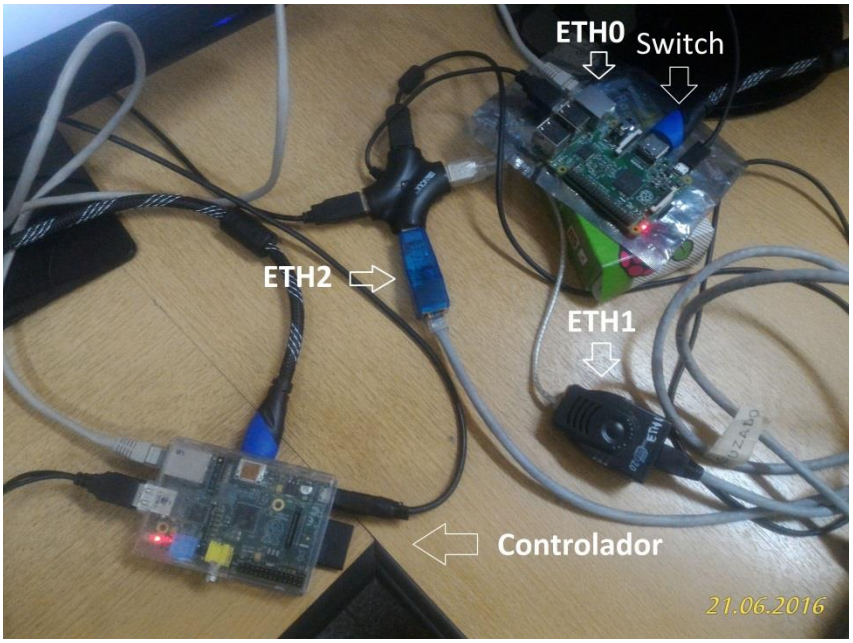

Ilustración 53: Prototipo armado [Fuente: Propia]

5. Pruebas y comparación

5.1. Introducción

El último objetivo de este proyecto es el de realizar pruebas sobre nuestro prototipo para poder medir sus capacidades y compararlas con la de los switch propietarios. Los dos parámetros que analizaremos serán el throughput del switch y el delay de los paquetes. Para realizar estas mediciones utilizaremos el software "Iperf" [21] que es Open-Source y funciona bien tanto en distribuciones de Windows como Linux.

5.2. Iperf

Es una herramienta que se utiliza para hacer pruebas en redes informáticas. Su funcionamiento consiste en crear y enviar flujo de paquetes TCP y/o UDP desde un cliente hacia un servidor y viceversa, y medir el rendimiento de la red. Fue desarrollado por el DAST (Distributed Applications Support Team) y está escrito en C++.

Una de sus características más sobresaliente es la de brindar la posibilidad al usuario de poder ajustar varios parámetros que pueden ser usados para hacer pruebas en una red, o para optimizar y ajustar la red. Se necesitan 2 hosts, uno actuara como servidor, el cual escuchara en cierto puerto (normalmente el 5001) y se quedara a la espera de paquetes enviados por el cliente, este último es el encargado de generar paquetes con dirección al servidor. Al ser de código abierto podemos analizar su metodología de funcionamiento.

Iperf posee una versión con GUI llamada Jperf, esa será la que utilizaremos en estas pruebas.

[21] Iperf, Disponible en: https://iperf.fr/

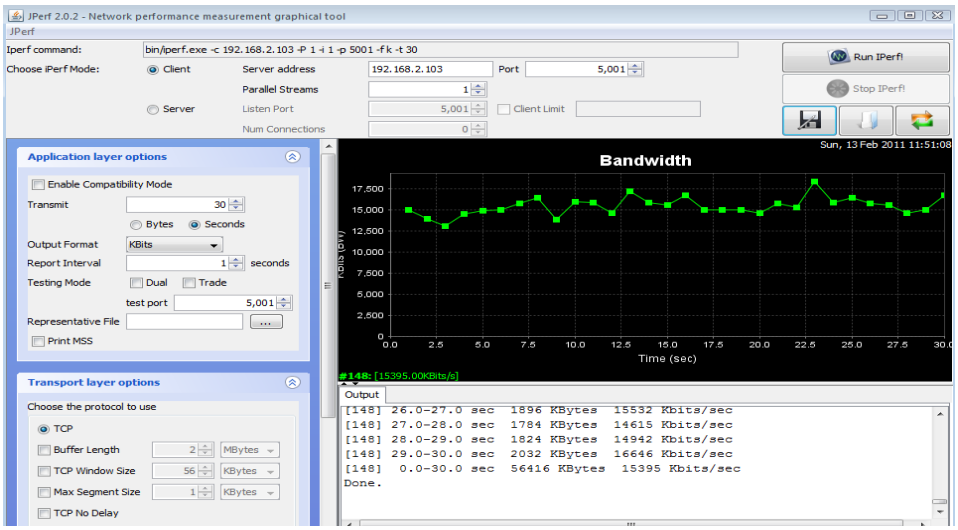

Ilustración 54: GUI Jperf [Fuente: Propia]

5.3. Diagrama de la prueba

Para asegurar que el único cuello de botella de la red sea el switch se utilizaron dispositivos de red que soporten la misma cantidad de datos, para tener una homogeneidad dentro de toda la red.

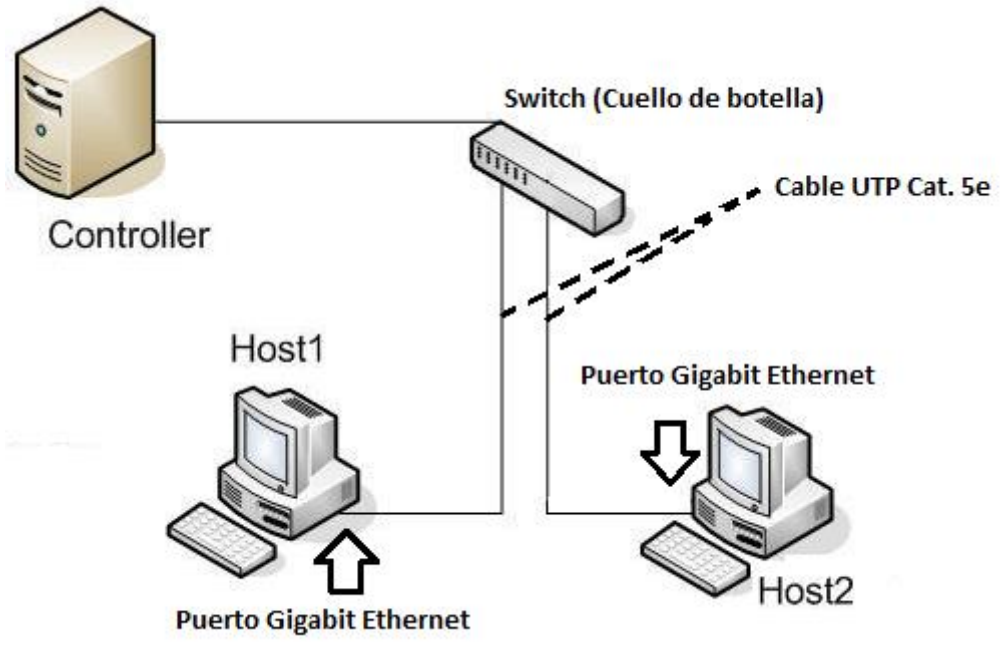

Ilustración 55: Topología de la prueba [Fuente: Propia]

Al saber la cantidad de datos que puede manejar la red podemos asegurarnos que el único limitante va a ser la velocidad de procesamiento que posee nuestro switch.

Las pruebas consistieron en inundar la red con paquetes UDP, utilizando un host como cliente (generador de los paquetes) y otro como servidor (Receptor de los paquetes). Luego Jperf nos entregó los resultados. Se eligió UDP por encima de TCP porque no posee control de flujo, por ende el cliente mandara datos al servidor solo teniendo en cuenta el ancho de banda especificado por el usuario. Al no haber control de flujo Iperf permite a la red desechar paquetes, por lo tanto, el throughput reportado por el servidor será el número de paquetes que circularon por la red sin ser desechados y es la actual velocidad de transferencia de la red.

Se tomaron varias pruebas, donde se enviaba un flujo de paquetes UDP por 30 segundos, en las cuales se fueron cambiando parámetros para ver como reaccionaban los switches, tanto el creado por nosotros como el Hewlett Packard.

5.4. Resultados

5.4.1. 10 Mbits de ancho de banda

Raspberry

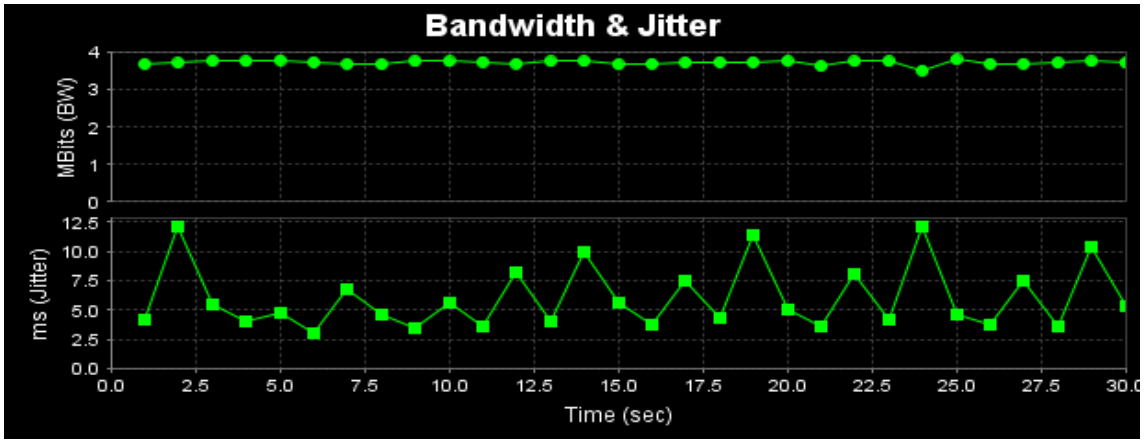

Ilustración 56: Paquetes en servidor a 10 Mbits (Raspberry) [Fuente: Propia]

Switch HP

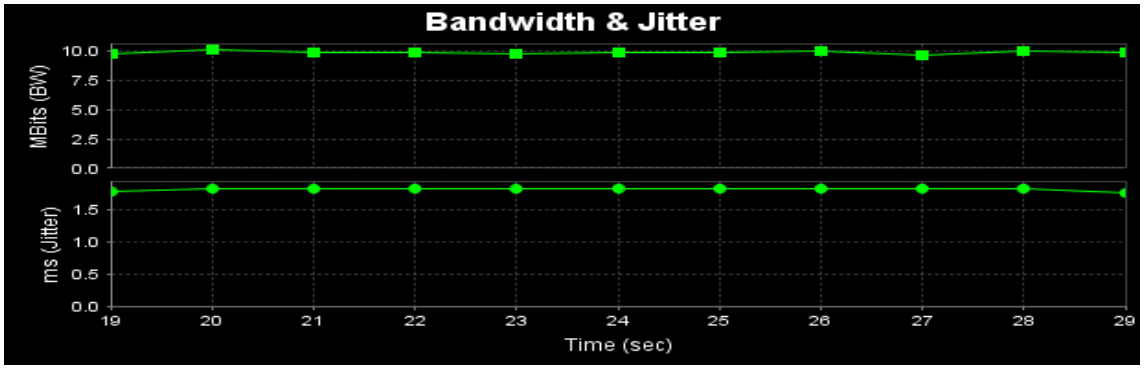

Ilustración 57: Paquetes en servidor a 10 Mbits (Switch HP) [Fuente: Propia]

5.4.2. 100 Mbits de ancho de banda

Raspberry

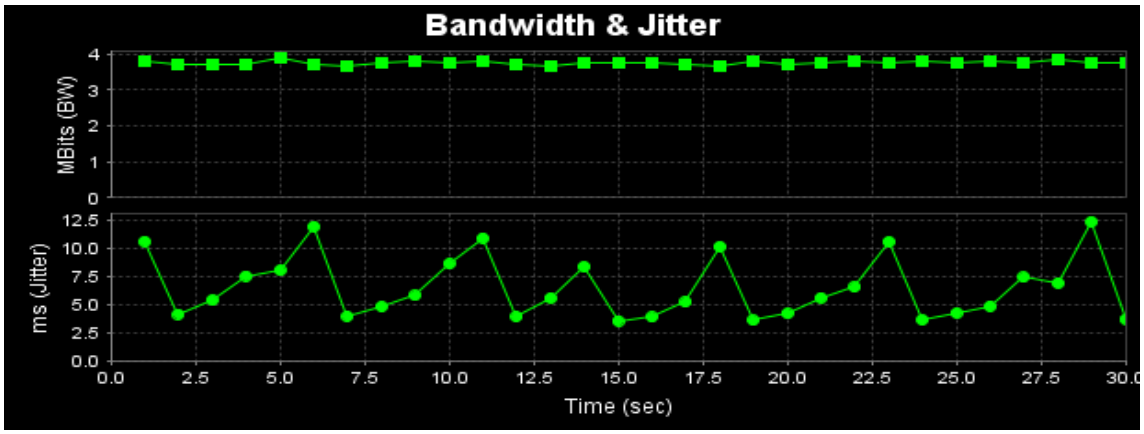

Ilustración 58: Paquetes en servidor a 100 Mbits (Raspberry) [Fuente: Propia]

Switch HP

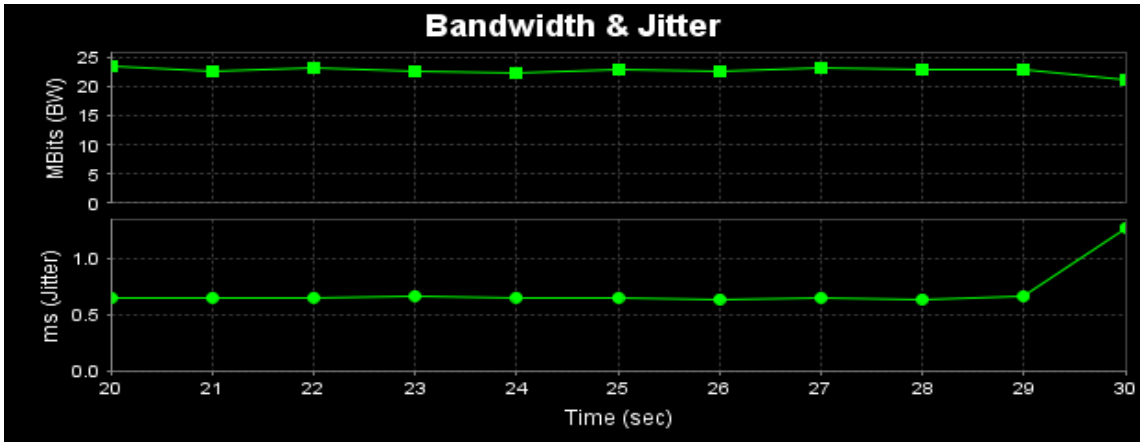

Ilustración 59: Paquetes en servidor a 100 Mbits (Switch HP) [Fuente: Propia]

5.4.3. 1000Mbits de ancho de banda

Raspberry

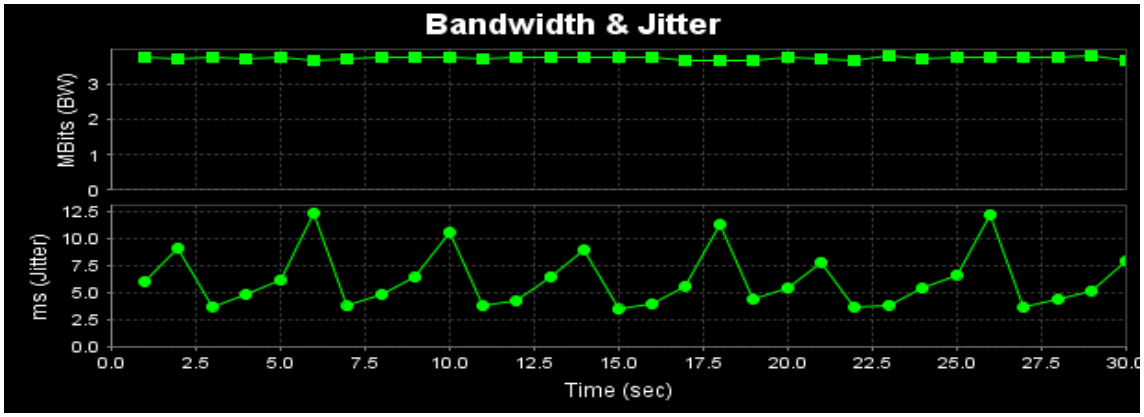

Ilustración 60: Paquetes en servidor a 1000 Mbits (Raspberry) [Fuente: Propia]

Switch HP

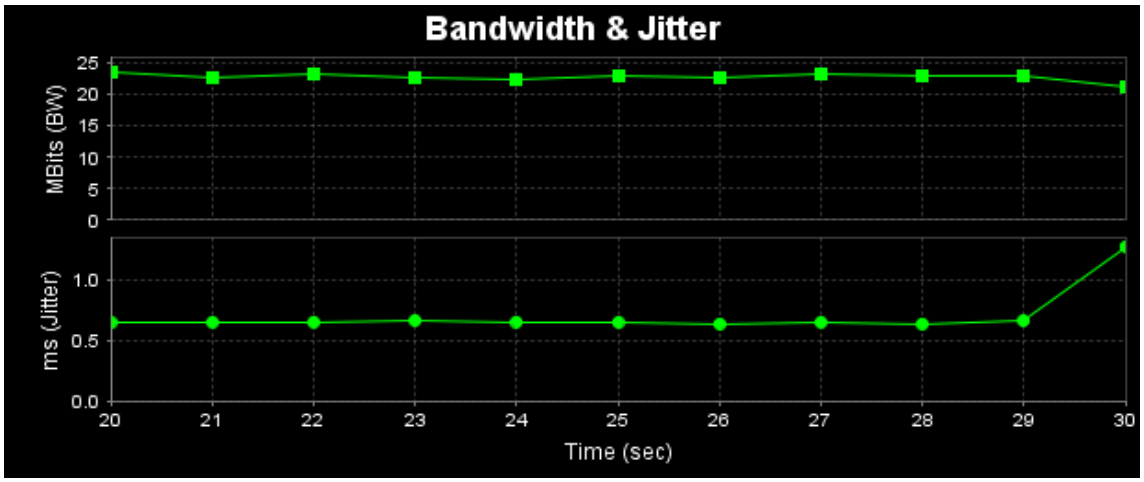

Ilustración 61: Paquetes en servidor a 1000 Mbits (Switch HP) [Fuente: Propia]

5.4.4. Datos en conjunto

	Prueba	Intervalo (seg)	Transferencia (Mbytes)	Ancho de banda (Mbps)	Jitter (ms)	Perdidos/Total (%)
Raspberry	Servidor a 10 Mbits/seg	0-30,2	13,4	3,73	7,845	55/9636 (0,57%)
Switch HP	Servidor a 10 Mbits/seg	0-30	35,2	9,85	0,689	394/25512 (1,5%)
Raspberry	Servidor a 100 Mbits/seg	0-30,2	13,5	3,75	7,917	6762/9661 (70%)
Switch HP	Servidor a 100 Mbits/seg	0-30,2	76	21,1	0,698	202205/256358 (79%)
Raspberry	Servidor a 1000 Mbits/seg	0-30,1	13,4	3,74	7,031	9407/9599 (98%)
Switch HP	Servidor a 1000 Mbits/seg	0-30,2	72,2	20	0,696	2014185/2065212 (98%)

Tabla 2: Datos de pruebas [Fuente: Propia]

Los resultados arrojaron valores muy bajos de ancho de banda, ese es el mayor problema del prototipo. Esto puede tener muchos orígenes que se analizaran más adelante.

Los resultados de Jitter y de paquetes perdidos se encuentran entre los valores normales para las aplicaciones que nosotros queremos imponerle al nuestro proyecto. Si se lo utilizara para VoIP habría que buscar una forma de optimizar estos dos valores.

6. Conclusiones

Luego de las pruebas realizadas al prototipo, y su posterior comparación con los resultados obtenidos de los switches HP se puede concluir que:

- Como era de esperarse, los switches HP poseen un mayor desempeño, ya que son dispositivos diseñados y fabricados para cumplir esa función, en cambio la placa Raspberry, además de correr el software del switch virtual, corre otros procesos y aplicaciones necesarias para su funcionamiento principal que es el de ser un mini ordenador.
- Las mediciones realizadas en la red diagramada en nuestro proyecto no arrojaron los resultados esperados. El mayor problema fue el ancho de banda de nuestra red, es un valor bajo, en el cual uno no experimenta problemas de conexión pero sus defectos aparecen a la hora del intercambio de archivos que es muy lento. Al analizar cuáles pueden ser los factores por los que ocurre esto nos encontramos con distintos cuellos de botella:
    - ***USB 2.0:*** La Raspberry Pi 2 Modelo B es casi idéntica a la Raspberry Pi Modelo B+, solo posee el doble de memoria RAM y un incremento en su performance pero no posee ningún cambio en su arquitectura. Esto significa que desde la primera aparición de las placas Raspberry se sigue utilizando el microchip LAN 9514 como controlador para el puerto Ethernet y los puertos USB. Este controlador posee un hub USB 2.0 de alta velocidad donde tiene integrado cuatro puertos USB físicos y un puerto 10/100 Ethernet por donde pasan todos los datos. En otras palabras, todos los datos, ya sea que entren por el puerto Ethernet o por los puertos USB atravesaran el mismo hub, por lo que en ese momento tendremos un cuello de botella ya que nosotros estamos utilizando el puerto Ethernet para el controlador y dos de los puertos USB para las interfaces del switch. USB 2.0 es limitado a 480 Mbps como tasa de transferencia teórica, pero en realidad su tasa de transferencia real es de 280 Mbps en promedio. Este valor debe ser repartido por los datos que entran por el puerto Ethernet y por los 4 puertos USB.

- ***10/100 Ethernet:*** El microchip LAN 9514 contiene un interfaz 10/100 USB Ethernet el cual, en circunstancias ideales, podría transferir datos a 100 Mbps, pero en condiciones reales la prueba con Iperf arrojo que transmite datos a una velocidad de 36 Mbps, 36% de la velocidad teórica. Esto puede variar de acuerdo a muchas cosas como ser la velocidad y el uso del procesador en ese momento, el uso del disco, el tipo de memoria que posee la placa, etc.
- ***Adaptadores USB a Ethernet:*** Al tener tantas adaptaciones en la red, tanto la de los adaptadores externos que se utilizan para las interfaces del switch como el del hub USB Ethernet imbuido en la Raspberry, se pierde mucho tiempo en procesado de paquetes que disminuyen el rendimiento de la red.
- ***Tarjeta de memoria:*** Este es otro limite que tiene la placa, ya que estas tarjetas de memorias poseen cierto límite en su velocidad de escritura y lectura y como la aplicación de switch virtual que estamos corriendo se encuentra dentro de esta micro SD, estamos atados a la velocidad con la cual trabaje.
- ***Procesador y memoria:*** Como se dijo más arriba, un switch es un dispositivo que está especialmente diseñado para direccionar paquetes, con todo lo que eso conlleva, mientras que la Raspberry es un mini ordenador que, además de correr la aplicación que la convierte en un switch OpenFlow, también está corriendo infinidad de aplicaciones extra para su normal funcionamiento. Estos procesos y aplicaciones corriendo en segundo plano consumen tanto memoria RAM como rendimiento del procesador central de la placa. Todo esto repercute directamente en el procesamiento de paquetes requerido por nuestro switch.
- ***Controlador en un hardware de menor rendimiento:*** Se utilizó una placa Raspberry para el controlador de características inferiores a la que actúa como switch, esto afecta el rendimiento de la red ya que el controlador se procesa las peticiones a un tiempo menor que si estuviera en un hardware de mejores capacidades.

- A pesar de su bajo rendimiento, la red funciona bien para una exigencia mínima, que era el objetivo buscado. Se puede ver que se pierde menos del 1% de los paquetes que circulan por la red y no hay problemas de conexión ya que se mantiene estable durante toda la comunicación. Para una red pequeña en donde se busque una conectividad entre pocos hosts su uso puede ser satisfactorio. Si se busca mejorar su

rendimiento se deberá pensar en reemplazar parte del hardware por otro que no presente estos cuellos de botellas pero que siga manteniendo su característica de bajo costo.

- Su uso para la educación y el aprendizaje sobre Redes Definidas por Software y el protocolo OpenFlow también se puede llevar a cabo ya que es sencillo de utilizar, posee comandos que muestran descripciones muy precisas de sus componentes y, se puede complementar con otros softwares, como Wireshark, para el análisis de protocolos y paquetes.

7. Bibliografía

[1] James F. Kurose & Keith W. Ross, Redes de computadoras, un enfoque descendente, 5° ed., Pearson, 2010.

[2] Joseph D. Sloan, Network Troubleshooting Tools, 1° ed., O'Reilly, 2001.

[3] William Stallings, Comunicaciones y redes de computadores, 7° ed., Person-Prentice Hall, 2004.

[4] Thomas D. Nadeau & Ken Gray, SDN Software Defined Network, 1° ed., O'Reilly, 2013.

[5] Óscar Roncero Hervás, Trabajo Final de Grado "Software Defined Network", Universidad Politécnica de Cataluña, España. Disponible en: http://upcommons.upc.edu/pfc/bitstream/2099.1/21633/4/Memoria.pdf

[6] Wikipedia, OpenFlow. Disponible en: http://es.wikipedia.org/wiki/Openflow

[7] Wikipedia, Redes definidas por software. Disponible en: http://es.wikipedia.org/wiki/Redes_definidas_por_software

[8] Departamento de ciencias de la computación de Stanford. Disponible en: http://yuba.stanford.edu/cs244wiki/index.php/Overview.

[9] Principia Technologica. Disponible en: http://principiatechnologica.com/2014/03/25/investigacion-de-redes-sdn-parte-iii-la-arquitectura-sdn/

[10] Brian Linkletter Blog, Disponible en: http://www.brianlinkletter.com/

[11] Researchgate, Disponible en: https://www.researchgate.net/

[12] Academia GNS3, Disponible en: http://academy.gns3.com/

[13] SDN testbed, Disponible en: https://sdntestbed.wordpress.com

[14] Pakiti, Disponible en: http://pakiti.com/

[15]Redes Cisco, Disponible en: http://www.redescisco.net/

[16] Mikronauts, Disponible en: http://www.mikronauts.com/

[17] Trabajo Final de Grado de David Andrés Serrano Carrera, "Redes Definidas por Software (SDN): OpenFlow", Universidad Politécnica de Valencia.

[18] Trabajo Final de Grado Diana Gabriela Morillo Fueltala "Implementación de un prototipo de una Red Definida por Software (SDN) empleando una solución basada en software", Escuela Politécnica Nacional, Ecuador.

[19] Trabajo Final de Maestría “Software Defined Networking”, Florian Siebertz, Bonn-Rhein-Sieg University, Alemania.

[20]Trabajo Final de Grado “Mecanismos de control de las comunicaciones en la internet del futuro a través de OpenFlow”, Sergio Rodríguez Santamaría, Universidad de Cantabria, España.

[21]Trabajo Final de Maestría “Propuesta de escenarios virtuales con la herramienta VNX para pruebas del protocolo OpenFlow”, Lorena Isabel Barona López, Universidad Politécnica de Madrid, España.

[22] Trabajo Final de Grado “Despliegue de una maqueta de red basada en OpenFlow”, Carlos Arteche González, Universidad de Cantabria, España.

[23]Trabajo Final de Grado “Gestión y control de calidad de servicio a través de SDN en tiempo real”, Daniel Guija Alcaraz, Universidad Politécnica de Cataluña, España.

[24]Trabajo Final de Grado “Implementación de un prototipo de una Red Definida por Software (SDN) empleando una solución basada en hardware”, Juan Carlos Chico Giménez, Escuela Politécnica Nacional, Ecuador.

[25]Trabajo Final de Grado “Seguridad en Redes Definidas por Software”, Andrés Ballesteros y Lucas Ezequiel Bujedo, Instituto Universitario Aeronáutico, Argentina.

[26]Trabajo Final de Grado “OpenFlow: Redes Definidas por Software”, Adolfo Herrando e Insong Kim, Instituto Universitario Aeronáutico, Argentina.

[27] Trabajo Final de Maestría “Increasing robustness of Software Defined Networks”, B.J. van Asten, Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS), Holanda.