Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros de Telecomunicación

PROPUESTA DE ESCENARIOS VIRTUALES CON LA HERRAMIENTA VNX PARA PRUEBAS DEL PROTOCOLO OPENFLOW

TRABAJO FIN DE MÁSTER

Lorena Isabel Barona López

2013

Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros de Telecomunicación

Máster Universitario en
Ingeniería de Redes y Servicios Telemáticos

TRABAJO FIN DE MÁSTER

PROPUESTA DE ESCENARIOS VIRTUALES CON LA HERRAMIENTA VNX PARA PRUEBAS DEL PROTOCOLO OPENFLOW

Autor

Lorena Isabel Barona López

Director

David Fernández Cambronero

Departamento de Ingeniería de Sistemas Telemáticos

2013

Resumen

El nacimiento de las aplicaciones en tiempo real, el video *streaming*; la masificación de las redes sociales, la introducción del *cloud computing* y muchos otros servicios, han dado como resultado el crecimiento exponencial del tráfico que circula por la red. A pesar de ello se continúa utilizando las mismas tecnologías que hace cincuenta años y los avances en cuanto a nuevas formas de comunicación y tratamiento de la información son casi inexistentes. Se ha intentado cubrir cada necesidad con la creación de protocolos, que dicho sea de paso toman mucho tiempo en ser estandarizados. Si bien se ha dado una solución temporal a todos los requerimientos que demanda el mercado, ésta no representa una solución global al verdadero problema, la carencia de métodos de comunicación más eficientes. En consecuencia se plantea la necesidad de cambiar la forma de comunicación de las redes, en la cual se le proporcione mayor inteligencia a la misma.

Es así que nace el concepto de *Software Defined Networking* , el cual está revolucionando el campo de las comunicaciones mediante el control de los dispositivos de la red desde un software exterior, con la ayuda del protocolo OpenFlow creado especialmente para este propósito. *OpenFlow* propone nuevas características que permiten la programabilidad de las redes y la experimentación sin la necesidad de que los fabricantes de equipos de *networking* expongan la estructura interna de sus productos; basta con que se adicione compatibilidad con el estándar en los equipos de cada casa fabricante. De igual forma se han desarrollado herramientas para experimentar con SDN, tal es el caso del emulador Mininet.

SDN es una tecnología que ofrece un amplio panorama de investigación y que a futuro se proyecta como la base de lo que se considere una nueva era de las comunicaciones. El presente trabajo se centra en la creación de un escenario virtual, similar a Mininet, para experimentación del protocolo OpenFlow, basado en la herramienta de virtualización desarrollada por la Universidad Politécnica de Madrid (VNX).

En este sentido es necesario proveer a VNX de dispositivos que soporten *OpenFlow*. En primer lugar un switch, para lo cual se ha escogido la herramienta Open vSwitch que permite la simulación de un switch para ambientes virtualizados y que tiene soporte para el protocolo *OpenFlow*. Como segundo elemento un software externo para el control de la red, en este caso se ha seleccionado POX, el cual es el controlador más

reciente dentro del stack SDN y que además es el más utilizado en la actualidad. Finalmente como tercer elemento se encuentra el protocolo OpenFlow.

En base a estos tres elementos se diseña y se implementa un escenario virtual similar al que se usa en el *OpenFlow Tutorial* y a la herramienta Mininet, adaptado al entorno usado en VNX. Finalmente sobre los escenarios se realiza pruebas de concepto del protocolo OpenFlow. Dicho escenario permite la experimentación con esta nueva tecnología y sienta las bases para escenarios más complejos.

Abstract

The emergence of real time applications, video streaming, the massification of social networks, the introduction of cloud computing and many other services, has resulted in the exponential growth of traffic on the network. Nevertheless, the telecommunications industry has been using the same technologies for fifty years and the improvements in new means of communication and information processing are almost nonexistent. The creation of protocols has tried to cover each of these necessities which, incidentally, take a long time to be standardized. Although, it has been a temporary solution to all industry requirements, this is not a real solution to the lack of efficient communication methods. Consequently, there is the necessity to improve the communication process, providing greater intelligence to the network.

Due to this requirement, the concept of Software Defined arises, which is revolutionizing the communications field, controlling network devices from external software with the help of the OpenFlow protocol, created especially for this purpose. OpenFlow offers new features that enable network programmability and experimentation without the need of exposing the internal structure of the equipments. It is sufficient to add compatibility with the standard on the devices of each vendor. Likewise, tools have been developed to experiment with SDN, such as the Mininet emulator.

SDN is a technology that provides a broad scope of research and it could be the basis of a new age of communications. This work focuses on the creation of a virtual scenario, similar to Mininet, for experimentation with OpenFlow protocol, based on a virtualization tool developed by the Universidad Politécnica de Madrid (VNX).

In this sense it is necessary to provide VNX with devices that support OpenFlow. First of all, with a switch, for which the Open vSwitch tool has been chosen. It allows the simulation of a switch in virtualized environments and it has support for OpenFlow protocol. A second element to control the network, external software, is in this case POX, which is the latest within the SDN stack and is also the most widely used today. Finally as a third element, the OpenFlow protocol.

With these three elements a virtual scenario, similar to OpenFlow Tutorial and Mininet tool, is designed and implemented, and adapted to the environment used in VNX.

Finally, proofs of concept of the OpenFlow protocol are performed. This scenario allows experimentation with this new technology, and provides the basis for more complex scenarios.

Índice general

Índice de Figuras

Índice de Tablas

Siglas

SDN	Software Defined Networking
VNX	Virtual Networks over linuX
OVS	Open vSwitch
ONF	Open Networking Foundation
NAT	Network Address Translation
RFC	Request for Comments
IP	Internet Protocol
MPLS	Multiprotocol Label Switching
VLAN	Virtual Local Area Networks
VPN	Virtual Private Networks
QoS	Quality of Service
VMM	Virtual Machine Monitor
API	Application Programming Interface
VNUML	Virtual Network User Mode Linux
GENI	Global Environment for Network Innovations
OVF	Open Virtualization Format
IT	Information Technology
ACED	Autoconfiguration and Command Execution Daemon
SSL	Secure Sockets Layer
STP	Spanning Tree Protocol
CLI	Command Line Interface
CPU	Central Processing Unit
TLS	Transport Layer Security

LACP	Link Aggregation Control Protocol
GRE	Generic Routing Encapsulation
OSPF	Open Shortest Path First
CDP	Cisco Discovery Protocol
LLDP	Link Layer Discovery Protocol
VM	Virtual Machine
JDK	Java Development Kit
JRE	Java Runtime Environment
IETF	Internet Engineering Task Force

1 Introducción

1.1 Contexto

La evolución de Internet ha pasado por diferentes etapas que han marcado el desarrollo tecnológico no solamente de la red en sí, sino también de aplicativos, dispositivos y terminales de red y formas de prestación de servicios. Internet pasó de conectar unas pocas computadoras a inicio de los años sesenta, a conectar cada punto del planeta a través de miles de millones de conexiones. Sin embargo, han transcurrido casi sesenta años y se continúa utilizando la misma arquitectura de red, que si bien ha funcionado hasta la fecha, producen una osificación de la misma. El estado de la industria de *networking* para el año 2007[1] deja como saldo 5400 RFCs, una gran cantidad de protocolos con funciones complejas tal es el caso de los NATs, firewalls, ingeniería de tráfico, servicios diferenciados, que se ven reflejadas en la creación de estándares específicos, que en lo posterior son puestos en hardware para cubrir dichas necesidades. Las redes son cada vez más rápidas, sin embargo no mejoran. Si se hace una retrospectiva, se continúa utilizando las mismas tecnologías tal es el caso de *Ethernet* e IP. Además los cambios que se realizan para mejorar las redes parecen más lentos en comparación con su crecimiento.

El modelo de comunicación de Internet no puede ser cambiado fácilmente debido a que cada casa fabricante de dispositivos de comunicación manejan una estructura cerrada y los actuales sistemas abiertos no proporcionan los atributos deseados (escalabilidad, fidelidad de rendimiento, tráfico real, etc). Además para validar una nueva alternativa de comunicación se pasa por un proceso largo hasta que ésta es estandarizada, motivo por el cual grandes descubrimientos han quedado limitados a una simple investigación.

Todos estos factores conjugados con la explosión de las redes sociales, servicios de *streaming*, aplicaciones en tiempo real y servicios sobre IP han puesto a trabajar a la comunidad científica en alternativas para mejorar los mecanismos de comunicación y que faciliten el despliegue y experimentación de las nuevas investigaciones en un ambiente donde se podrá trabajar con tráfico en tiempo real; con lo cual los procesos de validación serán mucho más rápidos, y por tanto se verá reflejado en una mejora en el proceso de comunicación y la prestación de servicios a través de la red. Es así que se han definido algunas posibles soluciones como *MultIProtocol Label Switching* (MPLS) y *Software Defined Networking* (SDN).

Dentro del marco de las redes definidas por software se tiene como precursor el proyecto *Ethane* [2] realizado en el año 2007. Este proyecto presenta una nueva arquitectura cuyo objetivo es que la red sea más administrable, donde la conectividad es gobernada a un nivel más alto y con políticas que permiten mayor granularidad del control de la red.

A partir de los resultados obtenidos de *Ethane* se empieza a plantear el concepto de SDN que propone una arquitectura de red en donde se desacopla el plano de control del plano datos de la red, de esta forma se vuelve programable. Lo que se intenta es ofrecer una arquitectura adaptable, que proporcione capacidades de automatización y programabilidad de las comunicaciones con un alto grado de eficiencia y que permita el soporte adecuado para todos los servicios manejados actualmente. Por ejemplo, cuando en la vida real un automóvil sigue una ruta hacia un destino, no está exento de los problemas de congestión o cierre de la vía, por tanto la solución sería dar un aviso oportuno y proponer una ruta alterna en ese mismo momento. De esta forma se obtiene una mayor eficiencia en cuanto a tiempos de respuesta, retrasos, uso de recursos, etc. SDN se plantea de manera similar, en donde los datos son enviados en forma de flujos y que en caso de incidentes se tenga la capacidad de reprogramar a través de software (desvinculación del hardware del software). Los actuales dispositivos de red permiten la reconfiguración de rutas a nivel de hardware, es el equipo de red el que a través de sus reglas controla el tráfico y cuya gestión está directamente ligada a una marca propietaria. SDN proporciona una arquitectura homogénea y modular en donde se puede experimentar o personalizar la conducta de la red a través de un programa de software simple que manipula los flujos de la red. La única condición, utilizar un lenguaje de comunicación común para lo conexión de los dispositivos, tal es el caso del protocolo *OpenFlow*.

SDN es una arquitectura muy joven que se encuentra en fase de experimentación. Una de las herramientas más utilizadas para ello es el emulador de redes Mininet, el cual permite construir escenarios virtuales en un único ordenador. De igual forma el Departamento de Ingeniería de Sistemas Telemáticos de la Universidad Politécnica de Madrid desarrolló la herramienta VNX para la creación de escenarios virtuales. El principal objetivo de un *testbed* es proporcionar una plataforma para la experimentación de redes y tecnologías bajo condiciones controladas.

Dentro de este contexto el presente trabajo se centra en la creación de escenarios virtuales con la herramienta VNX para la experimentación de SDN y *OpenFlow*, tomando como ejemplo los trabajos realizados en el tutorial *OpenFlow* y la herramienta Mininet.

1.2 Objetivos

La presente memoria pretende dar a conocer la importancia de *Software Defined Networking* y el protocolo *OpenFlow* como tecnologías que se proyectan a ser el futuro de las redes. En virtud de ello se ha definido los siguientes objetivos:

- ✓ Presentar los principales conceptos de la tecnología en base a la revisión del estado del arte de SDN y la herramienta *Open vSwitch*.

- ✓ Diseñar un escenario similar al que se utiliza en el *OpenFlow Tutorial*.

- ✓ Diseñar e implementar tres escenarios virtuales sobre la herramienta VNX, para la realización de pruebas del protocolo *OpenFlow*.

- ✓ Validar el funcionamiento de los escenarios a través de la realización de pruebas de concepto del uso del protocolo *OpenFlow*.

1.3 Metodología

Para llevar a cabo cada uno de los objetivos presentados en el apartado anterior, se requieren ciertos conocimientos que serán obtenidos a partir de la realización de las siguientes tareas:

- ✓ El desarrollo del trabajo requiere la conjugación de conceptos de SDN, virtualización y Open vSwitch, por tanto es necesario la revisión del estado del arte de dichos conceptos.

- ✓ Una vez establecidos los conceptos, se realizará el diseño e implementación de los escenarios, tomando como referencias el *OpenFlow Tutorial* y la herramienta de simulación Mininet desarrollada por la Universidad de Stanford, aplicados al desarrollo de escenarios virtuales con la herramienta VNX.

- ✓ Finalmente, se realizará la validación de resultados en base a mediciones, pruebas de conectividad y capturas de tráfico de los tres diferentes escenarios.

1.4 Estructura de la memoria

El presente trabajo proporciona una descripción de SDN, teniendo como protagonista al protocolo *OpenFlow* y su aplicación mediante el diseño del escenario virtual sobre la herramienta VNX, para lo cual se ha dividido la presente memoria en 5 capítulos, como se detalla a continuación.

El capítulo 2 presenta en primera instancia conceptos relacionados a la virtualización de redes y a la herramienta VNX. A continuación se realiza la revisión del estado del arte de SDN y *OpenFlow* para establecer los conceptos básicos asociados a dicha tecnología y también la descripción de los proyecto Open vSwitch y Mininet. Todos estos conceptos son las bases del escenario virtual propuesto.

En el capítulo 3 se establece los requerimientos de los diferentes escenarios; tomando como referencia el *OpenFlow Tutorial* y el emulador Mininet, y en lo posterior realizar el diseño e implementación de los mismos. El primer escenario virtual constará de tres máquinas conectadas a un switch, y éste a su vez a un único controlador. En el segundo escenario se probará funcionalidades de redundancia para lo cual se agrega a la topología anterior un controlador secundario. Finalmente el tercer escenario desplegará una red que contiene 3 diferentes subredes conectadas a través de un router, con lo cual se prueba las bondades proporcionadas por la herramienta VNX y se tiene el escenario ideal para realizar pruebas de concepto de la transmisión de flujos con *OpenFlow*.

En el capítulo 4 se muestran los resultados de la implementación de los escenarios virtuales y la validación de la transmisión de datos del protocolo *OpenFlow*, para lo cual se utiliza la herramienta *Wireshark* y pruebas de conectividad a través de línea de comandos.

Finalmente el capítulo 6, presenta las conclusiones y futuras líneas de trabajo que se desprenden del desarrollo de la presente memoria.

2 Estado del arte

En primera instancia es necesario abordar de manera general el tema de virtualización de redes ya que es la base de la herramienta VNX y del escenario propuesto.

2.1 Virtualización

La virtualización consiste en la creación de instancias virtuales de algún tipo de recurso (sistema operativo, almacenamiento, hardware) dentro de una infraestructura física común de red. La virtualización permite la coexistencia de múltiples instancias virtuales sobre el mismo soporte físico y que son independientes entre sí. El ejemplo más conocido es la creación de un sistema operativo dentro de otro denominado host o anfitrión. Existen algunos términos [3] que se relacionan con la virtualización de las cuales cabe mencionar:

- ✓ Virtual Local Área Network.- es una red de área local creada virtualmente que agrupa un conjunto de dispositivos conectados de manera lógica, de tal forma que se encuentren dentro de un único dominio de *broadcast* independientemente de su conectividad física, en consecuencia cada VLAN estará aislada del resto de redes.
- ✓ Virtual Private Networks.- una VPN permite la conectividad hacia un sitio a través de un canal privado virtual, que suele usar un túnel seguro para la comunicación. Las VPNs generalmente son usadas para la comunicación entre diferentes sucursales de organizaciones que necesitan un medio más seguro para la transmisión de la información.

- ✓ Redes activas y programables.- son aquellas que permiten modificar dinámicamente la operación de la red, de dos maneras; la primera muestra un enfoque de *señalización abierta* en donde se hace una distinción clara entre los planos de transporte, control y gestión, enfatizando la calidad de servicio y la segunda se enfoca en la personalización de los servicios de red, ofreciendo mayor granularidad en el transporte de paquetes y mayor flexibilidad que el enfoque de señalización abierta.

- ✓ Redes Overlay.- conocida también como red superpuesta, es una red de nodos y enlaces lógicos que están construidos sobre una red existente; su principal objetivo es la implementación de servicios que la red no posee, es decir presta un servicio a lo que se puede denominar la capa superior. El más claro ejemplo de una red overlay es Internet.

Cada uno de estos conceptos conllevan la coexistencia lógica de redes, el objetivo de la virtualización es que las redes virtuales puedan coexistir de forma aislada pero

realizando una compartición efectiva de recursos entre sí. El proceso de virtualización se produce en la capa de abstracción conocida como Hypervisor o *Virtual Machine Monitor*, la cual se encarga de la gestión de los recursos (almacenamiento, memoria, etc) otorgados por la máquina host (máquina física) a la máquina huésped (máquina virtual). La máquina virtual funciona como un sistema totalmente independiente. Existen dos tipos de VVM, los cuales son:

- ✓ Tipo I.- en la cual se virtualiza a nivel de hardware. Se puede citar dentro de este grupo a Hyper –V y Xen.
- ✓ Tipo II.-el proceso de virtualización se desarrolla sobre otro sistema operativo. Como ejemplos se tiene a virtualBox y QEMU.

2.1.1 Tipos de virtualización

Dentro de los métodos [4] de virtualización se pueden nombrar:

- ✓ Virtualización completa.- provee una simulación completa tanto de hardware como de software. El sistema operativo de la máquina huésped no es modificable. Dentro de este grupo se encuentra VMware Workstation.
- ✓ Paravirtualización.- no se realiza virtualización a nivel de hardware, sin embargo para cambios a nivel del sistema operativo de la máquina virtual se proporciona un conjunto de APIs que interactúan entre el hardware y el software. Como ejemplo de este tipo de virtualización se tiene XEN.

- ✓ Virtualización a nivel de sistema operativo.- virtualiza solamente a nivel de software y crea múltiples máquinas virtuales en paralelo sobre un solo equipo físico. Las máquinas virtuales corren el mismo sistema operativo que el equipo anfitrión.
- ✓ Virtualización de hardware.- en este tipo de virtualización el hipervisor se conecta directamente al hardware, controlando y sincronizando el acceso de las máquinas virtuales a los recursos de hardware.
- ✓ Virtualización a nivel de aplicación.- no se virtualiza ni hardware ni software, sin embargo el usuario puede correr una aplicación usando los recursos locales. Cada usuario tiene su ambiente aislado para la aplicación.

Dentro de las principales ventajas del uso de la virtualización se tiene la flexibilidad, la escalabilidad, programabilidad, el aislamiento, facilidades para el desarrollo, la investigación, entre otras. En este contexto y tratando de aprovechar las bondades que presta la virtualización se desarrolla los proyectos VNUML[5] (Virtual Network User-Mode-Linux) y EDIV, con su modificación VNX (Networks over linuX) desarrollado

por el departamento de Departamento de Ingeniería de Sistemas Telemáticos de la Universidad Politécnica de Madrid.

2.1.2 Herramienta VNX

Como parte de los proyectos de investigación realizados por la Universidad Politécnica de Madrid, se desarrolló la herramienta de código abierto VNX[6] que permite la creación de escenarios de prueba virtuales (*testbeds*) de forma automática. Su principal objetivo es brindar a los estudiantes un entorno donde puedan interactuar de forma más real con escenarios de red, de forma similar a los programas GNS3[1], NetKit[2], entre otros. Las mejoras que introduce VNX en comparación con su herramienta antecesora VNUML son:

- ✓ Integración de nuevas plataformas de virtualización que permiten otros sistemas operativos como *Windows*, *FreeBSD*; para ello se introduce el uso de libvirt. Libvirt[7] es el API estándar de virtualización de Linux, el cual permite acceder a las capacidades de virtualización. Entre los hipervisores que soporta están KVM/QEMU, virtualBox, XEN, etc. Además se integra el uso de Dynamips y Olive, el primero es el emulador de router Cisco y el segundo de Juniper.

- ✓ Gestión individual de máquinas virtuales. Se proporciona comandos como "*-console*" para abrir las consolas o terminales; los comandos *start, stop, restart*, *suspend*, etc.

- ✓ Autoconfiguración y capacidades para ejecución de comandos a través de la creación de imágenes de máquinas virtuales, similares a los archivos de virtualización OVF (Open Virtualization Format). Además se incluye un demonio de autoconfiguración y ejecución de comandos (ACED) que se ejecuta dentro de las propias máquinas virtuales. El demonio ACED realiza la lectura del fichero XML y configura las máquinas con esos parámetros.
- ✓ Versión para sistemas distribuidos (Mejoras conseguidas con EDIV).

VNX es una herramienta de código abierto, que funciona sobre un sistema operativo Linux y que presenta una arquitectura modular controlada por un API basada en plugins, como se puede visualizar en la figura 1.

[1]http://www.gns3.net/
[2]http://wiki.netkit.org/index.php/Main_Page

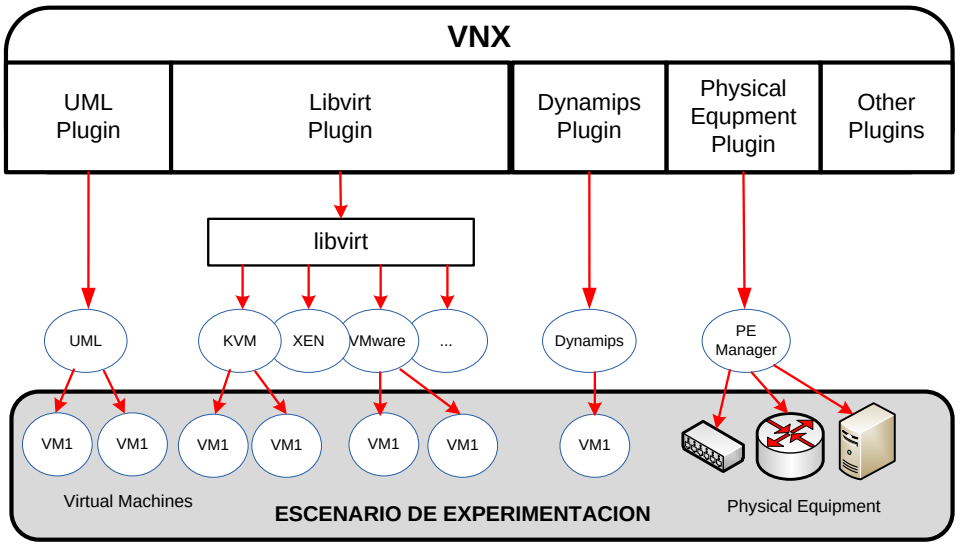

Figura 1. Arquitectura VNX.

Entre los plugins desarrollados se tiene dynamips, UML que incluye el código de VNUML y versión simplificada de libvirt. El proceso para la creación de escenarios virtuales es el siguiente:

1. Se realiza el diseño general del escenario.
2. En la fase de especificación el usuario generará el archivo en el lenguaje VNUML, que puede llevarse a cabo mediante un simple editor XML.
3. La herramienta VNX procesará la especificación y en base a ésta creará el escenario virtual.
4. Finalmente el usuario podrá interactuar con el escenario y podrá ejecutar comandos directamente en las máquinas virtuales o a través de VNUML.

En la figura 2 se puede visualizar el procedimiento de creación de escenarios virtuales.

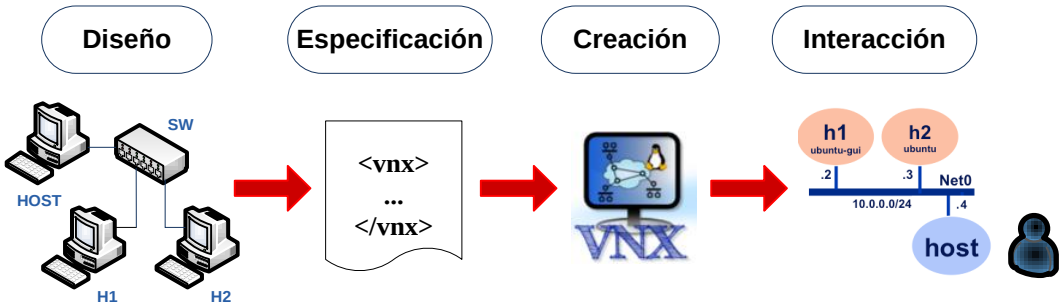

Figura 2. Proceso Creación Escenarios VNX

VNX se construye en base a dos componentes principales:

- ✓ El uso del lenguaje XML con las especificaciones de la herramienta para describir el escenario virtual que se desee desplegar (Especificación del lenguaje VNUML).

- ✓ Un intérprete del lenguaje VNUML el cual realiza el parseo del escenario definido y la construcción y gestión del mismo.

Las principales ventajas de VNX son:

- ✓ Su estructura modular, mediante la cual se crean módulos diferentes para otros sistemas de una forma rápida y sencilla.
- ✓ Mejor control sobre cada máquina virtual, ya que se puede realizar operaciones sobre la misma sin que afecte al resto de los elementos del escenario.
- ✓ Escalabilidad, que solamente se verá limitada por los recursos y capacidades de la máquina host. Aunque se tiene la posibilidad de dividir los escenarios virtuales y conectarlos entre sí a través de una infraestructura de red común. Cada escenario virtual será ejecutado en una máquina host diferente.
- ✓ Transparencia.
- ✓ Permite la creación de múltiples máquinas virtuales desde un solo sistema de archivos o *filesystem*, para lo cual utiliza la técnica *copy on write* conocida también como "*cow*". Esta técnica permite que dos o más máquinas virtuales puedan compartir un mismo *filesystem*, que es se encuentra en modo de solo lectura (*read-only*). Cada vez que una máquina necesita cambiar los parámetros del sistema utiliza un *filesystem* privado donde las diferencias (con el *filesystem* original) son guardadas. En consecuencia permite la reducción de la capacidad de almacenamiento.

El proyecto VNX ha pasado a una etapa de maduración en donde es posible el despliegue de grandes y complejas redes. Una de las líneas de acción para continuar con su desarrollo es la fusión con la herramienta *Open vSwitch*, así como el desarrollo de *testbed*s de nuevas tecnologías, tal es el caso de SDN y *OpenFlow*..

2.2 Software Defined Networking (SDN)

Con el desarrollo de la tecnología y la masificación de los servicios de Internet se ha visto la necesidad de plantear una nueva arquitectura de red que permita manejar de manera eficiente las comunicaciones y las nuevas capacidades inherentes a la virtualización, el *cloud computing*, el creciente uso de dispositivos móviles, aplicaciones

en tiempo real, etc; para las cuales la estructura de las redes tradicionales no proporcionan el rendimiento esperado. La tendencia[9] de los nuevos servicios de red obligan a:

- ✓ Cambios en los patrones de tráfico.- en los últimos años el tráfico que circula por la red ha crecido considerablemente si se compara con redes tradicionales. Ahora el usuario necesita la conexión a diferentes servidores y bases de datos (que pueden encontrarse en un entorno distribuido o en *Cloud)*, en cualquier momento, desde cualquier lugar y a través de diversos dispositivos.

- ✓ Personalización de servicios.- el incremento del uso de dispositivos móviles que acceden a los datos de la empresa obligan a generar mejores mecanismos de encaminamiento y también brindar protección de los datos que son utilizados en la conexión.
- ✓ Crecimiento de los servicios en *Cloud*.- los servicios tanto de *Cloud* público como privado se han masificado por tanto se requiere mayor agilidad para el acceso a servicios, infraestructura, recursos de procesamiento; etc.
- ✓ Requerimientos para *big data*.- implica mayor ancho de banda y capacidad de la red para el procesamiento en paralelo de miles de servidores, los cuales procesan información en el orden de los *tera bytes*.

Las tecnologías actuales han ayudado a palear todas estas necesidades, sin embargo no son eficientes. Antes de adentrarse en todo lo que implica SDN, existen algunas premisas que dieron como resultado su planteamiento y desarrollo, A continuación se describe algunas limitaciones de las arquitecturas actuales:

- ✓ Las arquitecturas tradicionales no fueron diseñadas para soportar las necesidades de ancho de banda y servicios actuales, tal es el caso del *streaming* o los juegos en línea, a lo cual se debe añadir presupuestos reducidos que limitan aun más las capacidades de la red.
- ✓ La complejidad para el desarrollo y estandarización de nuevas tecnologías. Cuando se desarrolla una nueva aplicación o servicio, se crea un nuevo protocolo que permite cubrir dicha necesidad, sin embargo el proceso es considerablemente largo.

- ✓ Dificultad de administración y configuración.- si por ejemplo se mueve un dispositivo, lo más probable es que el administrador de red necesite manipular reglas del router, *ACLs*, reglas de firewall, *VLANs*, etc. Cualquier cambio está ligado al tipo y en algunos casos a la marca del dispositivo, por tanto los encargados de *IT* prefieren mantener una estructura estática de la red y en

consecuencia se evita cualquier cambio que pueda generar un comportamiento inadecuado. Esta estructura estática de las redes provoca la pérdida de dinamismo de la red, en consecuencia no es posible adaptarse a los cambios de tráfico y a las demandas del usuario. Además muchos de los parámetros son configurados de forma manual.

- ✓ Políticas inconsistentes.- si se aplica un cambio de reglas en una red lo suficientemente grande el tiempo de convergencia puede llegar a ser muy largo y por tanto provocará inestabilidad, que eventualmente traerá problemas de seguridad y mal funcionamiento.
- ✓ Problemas de escalabilidad.- cada vez que un dispositivo es incorporado a la red es necesario su configuración y en lo posterior su gestión; el problema surge cuando son miles de equipos, en consecuencia se limita el crecimiento de la red.

- ✓ Dependencia del fabricante.- actualmente se necesita más y variados servicios que se ven limitados a la introducción de cambios por parte de los fabricante y al lanzamiento de nuevas versiones; por tanto muchas ideas innovadoras no son puestas en práctica. Cada fabricante trabaja sobre su propia plataforma de manera cerrada y no existe concordancia con las necesidades del mercado y las capacidades de la red.

En respuesta a todas estas limitaciones surge SDN. SDN puede ser definida como la tecnología que permite desacoplar el plano de datos del plano de control. La inteligencia y el estado de toda la red se encuentran centralizadas lógicamente en un dispositivo que se conoce como controlador. La arquitectura SDN permite tener el control de la red, proporciona facilidades de programabilidad, automatización, escalabilidad, flexibilidad y adaptabilidad en función de las necesidades y los nuevos servicios del mercado.

Como ya se mencionó SDN tiene sus cimientos en el proyecto *Ethane*[2] desarrollado en la universidad de *Stanford* en el año 2007, el cual proporciona un mejor modelo de administración de la red, basado en los siguientes tres principios:

- ✓ La red debe ser gobernada por políticas declaradas en los niveles altos.
- ✓ Las políticas determinan la ruta que siguen los paquetes.
- ✓ La red debe permitir la identificación unívoca entre el paquete y su procedencia.

En *Ethane* la comunicación de la red se basa en el intercambio de flujos entre switches y un controlador centralizado. El controlador conoce la topología de toda la red y lleva a cabo el tratamiento de los flujos en base a las políticas que tiene configurada. Al switch

se lo puede definir como el componente que envía los datos en función de las indicaciones del controlador, no tiene ninguna capacidad de decisión. El switch consiste en una tabla de flujos y un canal seguro para la comunicación con el controlador. Adicionalmente *Ethane* permite la compatibilidad hacia atrás con los dispositivos de comunicaciones actuales y fue implementado tanto con dispositivos cableados como inalámbricos.

Posterior a *Ethane* se desarrollan nuevas investigaciones y herramientas para SDN; es así que en el año 2008 aparecen el primer controlador conocido como NOX [10] y el protocolo *OpenFlow*. Un año más tarde se desarrolla la herramienta FlowVisor [11] y el emulador Mininet [12]. Los primeros despliegues de la tecnología se llevan a cabo en la propia universidad de *Stanford* en el 2009 y por el grupo GENI (Global Environment for Network Innovations) en el año 2010. GENI juntó a las universidades más importantes de Estados Unidos para realizar el despliegue de SDN y *OpenFlow*, su principal objetivo es promover la experimentación, la investigación y la colaboración.

Con este antecedente se crea en el año 2011 Open Networking Foundation (ONF) [9] con el apoyo de 17 organizaciones que adoptan SDN. Posteriormente se han ido integrando las mayores casas fabricantes de equipos de *networking*, como por ejemplo NEC, HP, IBM, Juniper, Cisco, etc. Actualmente ONF consta de más de 50 empresas que utilizan SDN y ayudan en la estandarización y creación de nuevos productos que soporten la tecnología. ONF es una organización sin fines de lucro cuyo objetivo es liderar investigaciones y adelantos en SDN, así como la estandarización de los elementos críticos de la arquitectura SDN, como por ejemplo *OpenFlow*.

Un impulso interesante para SDN se produjo en el año 2012 cuando el gigante *Google* adoptó e implementó SDN con el protocolo *OpenFlow* en su infraestructura. Google empezó a trabajar en su propia versión de *OpenFlow* aproximadamente desde el año 2010. El objetivo fue conectar la red interna que enlaza sus data centers (G-Escale). G-Escale es la que se encarga de enviar los flujos dependiendo de sus características y de la demanda. Aunque Google ha dado impulso a SDN, también ha anunciado que no liberará el proyecto.

SDN es una arquitectura muy joven que aún se encuentra en fase de experimentación, en la cual se ha pasado de una estructura totalmente cerrada y limitada (Figura 3) a una estructura abierta (Figura 4).

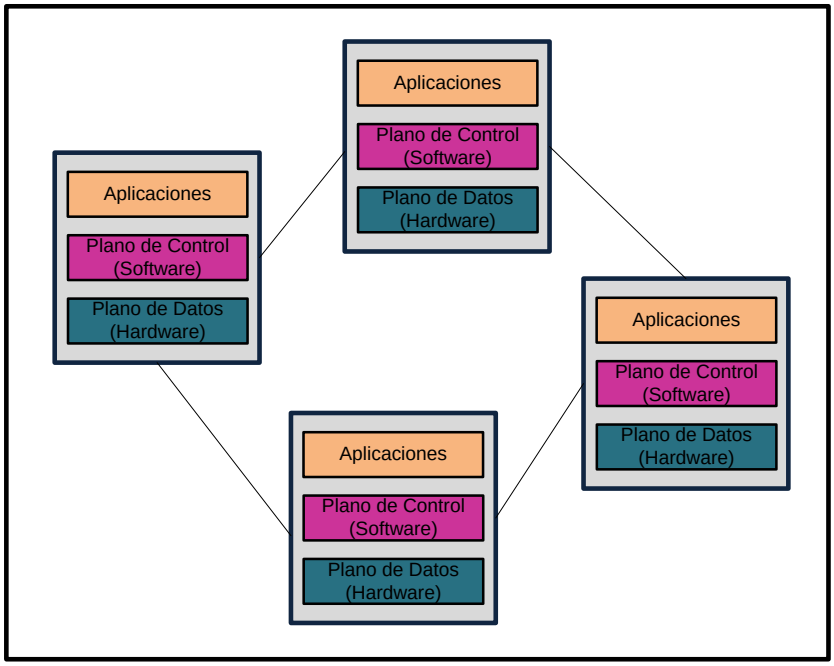

Figura 3. Arquitectura Cerrada

El inconveniente de una arquitectura cerrada es que la escalabilidad es limitada y que la prestación de servicios dependen de cada casa fabricante. Por ejemplo para poder usar nuevos protocolos de enrutamiento, VPNs, control de acceso, administración, o cualquier otro requerimiento ha sido necesario esperar la actualización de los sistemas y esto conlleva a ser totalmente dependientes del fabricante.

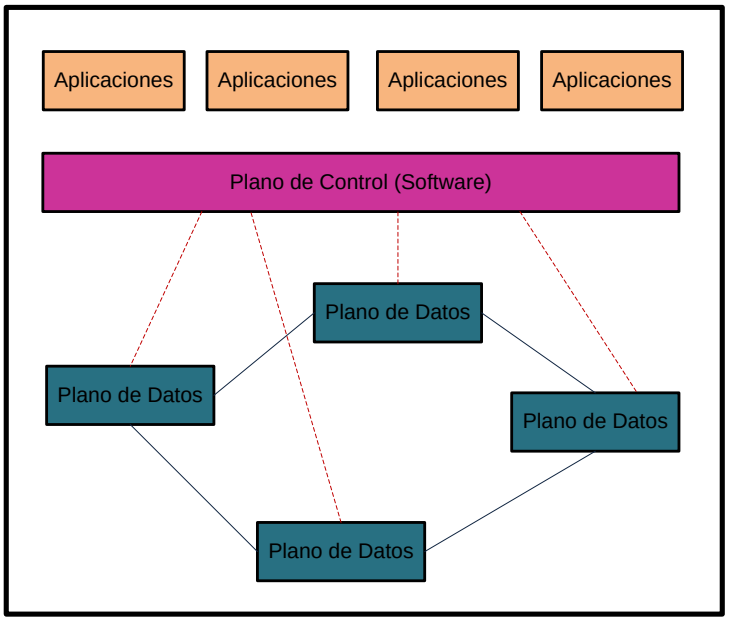

Figura 4. Arquitectura Abierta

Por otra parte una arquitectura abierta proporciona una interfaz común para el desarrollo, en donde se delimitan parámetros básicos y en el caso de SDN se puede citar los siguientes beneficios:

- ✓ Gestión centralizada de los dispositivos de red a través de un controlador, pudiendo ser estos dispositivos de diferentes fabricantes. A través de la gestión centralizada se tiene una visión global de la estructura de la red.
- ✓ Mejora la automatización y la gestión por medio del uso de un API común.
- ✓ Rápida innovación (no es necesario configuraciones de forma individual o actualizaciones de los dispositivos propietarios). Se puede alterar las políticas en tiempo real y desplegar nuevos servicios en poco tiempo.
- ✓ Capacidad de programación o gestión por parte de operadores, empresas e inclusive de usuarios finales a través de ambientes de programación comunes. En este caso se elimina la limitación de espera de un *release* del software.
- ✓ Incrementa la confiabilidad y la seguridad debido a la gestión centralizada, a la aplicación de políticas uniformes y a los tiempos de respuesta frente a fallos de la red.
- ✓ Mayor granularidad para el control de la red.
- ✓ Mejor calidad de experiencia para el usuario, ya que la red tiene la capacidad de adaptarse a las necesidades y a diferentes perfiles, tal es el caso de servicios de *multicast*, ingeniería de tráfico, calidad de servicio, entre otros.
- ✓ Proporciona mejores entornos de prueba y de fácil accesibilidad.

Según un reporte del crecimiento de SDN[13] se calcula que su impacto para el año 2018 superará los 25 billones por año, por tanto se muestra a SDN como el cambio que sucede una vez por generación.
SDN tiene una arquitectura flexible y dinámica que permite a toda la comunidad tecnológica trabajar sobre ella e impulsarla. SDN está cambiando el enfoque de gestión de los dispositivos de red e inclusive las estrategias de negocio de las grandes casas fabricantes que hoy en día forman parte activa de su desarrollo.

2.2.1 Arquitectura y Componentes

La arquitectura SDN se basa en la transmisión de lo que se conoce como *flows*. Un *flow* se puede definir como una secuencia de paquetes que atraviesan una red y que comparten campos de su cabecera de datos. [14]

El proceso de comunicación inicia cuando un usuario desde un dispositivo realiza una petición de un servicio. El switch recibe dicha petición y se comunica con el controlador SDN, el cual proporcionará las instrucciones a seguir para el servicio especificado. Luego, el switch guardará en una serie de tablas las instrucciones dadas para usarlas en comunicaciones posteriores. Para la comunicación entre el switch y el controlador se utiliza el protocolo *OpenFlow* a través de un canal seguro.

Toda la inteligencia de la red se encuentra en el software que controla la red, es decir en el controlador SDN. Los dispositivos de red no tiene ninguna capacidad de decisión. Se puede realizar una analogía con una arquitectura de tres capas:

- ✓ La capa de aplicación.- se relaciona con las aplicaciones del negocio que sean requeridas y que son personalizadas por el usuario. Para acoplarse con el software de control se ayudan del uso de diferentes APIs.
- ✓ La capa de control.- .- que se encargará de *definir* el tratamiento de los *flows* (en el plano de datos) mediante el software de control SDN (controlador). El software de control permite la transmisión de las instrucciones hacia el switch a través de un protocolo estandarizado, el más común *OpenFlow*. Esta capa se encuentra ligada con el plano de control.
- ✓ La capa de infraestructura.- formada por los dispositivos físicos de comunicación, tal es el caso de switches y routers. Los dispositivos de red administran las tablas de flujo en función de las indicaciones del controlador. Esta capa se encuentra relacionada con el plano de datos.

En la figura 5 se puede visualizar la arquitectura SDN.

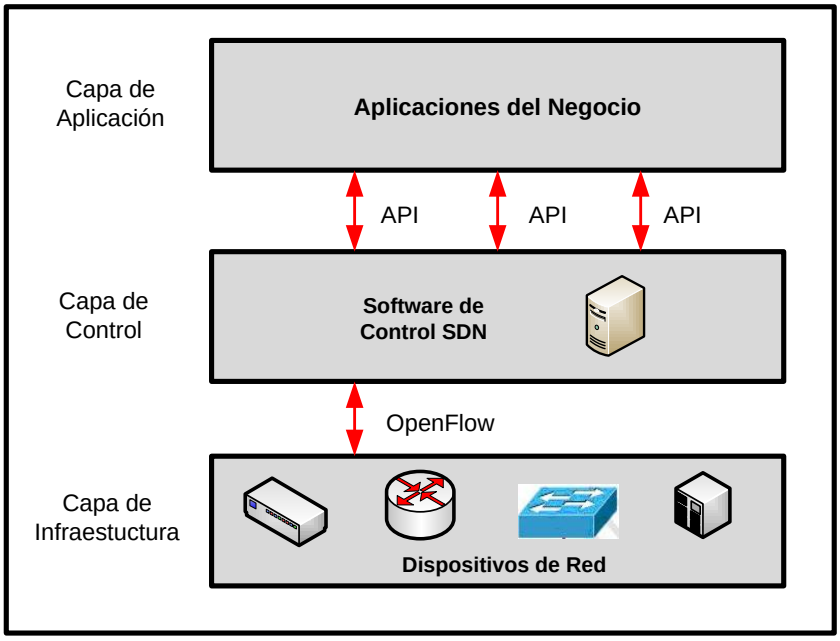

Figura 5. Arquitectura SDN

Los componentes fundamentales de la arquitectura SDN son:

- ✓ Un controlador SDN que se comunica con los switches y que mantiene una visión global de la red. En el controlador se generan todas las políticas de tratamiento de la información. La arquitectura SDN soporta el uso de un conjunto de APIs que hacen posible la implementación de servicios a través del *software* de control, tal es el caso del enrutamiento, requerimientos específicos de optimización de procesamiento y almacenamiento (*cloud computing*), calidad de servicio, entre otros.

- ✓ Un canal de comunicación seguro, como *Secure Sockets Layer* (SSL), el cual se encarga de conectar el software de control y el switch a través del protocolo *OpenFlow*.

- ✓ Un switch que soporte el protocolo *OpenFlow*. Cada *switch* tiene un conjunto de tablas que son usadas para la administración de flujos. Se define tres tipos de tablas:

  Flow Table.- relaciona los paquetes de entrada.

  Group Table.- contiene acciones que afectan a uno o más flujos.

  Meter Table.- puede desencadenar algunas acciones de performance de un flujo.

La arquitectura SDN ha sido realizada de tal forma que permite la flexibilidad con dispositivos de red de diferentes capas, ya sean switches (capa 2), routers (capa 3) ó dispositivos de conmutación a nivel de capa transporte (capa 4). SDN solamente encuentra un dispositivo de *networking* y lo administra. Existen algunos modelos que se ajustan a las necesidades del usuario, los cuales son detallados a continuación.

2.2.2 Modelos

SDN presenta cinco tipos de modelos de diseño[1], como se puede visualizar en la figura 6.

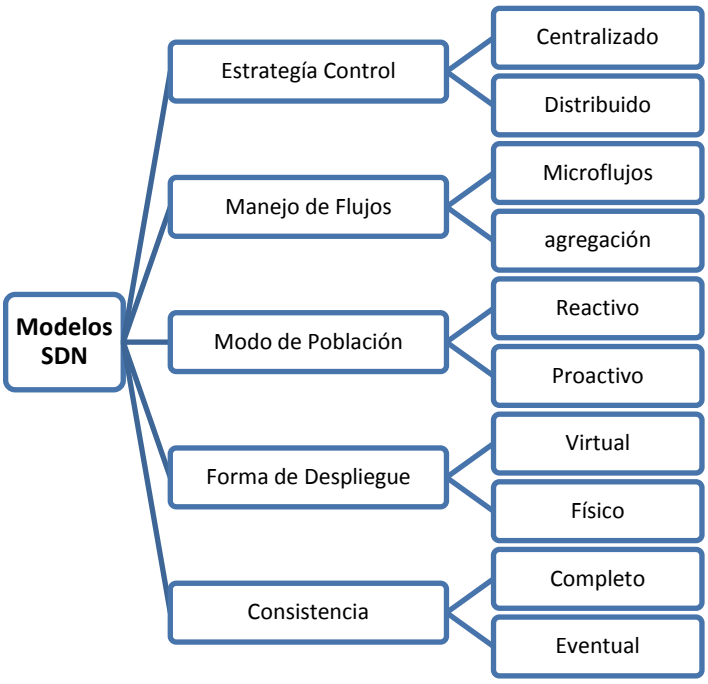

Figura 6. Modelos de Diseño SDN

Por la estrategia de control

- ✓ Centralizado.- existe un solo controlador, al cual se conectan todos los switches de la red (Figura 7a).

- ✓ Distribuido.- existe más de un controlador en la red, a los cuales se conectan los switches (Figura 7b).

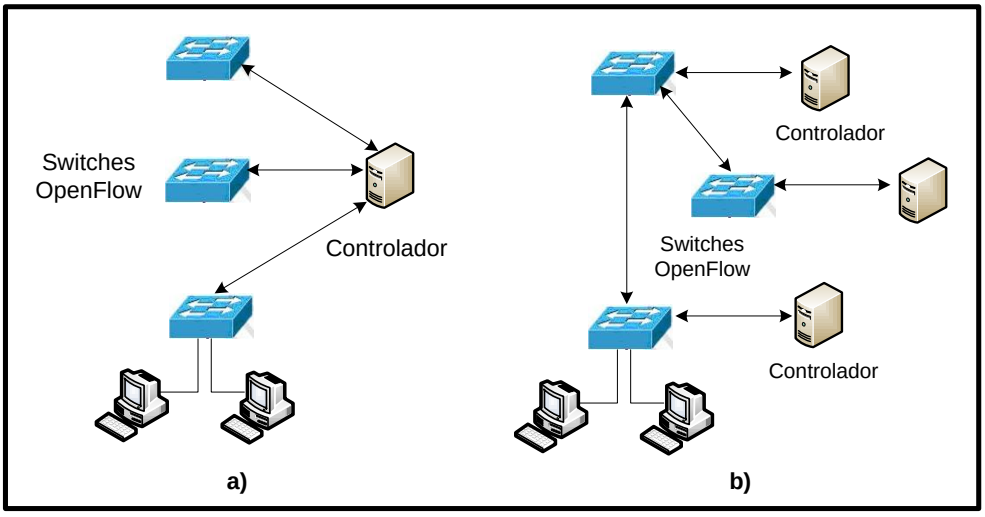

Figura 7. Modelo SDN por Estrategia de Control

Por el Manejo de Flujos

- ✓ Por microflujos: cada *flow* es tratado individualmente, por tanto cada flujo tiene su entrada en la tabla de flujos. Se tiene un control más fino de la red.

- ✓ Por agregación.- una entrada cubre un grupo de flujos, con lo que se denomina entrada comodín. La tabla de flujos contiene una entrada por cada grupo. Es utilizado en para grandes cantidades de flujo.

Por el Modo de Población

- ✓ Reactivo.- el primer paquete del flujo despierta al controlador para la inserción de la entrada de flujo en la tabla. La ventaja es que se tiene un uso eficiente de la tabla de flujos. Las desventajas son el tiempo adicional que se requiere por cada entrada de flujo y que si se pierde la conexión se limita la comunicación.

- ✓ Proactivo.- el controlador llena la tabla de flujos del switch con anterioridad, con lo cual no existe tiempo adicional para la instalación de flujos en la tabla y si la conexión se pierde el tráfico no se interrumpe. La desventaja es que necesita reglas de forma grupal.

Por su Forma de Despliegue

- ✓ Virtual.- asume la configuración de un switch dentro de un host. Es posible modificar arbitrariamente las capacidades, tal es el caso de la memoria, procesamiento, etc. La desventaja es que se encuentra limitado por el hardware sobre el que corre.

- ✓ Física.- usa un dispositivo físico que viene incorporado con la tecnología.

Por su consistencia

- ✓ Completamente consistente:.- se tiene certeza del estado de la red.
- ✓ Eventualmente consistente.- la convergencia es más lenta, por tanto en ciertos momentos no se tiene certeza del estado de la red.

2.2.3 Stack SDN

El Stack[3] SDN está dividido en una serie de capas que proporcionan servicios específicos para el funcionamiento de una arquitectura SDN, como se puede ver en la figura 8.

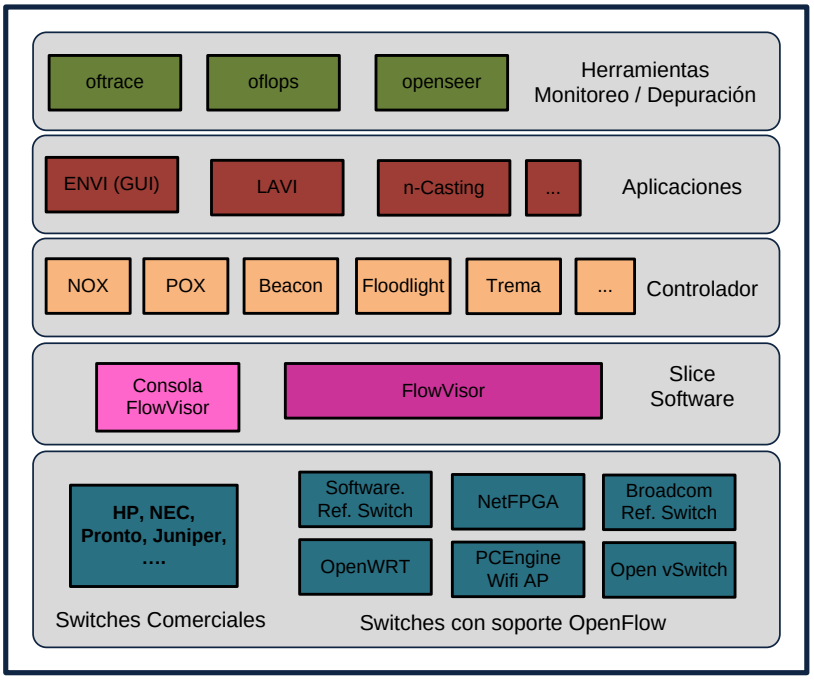

Figura 8. Stack SDN

Dentro de los switches para el despliegue de una arquitectura SDN se distinguen dos grandes grupos:

- ✓ Switches comerciales.- son aquellos que en su hardware viene incorporado el soporte para *OpenFlow*, como por ejemplo Pronto, NEC PF5240, Cisco 3750-X, etc.

- ✓ Switches con soporte *OpenFlow*.- estos switches se basan en software instalado en una máquina que funciona con Linux. Entre los más conocidos se tiene:
  Open vSwitch.- es un switch virtual de código abierto, creado por Nicira y su código fue escrito con python y C.
  Switch NetFPGA.- switch con soporte *OpenFlow* basado en la herramienta NetFPGA, la cual es una plataforma que permite a investigadores y estudiantes experimentar con hardware *de networking* a velocidades en el orden de los *Gigabits*.
  OpenWRT.- es una herramienta de código abierto que proporciona un marco de referencia para la creación de aplicaciones sin la necesidad de crear una imagen

[3]http://www.openflow.org/wp/openflow-components/

completa de *firmware*, a través del uso de WebUI y CLI (Command Line Interface).

En cuanto a los controladores actualmente existen una gran variedad y escritos en diversos lenguajes de programación, tal es el caso de C, java, python, ruby, etc. Entre los más utilizados se puede nombrar:

- ✓ NOX.- es el primer controlador para *OpenFlow*, escrito en C++ y python y con soporte para la versión *OpenFlow* 1.0. El uso de este controlador ha sido desplazado por POX.
- ✓ POX.- controlador escrito totalmente en Python, con mejor desempeño comparado con NOX y que brinda soporte para *Open vSwitch*.
- ✓ Beacon.- es un controlador basado en el lenguaje Java con funcionalidades multiproceso y que permite cambios en producción.
- ✓ BigSwitch.- es un controlador basado en Beacon que incorpora un CLI amigable para la administración centralizada de una red *OpenFlow*.

FlowVisor es una herramienta especial para controladores *OpenFlow* que actúa como proxy transparente entre los switches y múltiples controladores. Proporciona de cierta forma servicios de virtualización de redes. FlowVisor crea un *slice* de los recursos de la red y los asigna a un diferente controlador. Un *slice* puede ser definido como la combinación de puertos de switch (capa 1), direcciones ethernet fuente o destino (capa 2), direcciones IP (capa 3) y puertos de origen o destino TCP /UDP (capa 4). FlowVisor permite el aislamiento entre cada *slice* y en consecuencia entre cada controlador. Un *slice* no puede controlar el tráfico de otro.

El *stack* SDN es fruto de las diferentes investigaciones y proyectos que se han venido desarrollando en función de alguna necesidad específica. Cabe recalcar que el protocolo que se utiliza por excelencia junto a SDN es *OpenFlow*, el mismo que se ha utilizado en el presente trabajo.

2.3 Protocolo OpenFlow

OpenFlow nace por la necesidad de redes programables. McKeown et al. en [15] propone a *OpenFlow* como una vía para que los investigadores realicen pruebas con protocolos experimentales en las redes que utilizan en el día a día.

Lo que se intenta es dar respuesta a algunas preguntas, la primera ¿Cómo se va a conseguir la comodidad del administrador de red con el uso de equipos experimentales?; la segunda ¿Cómo los investigadores podrán controlar una porción

de la red local sin producir interrupciones? y la tercera ¿Qué funcionalidades son necesarias en los switches para poder experimentar?. Como respuesta nace el protocolo *OpenFlow*.

OpenFlow es el primer protocolo para redes definidas por software SDN, el cual facilita la programabilidad de la red, mediante la configuración, gestión y el control de los flujos desde un software centralizado. *OpenFlow* permite particionar el tráfico, escoger la mejor ruta y mejorar las formas en que los paquetes son procesados. Se enfoca en la creación de nuevas formas de enrutamiento, seguridad, control de tráfico, esquemas de direccionamiento, entre otros.

OpenFlow se basa en un switch Ethernet, el cual contiene una tabla de flujo interna y una interfaz estandarizada para poder añadir o remover las entradas de flujo. Se ha pasado de la estructura tradicional de un switch (figura 9a) a una estructura más abierta (figura 9b) con soporte para *OpenFlow*.

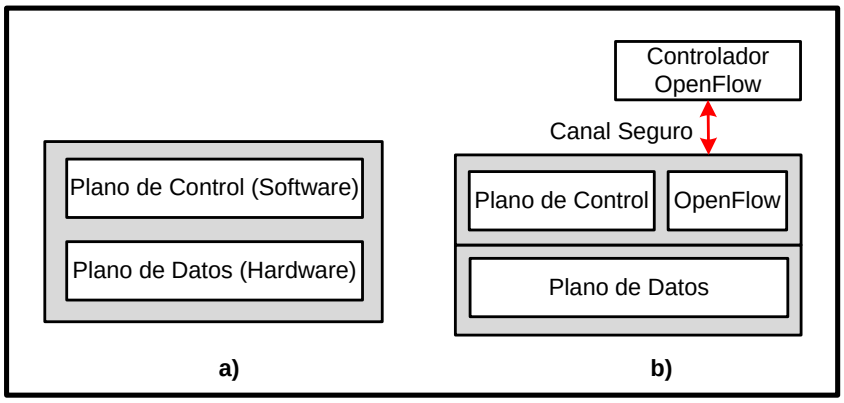

Figura 9. a) Switch tradicional. b) Switch *OpenFlow*

Por un lado se presenta un switch Ethernet tradicional, el cual tiene incluido en su estructura los planos el de control y el de datos. El plano de control es el que se encarga del intercambio de información (rutas, topología, estado) y del comportamiento con sus vecinos. El plano de control incluye funciones de configuración del sistema, gestión, intercambio de información de la tabla de enrutamiento y que indica al plano de datos que debe hacer con los paquetes entrantes. El plano de datos es el que se encarga del procesamientos de los datos en función de la información de sus tablas. Cualquier cambio se debe realizar a nivel de hardware, por ejemplo las rutas configuradas en un equipo.

Del otro lado se tiene el switch con soporte *OpenFlow*, en el cual la inteligencia de la red se encuentra en un dispositivo externo (controlador). Un switch *OpenFlow* proporciona una arquitectura lógica común que permite la comunicación entre un switch y un

controlador a través de un canal seguro. Todo dispositivo que tenga soporte para *OpenFlow* podrá ser controlado por el *software* controlador sin importar el fabricante del equipo de *networking*. El uso de *OpenFlow* promete:

- ✓ Manejo más fácil y eficiente de una red.
- ✓ Interacción dinámica de la red, es decir adaptabilidad.
- ✓ Separación del tráfico de datos del de control.
- ✓ Aceleración tecnológica.
- ✓ Una plataforma abierta de programación.

OpenFlow puede ser comparado con el conjunto de instrucciones de un CPU. El protocolo define un conjunto de primitivas básicas que son utilizadas por el software externo, en este caso el controlador, para programar o definir el comportamiento de reenvío de la información. Con *OpenFlow* se introduce una nueva forma de funcionamiento de un dispositivo de *networking*. La primera versión la 0.1 fue publicada en el año 2007 y ha ido mejorando continuamente, hasta la última versión la 1.3.1 que corresponde al 06 Septiembre del 2012 y sobre la cual se hace mención.

Un switch *OpenFlow* consta de tres elementos (figura 10) los cuales son:

1. Una o más tablas de flujo (correspondencia de cada entrada con la forma de procesamiento).
2. Un canal seguro (Establecido entre el switch y el controlador)
3. El protocolo *OpenFlow* (Mecanismo para que las entradas de las tablas puedan ser definidas externamente).

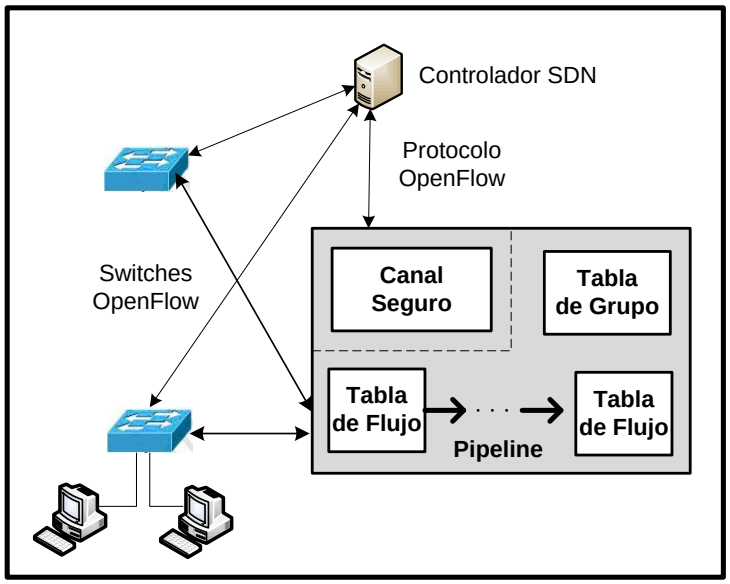

Figura 10. Componentes *OpenFlow*

OpenFlow utiliza el concepto de *flows* para identificar un tipo de tráfico que es predefinido con las reglas programadas por el controlador, ya sea de forma estática o dinámica. Por ejemplo un tipo de *flow* son las aplicaciones http, todos los paquetes que se dirigen hacia un mismo destino, etc. Entre las acciones que se pueden ejecutar se puede nombrar:

- ✓ Permitir / denegar flujos.
- ✓ Enrutar flujos.
- ✓ Aislar Flujos.
- ✓ Remover Flujos
- ✓ Convertir un flujo en privado.

Todas estas acciones son ejecutadas a través de las instrucciones guardadas en las diferentes tablas de flujo. A continuación se detalla los componentes y terminología de *OpenFlow*.

2.3.1 Puertos OpenFlow

Un switch *OpenFlow* define tres tipos de puertos:

- ✓ Puertos físicos.- corresponden a la interfaz de hardware del switch. En ambientes virtualizados representan los puertos creados virtualmente sobre el switch virtual. La asignación de una interfaz con un puerto es unívoca.
- ✓ Puertos lógicos.- no tienen correspondencia a un puerto físico como tal y son utilizados en métodos *non-OpenFlow* (túneles, agregación). Un puerto lógico puede estar relacionado a varios puertos físicos. La diferencia fundamental entre un puerto lógico de un físico, es que el primero tiene un campo de metadata[4] extra llamado Tunnel-ID.
- ✓ Puertos reservados.- son puertos para acciones específicas, como por ejemplo el envío de información al controlador, inundaciones, etc. Algunos de estos puertos reservados son requeridos en comunicaciones *OpenFlow*, entre estos el puerto ALL, puerto *controller*, puerto *table*, puerto *in_port* y el puerto *any*.

2.3.2 Tablas OpenFlow

Existen tres tipos de tablas:

- ✓ Tabla de flujo.- la cual permite relacionar los paquetes entrantes con un flujo y el conjunto de acciones específicas que debe llevar a cabo. Puede existir una o más tablas de flujo las cuales funcionan en un *pipeline*.

[4]Valor de registro usado para llevar información de una tabla a otra.

- ✓ Tabla de grupo.- consiste de un grupo de entradas. Un grupo representa un conjunto de acciones para inundación u operaciones de reenvío más complejas (*multipath, link aggregation*). Permite el reenvío de múltiples entradas de flujo hacia un solo identificador, similar a un *gateway*. Este proceso permite que acciones de salida comunes sean tratadas de forma más eficiente.

- ✓ Tabla Meter.- puede desencadenar algunas acciones de performance de un flujo, como calidad de servicio.

Cada entrada de la tabla de flujo está formada por seis componentes, como se visualiza en la figura 11.

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

Figura 11. Componentes Tabla de Flujo

- ✓ *Match Fields*.- representa el conjunto del paquete (información), el puerto de entrada y de manera opcional del *metadata*.
- ✓ *Priority*.- determina la prioridad de la entrada de flujo.
- ✓ *Counters*.- contadores de los paquetes relacionados que es actualizado cuando se encuentran coincidencias. Por ejemplo el número de bytes recibidos por puerto, número de paquetes descartados, etc.
- ✓ *Instructions*.- acciones que serán ejecutadas si los paquetes son relacionados con alguna entrada de flujo de la tabla.
- ✓ *Timeouts*.- tiempo máximo de espera antes de que un flujo caduque (tiempo de desocupación).
- ✓ *Cookie*.- es un valor usado por el controlador para modificaciones del flujo, filtrar estadísticas, etc. Este valor no es usado cuando se está procesando paquetes.

Una entrada se identifica en la tabla de flujo por el conjunto de los dos primeros campos: *March Fields* + *Priority*.

Para remover entradas de flujo existen dos métodos:

- ✓ Por el requerimiento del controlador.
- ✓ Por los mecanismos de expiración de flujos del switch.

Cada entrada de datos tiene un *idle_timeout* y un *hard_timeout*. Un campo *hard_timeout* diferente de cero causa que la entrada de flujo sea removida después de un tiempo independientemente de si existen paquetes que hayan sido relacionados. Un campo

idle_timoeout diferente de cero permite que la entrada de flujo sea removida cuando no existen paquetes relacionados.

Un switch *OpenFlow* puede tener una o varias tablas de flujo que son organizadas en un *pipeline*, como se puede visualizar en la figura 12. El *pipeline* describe el proceso de decisión y ejecución de un paquete que ingresa a un switch *OpenFlow*.

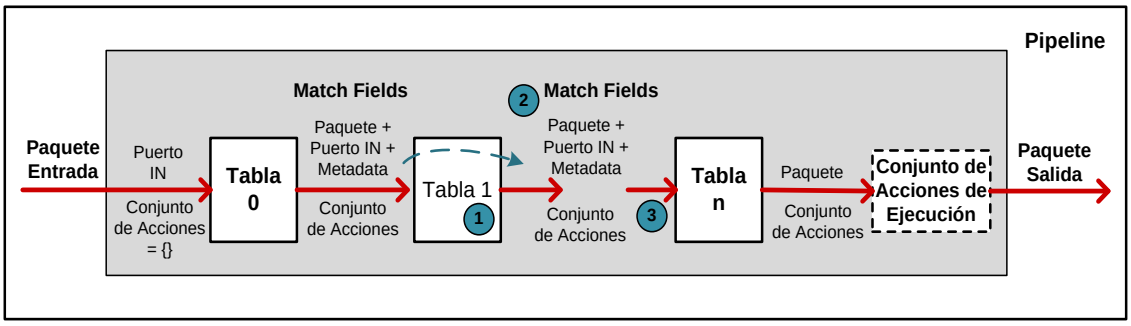

Figura 12. Pipeline *OpenFlow*

En el proceso de decisión cada switch tiene una o varias tablas, en las cuales irá realizando el siguiente proceso:

1. En la tabla de flujo respectiva se busca la entrada de flujo (con la prioridad más alta) que coincide con el paquete entrante. En caso de no existir coincidencia y si no existe la entrada *table-miss,* el paquete será descartado. Si existe coincidencia del paquete con una entrada de flujo, se realizarán cualquiera de las siguientes acciones: enviar el paquete al controlador ó enviar el paquete a otra tabla de flujo del *pipeline* ó descartar el paquete.

2. Se aplican el conjunto de instrucciones definidas.
   - ✓ Modificar los paquetes y actualizar los campos coincidentes (Match Fields)
   - ✓ Actualizar el conjunto de acciones.
   - ✓ Actualizar el *metadata*.

3. Enviar el dato coincidente y el conjunto de acciones a la siguiente tabla. Cuando el proceso haya terminado el paquete será dirigido a un puerto de salida, el conjunto de acciones acumuladas son ejecutadas y luego el paquete es encolado para la salida.

Una *pipeline OpenFlow* contiene múltiples tablas de flujo, y a su vez cada tabla contiene varias entradas. Las tablas son numeradas secuencialmente y el proceso empieza siempre desde la tabla 0. Se evalúa los datos ingresados con la primera tabla, las otras tablas serán usadas dependiendo de la salida de la predecesora (*match fields*). El

proceso que sigue un paquete dentro del *pipeline OpenFlow* es el que se detalla en la figura 13.

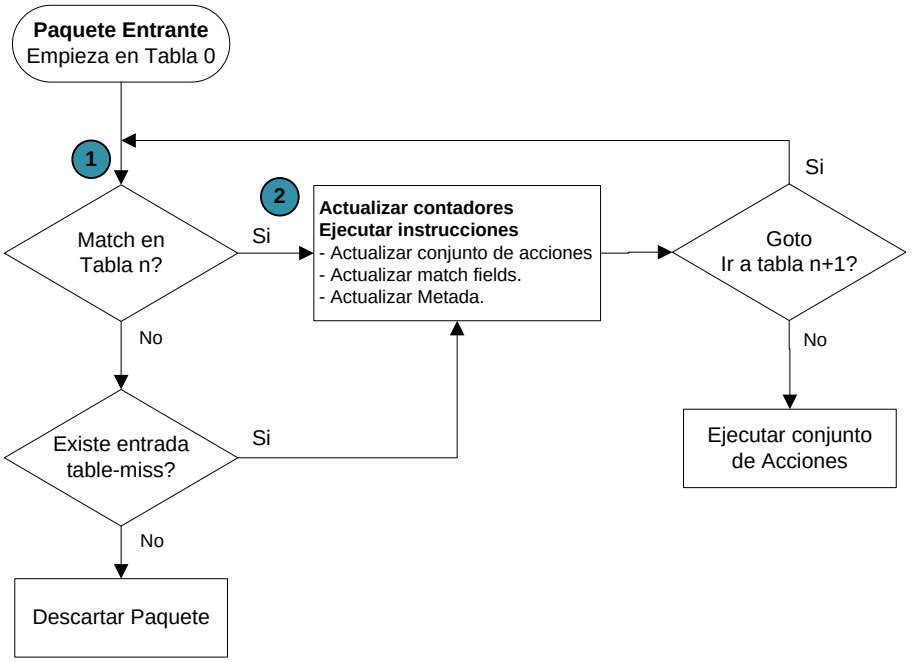

Figura 13. Diagrama de flujo Pipeline

Cuando un paquete ingresa se evalúa si existe coincidencia con alguna de las entradas de la tabla 0 (Paso 1). Existen dos posibilidades:

a) Si existe coincidencia:
   - ✓ Se actualizarán los contadores y se ejecutarán las instrucciones (Paso 2).
   - ✓ Luego se evaluará si se debe ir a la siguiente tabla (n+1). En el caso positivo se regresará al paso 1. En el caso negativo se ejecutarán el conjunto de acciones, terminando así el proceso.

b) De no existir coincidencia se evaluará si existe la entrada *table-miss*. En el caso positivo se pasará al paso 2. En el caso negativo se descartará el paquete.

La entrada *table-miss* contiene las instrucciones de tratamiento para paquetes que no tienen coincidencia con alguna entrada de la tabla de flujos. Las opciones pueden ser, enviar hacia otra tabla, al controlador, etc. Este tipo de entrada tiene la prioridad más baja 0. El estándar recomienda que todas las tablas de flujo tengan una entrada *table-miss*. Si dicha entrada no existe por defecto los paquetes sin coincidencia serán descartados. Esta configuración puede cambiarse.

Existen dos tipos de switch, sobre los cuales se puede aplicar este procedimiento, estos son:

- ✓ *OpenFlow-only*.- este tipo de switch solamente soporta el modo de operación con *OpenFlow*, todos los paquetes son procesados con el *pipeline OpenFlow*.

- ✓ *OpenFlow-hybrid*.- este tipo de switch soporta tanto la operación con *OpenFlow* como la de un switch normal *Ethernet*. El switch provee los mecanismos para que sea tratado ya sea por el *pipeline OpenFlow* o por el *pipeline* normal.

2.3.3 Canal OpenFlow

El canal *OpenFlow* es la interfaz que conecta el switch *OpenFlow* con el controlador. A través de dicha interfaz el controlador puede administrar y gestionar el switch.

Un switch *OpenFlow* puede tener múltiples canales *OpenFlow* cada uno con un controlador diferente. Cuando el controlador se encuentra en la misma red que administra el switch, el controlador estará en modo *in-band*, caso contrario se encontrará en modo *out-band*.

El switch y el controlador usualmente se comunican a través a través de una conexión segura con *TLS*. Dicha conexión siempre es inicializada por el switch, luego el switch y el controlador se autentican mutuamente por medio del intercambio de firmas certificadas. La comunicación también se puede realizar a través de *plain TCP*, en este caso será necesario el uso de mecanismos de seguridad alternativo para proteger el canal *OpenFlow*.

2.3.4 Mensajes OpenFlow

El protocolo *OpenFlow* define tres tipos de mensajes: *controller-to-switch*, *asynchronous*, y *symmetric*, cada uno con sus sub categorías.

- ✓ *Controller-to-switch*.- este mensaje es inicializado por el controlador y puede o no requerir una respuesta del switch. Estos mensajes son usados para administrar o inspeccionar el estado del switch. Entre las actividades más comunes se tiene la solicitud de las capacidades de un switch (*feature request*), la modificación del estado de un switch, etc. El mensaje utilizado es el *packet-out*.

- ✓ *Asynchronous*.- es inicializado por el switch y sirve para reportar al controlador de los cambios o eventos del estado del switch (arribo de paquetes, cambios del estado del switch, errores). El mensaje más utilizado es *packet-in*, con el cual el switch envía un paquete al controlador cuando no encuentra una coincidencia en la tabla de flujo.

- ✓ *Symmetric*.- este tipo de mensaje puede ser inicializado tanto por el controlador como por el switch, se puede intercambiar mensajes *hello, echo* y *experimenter*. Por ejemplo el mensaje *hello* es utilizado en la primera conexión entre el switch y el controlador.

Una vez que se inicia la comunicación, el switch se conecta a todos los controladores y debe mantener conectividad con todos ellos. El rol por defecto de un controlador es *OFPCR_ROLE_EQUAL*, en el cual todos los controladores con ese estado tienen total acceso al switch. Por defecto el controlador recibe *asynchronous messages* del switch, luego el controlador envía comandos *controller-to-switch* para modificar el estado del switch. En todo este proceso el switch no realiza ningún tipo de arbitraje o intercambio de recursos entre los controladores. Para manejar esta deficiencia el controlador puede ser cambiado a modo *OFPCR_ROLE_SLAVE*, en el cual el controlador tiene acceso solo de lectura al switch, únicamente recibirá mensajes del estado de puertos (*Port-status messages*) y no podrá ejecutar comandos *controller-to-switch*.

Existe también el modo *OFPCR_ROLE_MASTER*, el cual funciona de manera similar que el rol *OFPCR_ROLE_EQUAL*, con la diferencia de que el switch verifica que sea el único controlador con dicho rol. Cuando se establece un controlador en el rol de máster el switch cambia a esclavos a los demás controladores.

Un controlador puede también verifica que tipo de *asynchronous messages* son enviados sobre el canal *OpenFlow*. Con esta característica el controlador máster puede deshabilitar algunas notificaciones y el controlador esclavo podrá habilitar las notificaciones que quiera monitorear.

Ahora en caso de desconexión del controlador se deberá realizar las configuraciones respectivas para que el switch entre en uno de los siguientes modos:

- ✓ *fail secure mode*.- en este modo los paquetes y mensajes destinados al controlador son descartados.
- ✓ *fail standalone mode*.- el switch pasará a funcionar como un switch *ethernet* convencional. Este modo es soportado solamente por switch híbridos.

El switch como ya se ha mencionado se encuentra en la capacidad de establecer comunicación con múltiples controladores. Sin embargo el proceso de *hand-over* se encuentra totalmente a cargo de los controladores.

Por defecto el canal *OpenFlow* entre un switch y un controlador es una conexión de red única, sin embargo el canal puede estar compuesto por una conexión principal y algunas auxiliares. Cada conexión desde el switch al controlador es identificada a través del *datapath ID* y de un *auxiliary ID*. Este tipo de conexiones son conocidas como auxiliares.

2.3.5 Beneficios y aplicaciones

Algunos de los demos y trabajos realizados, especialmente por la universidad de *Stanford* [5] son:

- ✓ Creación de nuevos switches.
- ✓ Pruebas para movilidad inalámbrica.
- ✓ Movilidad y migración de máquinas virtuales.
- ✓ Virtualización de redes.
- ✓ Balanceo de carga.
- ✓ Ingeniería de tráfico.
- ✓ *Spanning Tree Protocol* (STP)
- ✓ Data centers.
- ✓ Creación de nuevos controladores.
- ✓ Simuladores.
- ✓ Eficiencia energética en redes OpenFlow.

Muchas implementaciones de SDN son centralizadas lo que desemboca en un solo punto de falla y por tanto se compromete seriamente toda la red. La utilización de un solo controlador para un dominio no es una condición deseable debido a los problemas ya conocidos de escalabilidad, privacidad y de despliegue incremental, por tanto el IETF actualmente está trabajando en el desarrollo del protocolo SDNi *(interfacing SDN Domain Controllers*). El protocolo permite intercambiar información de enrutamiento entre controladores de diferente dominio. Entre las funciones de dicho protocolo se puede nombrar:

- ✓ Coordinar el envío de los datos del flujo tal cual en el origen, por ejemplo que cierto nivel de servicio pueda viajar en el flujo a través de múltiples dominios.

[5] http://yuba.stanford.edu/~casado/of-sw.html

- ✓ Intercambiar información para facilitar el enrutamiento a través de múltiples dominios SDN.

Otro de los retos que introduce SDN es la capacidad de recuperarse frente a fallos, para ello existe la posibilidad de tener varios controladores sobre una misma red. Esto ha introducido algunos inconvenientes para mantener consistencia entre los controladores ya que el protocolo *OpenFlow* no ofrece un mecanismo específico.

Fonseca et al. en [16] propone un mecanismo de replicación que provee una mayor capacidad de recuperación de la red frente a fallos, para lo cual se desarrolla un nuevo componente llamado *CPRecovery*. Dicho componente se basa en un mecanismo primario de *backup* en caso de de fallas de un controlador centralizado. La aportación de esta investigación es que provee una transición sin problemas en caso de fallas. Además proporciona consistencia de los flujos entre el controlador primario y uno o más controladores secundarios a través de la comunicación entre los dos controladores y el envío de mensajes de actualización.

2.4 Mininet

Mininet es un emulador que permite la creación de escenarios de red virtuales. Fue ideado con el objetivo de que cualquier persona, ya sea un estudiante o un investigador pueda descargar, correr, evaluar, explorar, realizar cambios sobre entornos previamente creados ó mejor aún diseñar sus propios escenarios de prueba. [12]. Una de las principales características de Mininet es que sus switches soportan el protocolo *OpenFlow* y por tanto permiten la experimentación de SDN.

Mininet se basa en las funcionalidades que presta la virtualización de un sistema operativo Linux, permitiendo la creación de cientos de nodos, ya sean estos switches, máquinas o controladores en un solo computador. La virtualización con Linux permite el funcionamiento de procesos individuales con interfaces, tablas ARP y de enrutamiento diferentes. La mayor parte del código de Mininet es desarrollada con Python y una pequeña parte con C. Adicionalmente Mininet usa una interfaz de línea de comandos (CLI) para crear, gestionar y compartir los escenarios creados. La última versión de Mininet la 2.0 corresponde a noviembre de 2012 y es de la cual se hace referencia.

Mininet se ha convertido en la herramienta preferida para emular redes definidas por software por las siguientes razones:

- ✓ Flexibilidad.- el usuario puede crear sus propios escenarios según su necesidad. De igual forma a través del controlador incluido puede generar las instrucciones necesarias para controlar la red.
- ✓ Fácil de desplegar y usar.- no requiere configuraciones especiales, con algunos comandos predefinidos el usuario puede generar su propia topología.
- ✓ Interactividad.- se puede realizar un seguimiento y gestión de los experimentos en tiempo real.
- ✓ Escalable.- permite el despliegue de escenarios complejos. El número de nodos que puede soportar asciende a los cientos o miles, todos ellos manejados por un solo equipo.
- ✓ Fácil de compartir.- Mininet permite compartir los escenarios creados de una forma fácil por tanto se promueve la colaboración.
- ✓ Se puede crear un escenario de forma rápida e independiente del sistema operativo ya que existe la posibilidad de trabajar directamente en Linux o descargar la imagen *OVF* de la máquina virtual para probarla sobre VirtualBox o cualquier otro programa similar.
- ✓ Ofrece ancho de banda para pruebas en el orden de los 2 Gbps[17].
- ✓ La inclusión de un CLI permite realizar pruebas en los escenarios.
- ✓ El código desarrollado en un controlador *OpenFlow*, puede ser probado fácilmente en un escenario real sin ningún cambio.

A continuación se presenta algunos lineamientos para la creación y gestión de escenarios en Mininet.

2.4.1 Creación

Existen dos opciones para poder utilizar Mininet:

1) Descargar directamente la imagen de la máquina virtual Mininet/Ubuntu cuyo tamaño aproximado es de 1GB. Para utilizar la imagen se necesitará un software de virtualización, tal es el caso de VMware o VirtualBox. Se recomienda seguir la guía del *OpenFlow Tutorial* [18].

2) Directamente sobre un sistema operativo Linux. Se recomienda la versión Ubuntu 12.10, la cual tiene soporte para *Open vSwitch*.

Una vez instalado Mininet se podrá crear topologías a través del comando:

$ sudo mn *parámetros*

Los diferentes parámetros de inicialización de un escenario en Mininet permiten definir características tales como: el número de hosts virtuales., el número de switches *OpenFlow*, definir si el controlador es remoto, etc. Una vez que se crea el escenario se tendrá acceso al CLI de Mininet (*Mininet*>).

Los comandos básicos para el acceso a los componentes de un escenario se detallan en la tabla 1.

Tabla 1. Comandos Básicos Mininet

Comando	**Función**
Help	Presenta una lista con los comandos disponibles.
Nodes	Despliega la lista de nodos de la topología.
Net	Despliega los enlaces.
Dump	Visualización de la información de volcado de todos los nodos.
h1 ifconfig	Permite ver la configuración del host *h1*.
Xterm *h1*	Creación de un terminal virtual de *h1*.
Pingall	Test de conectividad.

Todos estos comandos vienen integrados por defecto con Mininet, al igual que un conjunto de escenarios de prueba.

2.4.2 Arquitectura

Mininet permite la creación de redes virtuales para lo cual emula hosts, enlaces, switches y controladores.

- ✓ Host.- se encuentran contenidos dentro del *network namespace*. Cada uno cuenta con sus propias interfaces, puertos y tablas de enrutamiento. Un host representa la creación de un proceso *shell* y tiene una conexión hacia el supervisor (*pipe*).
- ✓ Enlace.- emula una conexión entre dos interfaces.
- ✓ Controladores.- permite el control de la red y puede estar ubicado dentro de la máquina virtual de Mininet, fuera de ella ó inclusive en un entorno *cloud*.
- ✓ Switches.- switches con soporte *OpenFlow*.

La arquitectura de Mininet define los componentes que se visualizan en la figura 14.

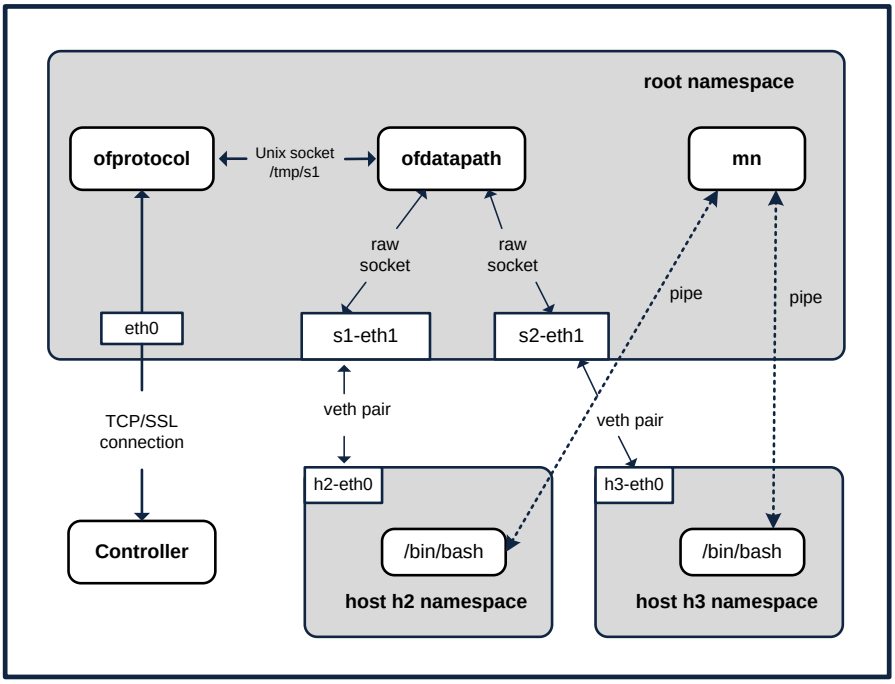

Figura 14. Arquitectura Mininet

El controlador que se conecta a los switches a través del protocolo *OpenFlow* sobre una conexión SSL.

Se tendrá un proceso supervisor (mn) y algunos subprocesos.

- ✓ */bin/bash*
- ✓ *veth* (Enlaces virtuales para la conexión con los hosts)
- ✓ *pipes* (comunicación con el supervisor).

Además cuando se crea un nuevo escenario se tendrá dos tipos de *namespaces*:

- ✓ El *root namespace*.- el cual cuenta con una interfaz para la administración y los switches *OpenFlow* (*Open vSwitch*).
- ✓ *Network namespace* .- contiene los procesos de creación de los hosts.

El switch utilizado en Mininet es *Open vSwitch*, el cual es descrito a continuación.

2.5 Open vSwitch

Open vSwitch (OVS) es una herramienta de código abierto que permite la creación de switches para ambientes de virtualización. OVS permite la comunicación entre

diferentes máquinas virtuales con mejores prestaciones que el tradicional *bridge*. Capacidades como el filtrado de paquetes, control de admisión de flujos, entre otros.

OVS está dirigido para sistemas de virtualización multi servidor, en donde los ambientes son dinámicos y donde es necesario la automatización, como por ejemplo en escenarios tipo *cloud*. OVS es el switch utilizado por defecto en plataformas de virtualización como Citrix XenServer[19] u OpenStack pero también puede utilizarse con sistemas de virtualización como KVM, Xen, etc.

OVS permite una mejor utilización de las capacidades del kernel Linux. Entre las principales ventajas del uso de OVS se pueden nombrar:

- ✓ Proporciona soporte para netFlow y sFlow.- permiten obtener información de la base de datos del switch (OVSDB) [20] para detectar cambios y responder a los mismos si así lo amerita. Por ejemplo la detección de malware en la red.
- ✓ Habilita el control programado de la infraestructura de red.
- ✓ *OVS* introduce métodos para un mejor mantenimiento y la correcta gestión de etiquetas para identificación de las diferentes VMs en entornos distribuidos.
- ✓ Mejora la seguridad de la red.
- ✓ Soporte para túneles GRE, que pueden ser usados para conectar una VM en diferentes data centers.
- ✓ Mayor granularidad, por ejemplo con ACLs.
- ✓ Permite el control de tráfico a través del descubrimiento de estados de enlaces con diferentes protocolos (*OSPF, CDP, LLDP*).
- ✓ Mayor escalabilidad (miles de puertos virtuales).
- ✓ Soporte para servicios *multicast* y *broadcast*.
- ✓ Permite el balanceo de carga entre diferentes enlaces y *mirroring*.
- ✓ Introduce la posibilidad de establecer niveles de calidad de servicio en cuanto a ancho de banda, latencia, tráfico prioritario, etc.
- ✓ Soporte para Vlans.
- ✓ Open vSwitch se enfoca en la necesidad de tener mecanismos de control dinámicos y automatizados a gran escala, que permitan mantener el kernel lo más pequeño posible [21].

Todos estas ventajas se derivan de su estructura, como se puede visualizar en la figura 15.

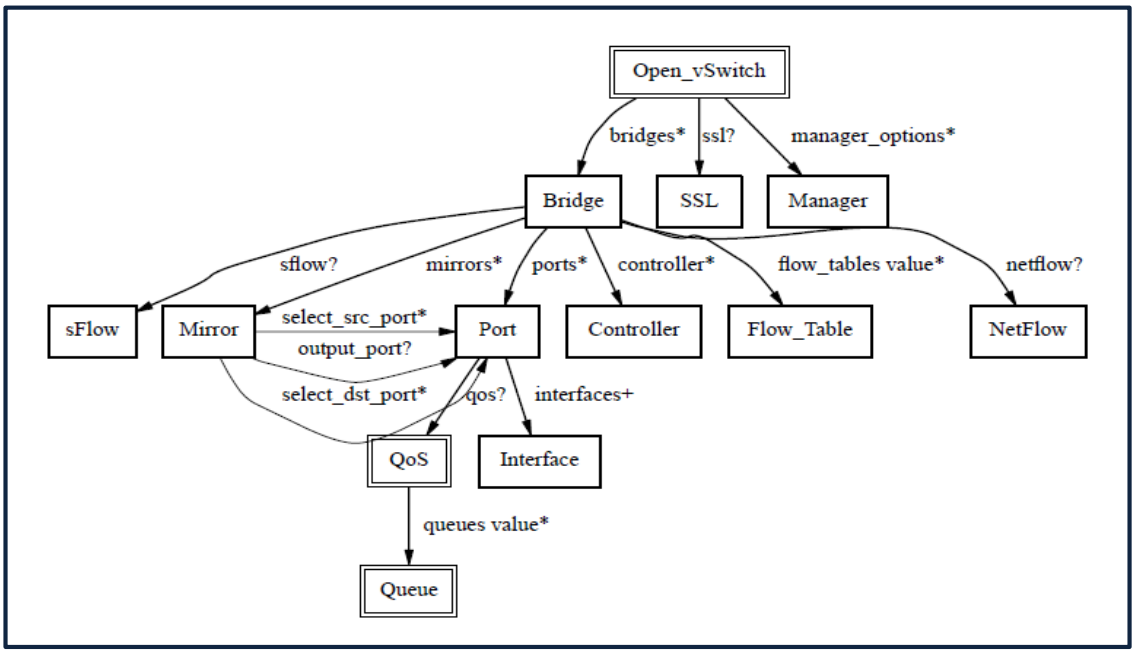

Figura 15. Diagrama de relaciones Open vSwtich

OVS trae consigo algunos inconvenientes el primero es que puede convertirse en un cuello de botella por la gran cantidad de tráfico que maneja. Además es un único punto de falla.

2.5.1 Protocolos y servicios soportados

Open vSwitch proporciona soporte para los siguientes protocolos y servicios [22]:

- ✓ Protocolo *OpenFlow*.
- ✓ Protocolo LACP (Agregación virtual de enlaces).
- ✓ Estándar 802.1Q.
- ✓ Estándar 802.1ag (Administración de fallas de conectividad).
- ✓ Protocolo STP.
- ✓ Control y gestión de QoS.
- ✓ Soporte para HPSC qdisc (Hierarchical Fair Service Curve's control).
- ✓ Bonding y mirroring.
- ✓ Soporte IPv6.
- ✓ NetFlow, sFlow, SPAN, RSPAN (monitoreo y auditoría).
- ✓ Protocolos para tunneling (Ethernet sobre GRE, IPsec, etc).
- ✓ Compatibilidad con las funcionalidades de bridge de Linux.

Como se mencionó anteriormente OVS tiene soporte para el Protocolo *OpenFlow* y por tanto puede ser gestionado por un controlador de una arquitectura SDN. Existe una gran cantidad de controladores, de los cuales se detallará POX.

2.6 POX

POX es un controlador SDN, desarrollado en Python que permite programar y controlar switches que soportan el protocolo *OpenFlow*. POX es uno de los controladores más recientes sin embargo se está posicionando como el controlador preferido en SDN por características como las siguientes:

- ✓ Interfaz realizada en python. Python es un lenguaje de programación multiparadigma, es decir que permite varios estilos de programación tales como orientación a objetos o la programación funcional, lo cual hace de python la herramienta ideal para cualquier programador.
- ✓ Ejemplos de componentes reusables.
- ✓ Es de fácil despliegue.
- ✓ Al ser considerado el "*hermano*" del controlador NOX, soporta la misma interfaz de usuario y herramientas de visualización.
- ✓ Tiene un mejor rendimiento comparado con NOX en aplicaciones escritas en Python.
- ✓ Incluye soporte para Open vSwitch.

En la página oficial de POX[23] se puede encontrar información detallada acerca de su funcionamiento y componentes.

3 Diseño e Implementación

Como se mencionó en el primer capítulo el principal objetivo del presente trabajo es la presentación de escenarios virtuales que permitan interactuar con el protocolo *OpenFlow*. En este sentido un buen ejemplo a seguir es el *OpenFlow Tutorial,* el cual presenta los conceptos básicos del protocolo *OpenFlow* y los componentes de la arquitectura SDN, a través del uso de la herramienta Mininet. La topología usada en dicho tutorial se visualiza en la figura 16.

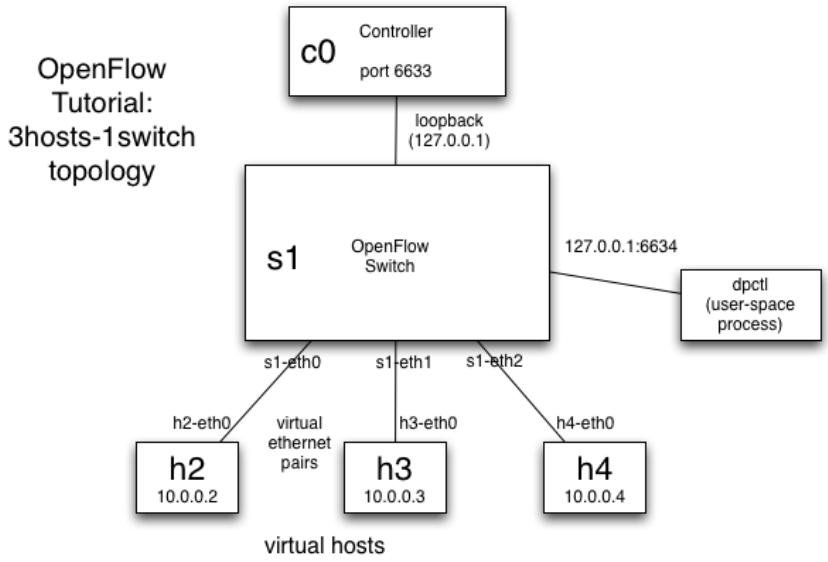

Figura 16. Topología OpenFlow Tutorial[18]

La topología *OpenFlow Tutorial* presenta un esquema en el cual se conectan 3 hosts virtuales (h2, h3, h4) a una misma subred, la 10.0.0.0/24, a través de un switch (s1) con soporte OpenFlow. Las políticas de control de flujos están a cargo del Controlador *c0*. De manera predeterminada el puerto designado para *OpenFlow* es el 6633.

3.1 Diseño

3.1.1 Requisitos iniciales

Para poder desplegar una topología con soporte *OpenFlow* se definen los siguientes requisitos iniciales:

1) Capacidad de desplegar entornos virtuales, para lo cual se utiliza la herramienta VNX. VNX, como ya se mencionó en el punto *2.1.2*, es una herramienta que permite la creación de entornos virtuales. Actualmente los escenarios generados con VNX utilizan el bridge normal del sistema operativo Linux, el cual no cuenta con soporte para *OpenFlow*. En consecuencia es necesario la integración de VNX con una herramienta que permita la generación de switches virtuales con soporte para *OpenFlow*. Como se muestra en la figura 17.

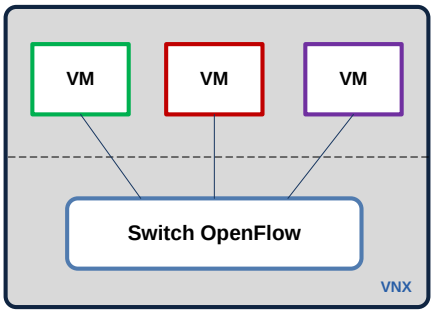

Figura 17. Escenario Virtual con Switch OpenFlow

2) Contar con dispositivos virtuales que tengan compatibilidad con *OpenFlow*, que realicen la función de controlador. En este sentido los dispositivos virtuales que actualmente se utilizan, no cuentan con soporte para OpenFlow o no tienen el software necesario para poder utilizar sus funcionalidades. Por tanto es indispensable adaptar los dispositivos para que puedan funcionar como controlador *OpenFlow*.

3) Herramientas que permitan la verificación de uso del protocolo *OpenFlow*. Una vez generado los escenarios será indispensable la verificación del tráfico *OpenFlow*. Al ser OpenFlow tan reciente la mayor parte de herramientas de control de tráfico y monitoreo no lo diferencian de los demás protocolos que atraviesan la red.

De forma general los requisitos iniciales se centran en la adaptación de los componentes de la red para que permiten la utilización del protocolo *OpenFlow*. Estos componentes se detallan a continuación.

3.1.2 Componentes

Los componentes básicos que intervienen en el escenario *OpenFlow* son:

- ✓ Switches.- el switch escogido tendrá soporte para *OpenFlow* y deberá funcionar en ambientes virtualizados.
- ✓ Software controlador.- el software controlador deberá tener capacidades de control y programación de flujos *OpenFlow*. Este deberá ser instalado en una de los hosts de los escenarios, de preferencia que cuente con una interfaz gráfica.

Como elementos complementarios se tienen los hosts que se conectarán a los switches *OpenFlow*, los cuales permitirán la realización de pruebas a través del uso de herramientas de control de tráfico y de conectividad de red.

3.1.3 Esquema de escenarios

Para el presente trabajo se proponen tres escenarios, partiendo de una topología básica, idéntica al del *OpenFlow Tutorial*. Para el segundo escenario se adiciona un segundo controlador y finalmente en el tercer escenario se presenta una topología con varias redes conectadas a través de un router.

Escenario 1.

Este escenario presenta una topología básica que consta de un switch *OpenFlow* y cuatro hosts, tres de ellos son consolas de Ubuntu 12.04 y el cuarto host presenta un sistema operativo Ubuntu 12.04 con interfaz gráfica y que hará las veces de controlador. En la figura 18 se observa el esquema del escenario 1.

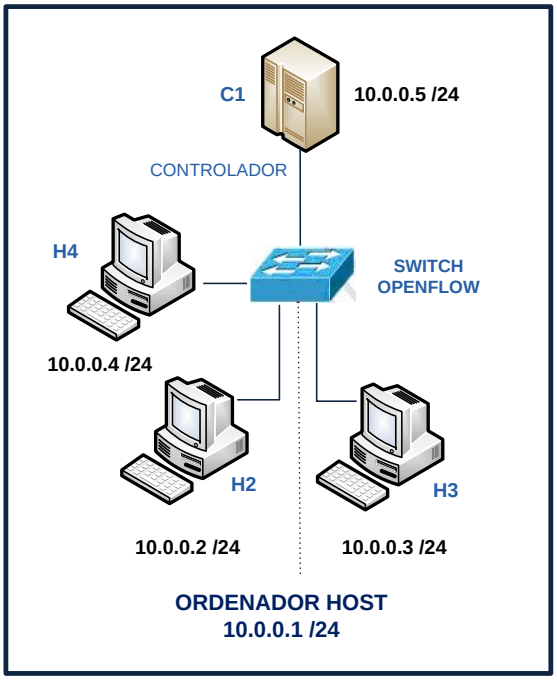

Figura 18. Escenario 1

Escenario 2.

El escenario dos presenta una ligera modificación del escenario uno, solamente se adiciona un nuevo host controlador, con lo cual la topología está compuesta por un switch *OpenFlow* y cinco hosts, dos de los cuales llevan a cabo la función de controladores y cuentan con interfaz gráfica del sistema operativo Ubuntu 12.04. Los hosts normales utilizan de igual forma Ubuntu 12.04 pero solo en modo consola En la figura 19 se puede visualizar el esquema del escenario 2.

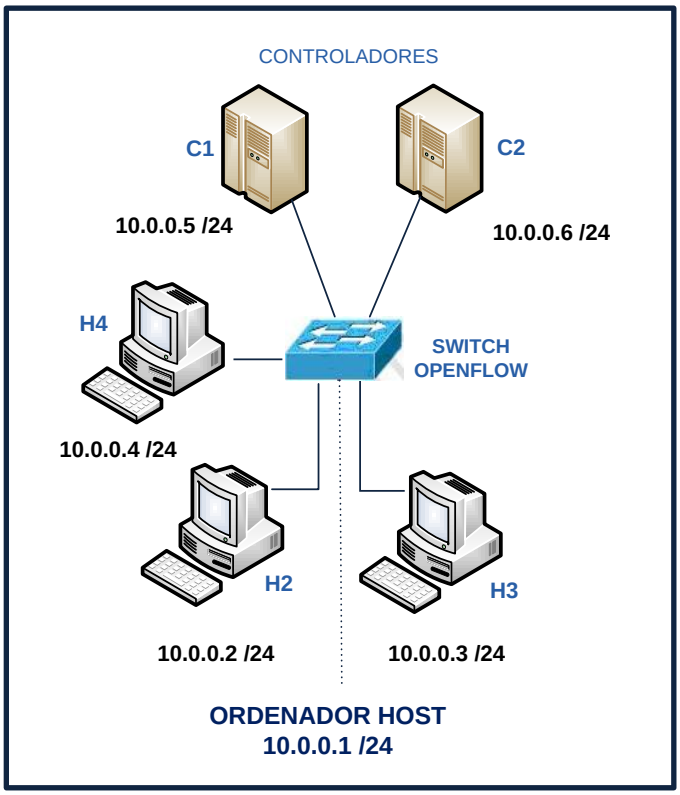

Figura 19. Escenario 2

Escenario 3.

El escenario tres presenta una estructura más compleja, compuesta por los siguientes elementos:

- ✓ 3 switches, cada uno controla una subred diferente.
- ✓ 2 controladores, ubicados en subredes diferentes.
- ✓ 2 hosts , ubicados de igual forma en subredes diferentes.
- ✓ 1 router que conecta las subredes.

Tanto el router como los hosts normales utilizan el sistema operativo Ubuntu 12.04 en modo consola, a diferencia de los dos hosts controladores que hacen uso de Ubuntu 12.04 con interfaz gráfica.

En la figura 20 se puede visualizar el esquema del escenario 3.

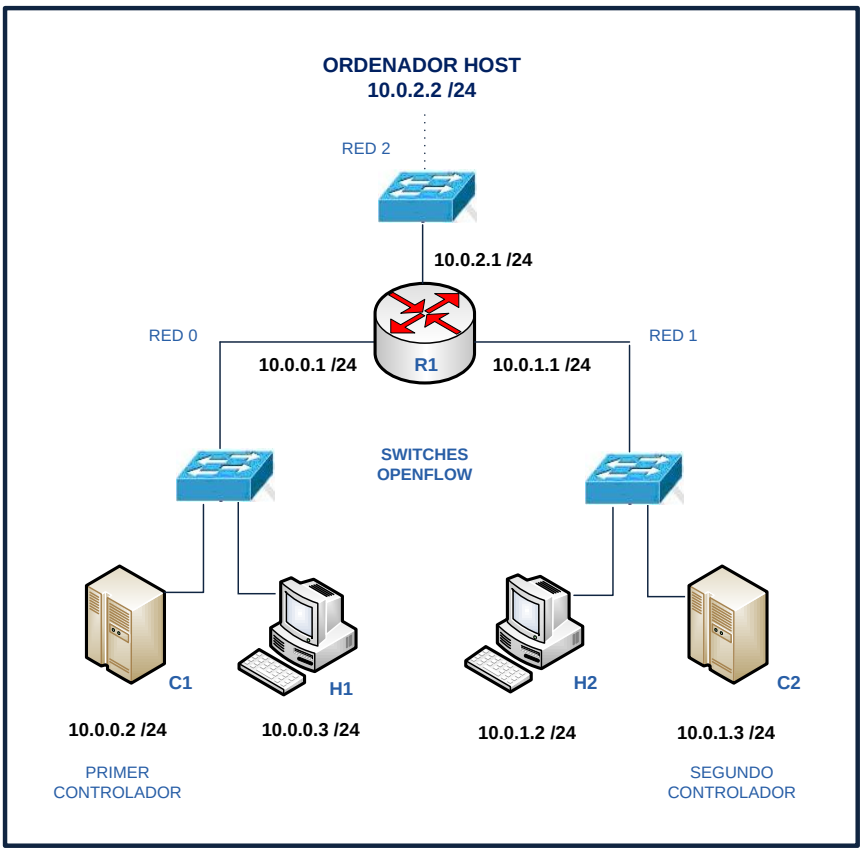

Figura 20. Escenario 3

Cabe recalcar que el sistema operativo de los dispositivos virtuales de los escenarios (hosts normales, routers o controladores) puede variar, siempre y cuando se ajuste a los sistemas operativos soportados por VNX. Además los escenarios se pueden cambiar o ampliar de manera sencilla gracias a las funcionalidades de la herramienta VNX.

Cada componente dentro de la estructura de los escenarios cumple con una función específica, y debe acoplarse correctamente para su funcionamiento. Dentro de este contexto, en los apartados siguientes, de diseño y pruebas se clarifica la forma en que se consigue el despliegue de los escenarios.

3.2 Implementación

Para el despliegue de un escenario virtual con VNX es indispensable el uso de un sistema operativo Linux en modo nativo. El sistema operativo escogido fue Ubuntu

12.04 sobre el cual se procedió a la instalación y configuración respectiva de la herramienta VNX [6].

En lo que respecta a las imagen del sistema operativo utilizado en las máquinas virtuales que actúan como controladores fue necesaria la modificación del *root filesystem* ubuntu-12.04-gui-v024, al cual se le agregó los paquetes y el software necesario para dicho propósito. Para el router y las máquinas hosts se utilizó el *root filesystem* ubuntu-12.04-v024 sin ninguna modificación. Los dos archivos originales se pueden descargar desde la página oficial del proyecto VNX.

Posteriormente se genera los archivos *.xml* correspondientes a los escenarios virtuales definidos en el punto *3.1.3*; se inicializa y se prueba el buen funcionamiento de dichos escenarios.

Con los escenarios listos y operativos funcionando de manera normal con VNX, se procede a la instalación y configuración de los elementos definidos en el punto 3.1.2, para que los escenarios funcionen con *OpenFlow*. Tal es el caso de la modificación del *root filesystem* que funcionará como controlador y la instalación de Open vSwitch, el cual reemplazará a la utilidad bridge del sistema operativo Linux.

3.2.1 Open vSwitch

Actualmente VNX trabaja de forma predeterminada con la utilidad bridge que viene por defecto integrada en el sistema operativo de Linux. En virtualización un bridge permite la comunicación entre diferentes máquinas virtuales. Para poder utilizar los escenarios se procede a reemplazar el bridge por un switch *OpenFlow*, tal es el caso de la herramienta Open vSwitch.

Al momento de la instalación se tuvo algunos inconvenientes al inicializar el servicio *openvswitch*, debido a que el módulo bridge y Open vSwitch no pueden ejecutarse al mismo tiempo, por tanto es necesario eliminar el módulo bridge, para lo cual se verifica si dicha utilidad se encuentra activa mediante el comando:

lsmod | grep bridge

De encontrase activa se procede a su eliminación:

rmmod brigde

[6] http://web.dit.upm.es/vnxwiki/index.php/Docintro

En algunos casos será necesario la actualización e instalación de algunos paquetes adicionales, como se observa en la figura 21.

```
# apt-get update

# apt-get install openvswitch-datapath-source

# module-assistant auto-install openvswitch-datapath

# apt-get install openvswitch-brcompat

# apt-get install openvswitch-common

# apt-get install openvswitch-switch
```

Figura 21. Paquetes para Open vSwitch

Primero se realiza una actualización de los repositorios del sistema, segundo se instala el módulo de compilación para el kernel, posteriormente se genera el módulo y finalmente se instalan los paquetes para compatibilidad con *bridge* de linux (*brcompat y common*) y el paquete *openvswitch*.

Luego se edita el archivo: */etc/default/openvswitch-switch*, en el cual se deberá agregar una línea de código con el siguiente formato:

brcompat=yes

Finalmente se hace un *restart* del servicio *openvswitch*:

 # /etc/init.d/openvswitch-switch restart

Y se comprueba que los servicios se encuentre operativos:

 # ps -ea | grep ovs

 # ovs-vsctl show

Si la instalación es correcta se podrá empezar a utilizar Open vSwitch, y se visualizará una pantalla similar a la de la figura 22.

```
root@SATELLITE:/usr/share/tfm# ovs-vsctl show
8d6fae9f-9314-450a-99ae-fd8382233cc6
    ovs_version: "1.4.0+build0"
root@SATELLITE:/usr/share/tfm#
```

Figura 22. Comprobación Open vSwitch.

La versión más actual de Open vSwitch es la 1.10.0, sin embargo la versión utilizada en el presente trabajo es la 1.4.0; debido a que ésta se instaló al momento de la descarga directa del paquete.

Este procedimiento podrá variar dependiendo del sistema operativo y de los aplicativos que se tenga instalado en el mismo. Adicionalmente se puede instalar un controlador básico de OVS con el comando:

apt-get install openvswitch-controller

Actualmente la versión Ubuntu 12.10 tiene soporte incluido para Open vSwitch.

3.2.2 Controlador POX

Una vez que se cuenta con switches que soporten *OpenFlow*, queda por modificar el *root filesystem* de la máquina virtual que contiene el software controlador. En el presente trabajo se utiliza el controlador POX para las pruebas de funcionamiento, sin embargo se ha instalado dos controladores adicionales, NOX y Beacon. Con esto se tiene dos entornos de programación, Java y Python, a total disposición del usuario para que realice código de control para redes *OpenFlow*.

En primera instancia se debe realizar el procedimiento de modificación de un *root filesystem*, especificado en la página de VNX[7]. Antes de la instalación de OVS se probó dicho proceso y no existió inconvenientes; sin embargo luego de su instalación se presentó algunas dificultades, las cuales se detallan a continuación.

- ✓ No es posible iniciar individualmente el *root filesystem*. El mensaje indica que la red "*default*" no existe. Para dar una pronta solución se verifica si dicha red se encuentra creada mediante el comando #*virsh net-list*. En caso negativo se procede a inicializar dicha red con la sentencia # *virsh net-start default* y se intenta realizar nuevamente el proceso de arranque del *root filesystem*.

[7] http://web.dit.upm.es/vnxwiki/index.php/Vnx-modify-rootfs

- ✓ En algunas ocasiones otros bridges creados interfieren, por tanto la solución es borrar los bridges que no se estén utilizando.
  # *brctl delbr "nombre del bridge"*

Para la instalación de la última versión de POX se requiere Python 2.7 y se trabajó con el repositorio git[8]. Se realizó un clon en la máquina virtual que hará las funciones de controlador a través del comando:

git clone http://github.com/noxrepo/pox

Una vez instado POX será necesario acceder a la carpeta principal del mismo, para poder correr ya sean los ejemplos que vienen por defecto o crear nuevos programas.

cd pox

Para la instalación y modificación del controlador dentro de la máquina virtual no se tuvo inconvenientes.

De igual forma para la instalación de NOX[10] se sigue exactamente el mismo procedimiento de POX, es decir se realiza un clon desde el repositorio *git*.

Para la instalación de Beacon es necesario instalar los paquetes JDK, JRE y de igual forma realizar un clon de los directorios *openflowj* y *beacon* del repositorio *git*. El procedimiento seguido se encuentra en la página oficial de Beacon[9], así como las respectivas guías de configuración y uso.

3.2.3 Wireshark

Para la realización de pruebas es indispensable la instalación del analizador de tráfico *Wireshark*. Por defecto *wireshark* no cuenta con soporte para el protocolo *OpenFlow* al ser éste tan nuevo. Se plantea la opción de instalar un disector que servirá para dicho fin, y con lo cual se podrá capturar en la fase de pruebas si la red transmite paquetes con el protocolo OpenFlow. Existen algunas formas de instalar el disector, la más sencilla se detalla a continuación:

apt-get update

apt-get install wireshark-dev wireshark mercurial git

[8] Sistema de código abierto de control de versiones.
[9] https://openflow.stanford.edu/display/Beacon/Quick+Start

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

 # cd of-dissector/src

 # apt-get install scons

 # export WIRESHARK=/usr/include/wireshark/

 # scons install

Se crea un objeto compartido llamado openflow.so, el cual debe ser copiado al directorio que contiene *Wireshark*, en nuestro caso:

 # cp openflow.so /usr/lib/wireshark/libwireshark1/plugins/openflow.so

Si aún no reconoce el disector, o si se tiene otros inconvenientes se puede seguir las instrucciones dadas en la página *networkstatic* [10], la cual contiene tutoriales paso a paso relacionados con SDN.

Como complemento también se instaló las herramientas *tcpdump* que permitirá analizar el tráfico que circula por la red. En la parte de pruebas será utilizado para visualizar los paquetes que circulan por una interfaz determinada de un host. Además se instala la herramienta *iperf* que es una herramienta desarrollada para evaluar el desempeño en una comunicación y que funciona en modo cliente servidor. Una de las pruebas más utilizadas es la medición del ancho de banda.

Una vez instalados correctamente todos los elementos y con los tres escenarios VNX funcionando se puede realizar las pruebas de validación de los mismos, como se verá a continuación.

[10] http://networkstatic.net/installing-wireshark-on-linux-for-openflow-packet-captures/

4 Pruebas

Para la constatación del funcionamiento de los escenarios planteados las pruebas se centran en la captura de tráfico *OpenFlow* con la herramienta *Wireshark*. Adicionalmente se puede utilizar *tcpdump* para observar el comportamiento del sistema bajo diferentes configuraciones ya sea del controlador o de los switches *OpenFLow*. También a manera de tutorial, se detallará las posibles combinaciones de configuraciones del Open vSwitch y de los ejemplos de códigos de la interfaz del controlador.

4.1 Configuración Open vSwitch

Al término de la instalación de Open vSwitch, de manera predeterminada, funcionará igual que un switch normal o que la utilidad *brigde*, es decir solo permitirá la conexión de las diferentes máquinas virtuales generadas.

Es necesario la familiarización con algunos comandos básicos para comprobar el funcionamiento y las configuraciones de OVS, los cuales se resumen en la tabla 2.

Tabla 2 Comandos Básicos OVS

Comando	Función
# *ovs-vsctl show*	Ver la configuración del OVS.
# *ovs-ofctl show dump-dps*	Imprime los nombres de cada *datapath* configurado.
# *ovs-ofctl show Net0*	Visualizar la tabla de un *datapath* específico.
# *ovs-ofctl -s show Net0*	Ver un resumen de un *datapath* específico, incluyendo algunas estadísticas.
# *ovs-ofctl show dump-flows Net0*	Imprime las entradas de flujo de la tabla del switch. Las entradas que han hecho *match*.
# *ovs-vsctl emer-reset*	Realiza un *restart* del OVS, volviendo al mismo al estado inicial (sin ninguna configuración).
# *ovs-vsctl list-ifaces Net0*	Listar las interfaces asociadas a un bridge específico.
ovs-vsctl set-controller Parámetros	Asignar el controlador al switch respectivo, según la configuración de sus parámetros.

En este caso se ha tomado de referencia las capturas realizadas sobre el primer escenario, en el cual se define el bridge "*Net0*".

- ✓ Para visualización de la configuración de OVS se aplica el comando # *ovs-vsctl show* en la máquina host (ordenador físico), como se puede ver en la figura 23.

```
root@SATELLITE:/usr/share/tfm# ovs-vsctl show
8d6fae9f-9314-450a-99ae-fd8382233cc6
    Bridge "Net0"
        Port "h4-e1"
            Interface "h4-e1"
        Port "Net0"
            Interface "Net0"
                type: internal
        Port "h2-e1"
            Interface "h2-e1"
        Port "C1-e1"
            Interface "C1-e1"
        Port "Net0-e00"
            Interface "Net0-e00"
        Port "h3-e1"
            Interface "h3-e1"
    ovs_version: "1.4.0+build0"
```

Figura 23. Comando Visualización OVS

La configuración en el escenario 1, indica la creación de un bridge denominado "*Net0*", que a su vez contiene los puertos a los cuales se conectan los tres ordenadores ("h4-e1","h3-e1","h2-e1",) y el controlador del escenario (port "C1-e1").

- ✓ Para la visualización de los *datapaths* configurados:

```
root@SATELLITE:/usr/share/tfm# ovs-dpctl dump-dps
system@Net0
root@SATELLITE:/usr/share/tfm#
```

Figura 24. Datapaths del OVS

Como se tiene un solo *bridge*, se tendrá un solo *datapath*.

- ✓ Para la visualización de la tabla de flujos de un bridge específico se utiliza el comando:

ovs-ofctl show Net0

Donde *Net0* corresponde al nombre del bridge creado.

✓ Para visualizar un resumen completo con algunas estadísticas de un *datapath* específico (Net0 en el ejemplo) se utiliza el comando de la figura 25.

```
root@SATELLITE:/usr/share/tfm# ovs-dpctl -s show Net0
system@Net0:
        lookups: hit:208 missed:123 lost:0
        flows: 0
        port 0: Net0 (internal)
                RX packets:126 errors:0 dropped:0 overruns:0 frame:0
                TX packets:205 errors:0 dropped:0 aborted:0 carrier:0
                collisions:0
                RX bytes:27764 (27.1 KiB)  TX bytes:36058 (35.2 KiB)
        port 1: Net0-e00
                RX packets:0 errors:0 dropped:0 overruns:0 frame:0
                TX packets:282 errors:0 dropped:0 aborted:0 carrier:0
                collisions:0
                RX bytes:0  TX bytes:59188 (57.8 KiB)
        port 2: C1-e1
                RX packets:123 errors:0 dropped:0 overruns:0 frame:0
                TX packets:126 errors:0 dropped:0 aborted:0 carrier:0
                collisions:0
                RX bytes:23730 (23.2 KiB)  TX bytes:24452 (23.9 KiB)
        port 3: h2-e1
```

Figura 25. Estadísticas de un Datapath

Este comando es de vital importancia para la programación de flujos por parte del controlador, ya que permite ver la relación entre los nombres de los puertos con el número que le ha sido asignado; siendo este último el utilizado por el controlador. Por ejemplo al puerto denominado "*C1-e1*" se le ha asignado el valor numérico de 2.

Una vez familiarizados con los comandos de verificación del OVS, se procede a la asignación del controlador para lo cual se utiliza el siguiente comando:

ovs-vsctl set-controller name_bridge tcp:IP:port

En donde:

name_bridge representa el nombre del bridge configurado.

tcp:IP:port específica el puerto *tcp* de la IP dada, hacia el cual se conectará el bridge. El puerto de conexión por defecto es el 6633.

Por ejemplo en el escenario 1, la configuración será:

 # ovs-vsctl set-controller Net0 tcp:10.0.0.5:6633

Lo que indica es que para el bridge Net0 se ha configurado la conexión a un controlador que tiene la dirección IP 10.0.0.5 y que se conecta a través de una conexión TCP en el puerto 6633. Se podrá visualizar una pantalla similar a la figura 26.

```
Bridge "Net0"
    Controller "tcp:10.0.0.5:6633"
        is_connected: true
```

Figura 26. Controlador asignado al OVS

Si el controlador se encuentra operativo aparecerá "*is connected: true*"; caso contrario solamente aparecerá los parámetros fijos del *bridge*.

Existen dos modos en los que se puede configurar el controlador, estos son *standalone* y *secure*. La configuración se realiza a través de los siguientes dos comandos:

 # ovs-vsctl set-fail-mode Net0 standalone

 # ovs-vsctl set-fail-mode Net0 secure

En la figura 27 se puede visualizar los dos tipos de configuración.

```
Bridge "Net0"
    Controller "tcp:10.0.0.5:6633"
        is_connected: true
    fail_mode: standalone
```

a) Modo Standalone

```
Bridge "Net0"
    Controller "tcp:10.0.0.5:6633"
        is_connected: true
    fail_mode: secure
```

b) Modo Secure

Figura 27. Modos de Configuración Controlador OVS

De manera normal cuando un controlador toma el control de un *bridge*, es éste el que determina el tratamiento del tráfico proveniente del switch. Si la conexión falla y permanece así durante un tiempo los paquetes no podrán llegar a su destino. Para este caso es importante la configuración de uno de los dos modos mencionados anteriormente, dependiendo de qué tipo de funcionamiento se desee del OVS.

- ✓ *Standalone Mode*.- en este modo después de un intervalo de sondeo de inactividad del controlador (*inactivity probe interval*), será el OVS el que tome el control del switch y por tanto el que se encargue de reenviar los paquetes a su destino, basándose en el aprendizaje tradicional de un switch (*MAC-learning switch*). El intervalo de inactividad es determinado por el censado en tres tiempos por parte del switch hacia el controlador. Luego en background continuará intentando conectarse al controlador, en caso de conseguirlo delegará nuevamente al

controlador la responsabilidad sobre los flujos. *Standalone* es el modo por defecto con el que funciona OVS, aunque no se visualice de forma explícita. El problema del modo *standalone* es que crea lazos de reenvío en algunas topologías, por tanto la solución a este problema es activar el modo seguro o activar *STP*.

- ✓ *Secure Mode*.- con esta configuración se asegura que después de superado el tiempo de sondeo de inactividad, el OVS no pueda tomar el control de la tabla de flujos, por tanto la comunicación en la red se verá interrumpida hasta que el controlador se encuentre operativo nuevamente. Por su parte el OVS seguirá intentando conectarse al controlador.

4.2 Configuración del controlador POX

Como se ha mencionado anteriormente, el *root filesystem* para el controlador cuenta con los controladores POX, NOX y Beacon. Se hará mención solamente de POX.

Para poder probar la programación del flujos desde el controlador hacia los switches, se puede utilizar algunos componentes que vienen integrados con POX. Para lo cual se accede al directorio */home/vnx/pox*, a través de una consola en la respectiva máquina virtual. Los componentes más utilizados se detallan en la tabla 3.

Tabla 3. Componentes Controlador POX

Componente	Descripción
py	Permite inicializar un intérprete interactivo de Python, utilizado para debug y pruebas.
misc.of_tutorial	Permite que el switch actúe como un simple hub. Creado especialmente para el *OpenFlow Tutorial*.
forwarding.hub	Permite que el switch actúe como un hub.
forwarding.l2_learning	Actúa como un switch capa 2, es decir instala reglas de flujo en base a las direcciones MAC.
forwarding.3_learning	Instala reglas de flujo en base a las direcciones IP.
openflow.spanning_tree	Construye una topología de la red para evitar lazos , similar a STP.
web.webcore	Inicializa un servidor Web.
messenger	Provee una interfaz para interactuar con procesos externos a través de mensajes bidireccionales JSON.
openflow.discovery	Componente utilizado para el aprendizaje de la

	topología completa de la red, a través del envío de mensajes especiales LLDP.
openflow.keepalive	Permite el envío de mensajes *echo request* para conexión con el switch. Su principal objetivo es indicar al switch que el controlador se encuentra activo.
misc.packet_dump	Permite el volcado de los paquetes en un log.
log.level	Permite configurar que log se desea ver y en qué nivel de detalle. Los niveles son: critical, error, warning, info y debug.

Para probar algunos de los componentes que permiten el funcionamiento del switch, se puede hacer un inicio rápido a través del comando:

#*./pox.py component_name*

Donde *component_name* corresponde al nombre del componente que se desea ejecutar. Por ejemplo existe el componente creado especialmente para el *OpenFlow Tutorial*, el cual permite que el switch funcione como un hub, como se observa en la figura 28. En este caso se permite la visualización del log con nivel *Debug*.

```
root@C1:/home/vnx/pox# ./pox.py log.level --DEBUG misc.of_tutorial
POX 0.1.0 (betta) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.1.0 (betta) going up...
DEBUG:core:Running on CPython (2.7.3/Aug 1 2012 05:16:07)
DEBUG:core:Platform is Linux-3.2.0-38-generic-pae-i686-with-Ubuntu-12.04-precise
INFO:core:POX 0.1.0 (betta) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[5e-38-fb-61-af-4c 1] connected
DEBUG:misc.of_tutorial:Controlling [5e-38-fb-61-af-4c 1]
```

Figura 28. Inicio de Componente misc.of_tutorial

Cabe recalcar que los componentes de POX puede ser invocado con pox.py o debug-pox.py. También es posible usar el controlador tanto con el intérprete estándar de *Python* o con el compilador *PyPy*. *PyPy* es una alternativa rápida para compilar código realizado en *Python*.

Para entender de mejor forma el funcionamiento del protocolo OpenFlow y para que los resultados de las pruebas sean más legibles se considera conveniente describir los mensajes que se intercambian en un escenario que trabaja con el protocolo OpenFlow. La tabla 4 resume los principales mensajes intercambiados en el proceso.

Tabla 4. mensajes de Conexión OpenFlow

Mensajes		Descripción
Controller - to - Switch	*Features request* *Features reply*	Son usados para la verificación de las capacidades del switch.
	Configuration	El controlador configura algunos parámetros en el switch.
	Modify state	Sirve para administrar el estado del switch. El principal objetivo es añadir, borrar y modificar las entradas en las tablas.
	Read state	Utilizado para que el controlador pueda obtener información del switch, por ejemplo su configuración actual, capacidades y estadísticas.
	Packet out	El controlador usa este mensaje para el envío de paquetes a través de un puerto específico del switch. Reenvía los paquetes *Packet-in* que recibe. También contiene una lista de acciones para ser aplicadas en un orden determinado.
	Barrier Request *Barrier Reply*	Usado para recibir notificaciones de operaciones realizadas.
	Role-request	Este mensaje permite indicar el rol. Es muy utilizado para la conexión de un switch a múltiples controladores.
Asynchronous	*Packet in*	Permite la transferencia del control de un paquete al controlador.
	Flow Removed	Notifica al controlador de la remoción de una entrada de flujo en la tabla.
	Port status	Informa al controlador de un cambio en un puerto.
	Error	Notificación de problemas.
Symmetric	*Hello*	Es un mensaje que se intercambia entre el controlador y el switch al iniciar la conexión.
	echo request *echo reply*	Son usados tanto por el switch como por el controlador para verificar el estado de la conexión. Por ejemplo la medida de la latencia.
	Experimenter	Utilizado para proveer funcionalidades adicionales a OpenFlow. Pensado para futuras versiones del protocolo.

Por ejemplo al iniciar la conexión en uno de los escenarios de pruebas, se realiza el intercambio de los mensajes *echo request y echo reply*, en la figura 29 se puede visualizar la versión del protocolo en este caso la versión 1.0 así como el tipo de mensaje (2).

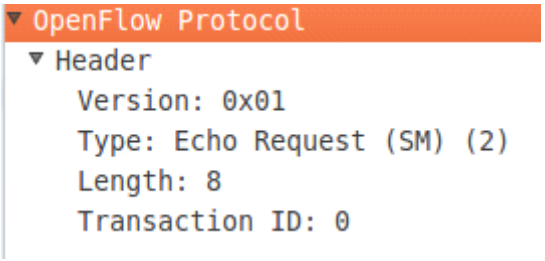

Figura 29. Mensaje Echo Request

Ahora, en base a los diferentes comandos y configuraciones tanto de switch como del controlador se procede a realizar algunas pruebas en los tres escenarios, reiterando una vez más que cada una de las configuraciones es posible para cualquiera de ellos.

4.3 Escenario 1: Esquema básico OpenFlow

En el escenario número 1, en la máquina virtual controlador (10..0.0.5), se ejecuta el componente *misc.of_tutorial*, que permite programar al OVS como un hub simple. El OVS es configurado en primera instancia en modo *standalone* y luego en modo *secure*.

Para ambos casos la captura del wireshark es la mostrada en la figura 30. Cabe recalcar que wireshark se encuentra instalado en la máquina virtual que hace las veces de controlador.

1	0.000000	10.0.0.1	10.0.0.5	OFP	74	Echo Request (SM) (8B)
2	0.022342	10.0.0.5	10.0.0.1	OFP	74	Echo Reply (SM) (8B)
4	5.001090	10.0.0.1	10.0.0.5	OFP	74	Echo Request (SM) (8B)
6	5.046200	10.0.0.5	10.0.0.1	OFP	74	Echo Reply (SM) (8B)

Figura 30. Captura inicialización Escenario 1

Como se puede ver en la figura 30, la red funciona con el protocolo OpenFlow, en este caso se genera dos tipos de mensajes Echo Request y Echo Reply, los mismos que corresponden a la conexión satisfactoria entre controlador (10.0.0.5) y el OVS (10.0.0.1). Todos los mensajes que atraviesen la red tendrán como direcciones de origen o destino las del controlador o la del OVS.

Ahora se realiza un ping desde el la máquina virtual h2 (10.0.0.2) a la máquina host (10.0.0.1), como se puede visualizar en el paquete entrante en la figura 31.

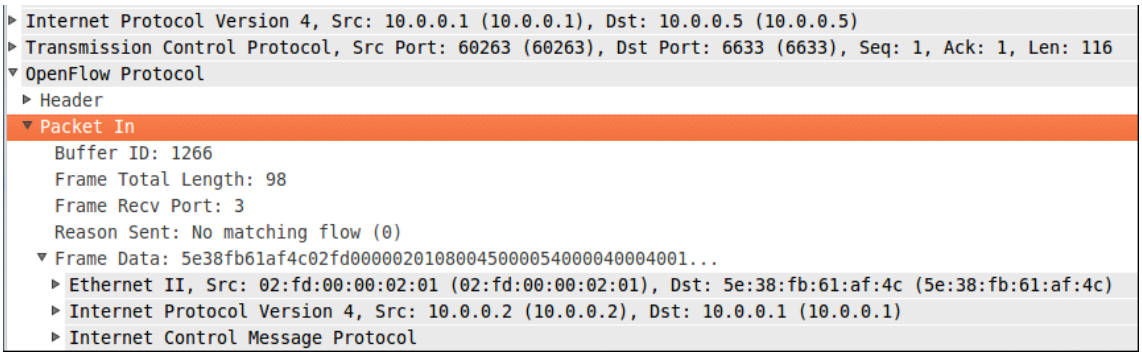

Figura 31. Captura Wireshark Packet In

Se observa el paquete de entrada "*In*", desde la dirección IP de origen 10.0.0.2 hacia la de destino 10.0.0.1. Como respuesta a este requerimiento se genera un paquete "*Out*", el cual indica el tratamiento del paquete como se observa en la figura 32.

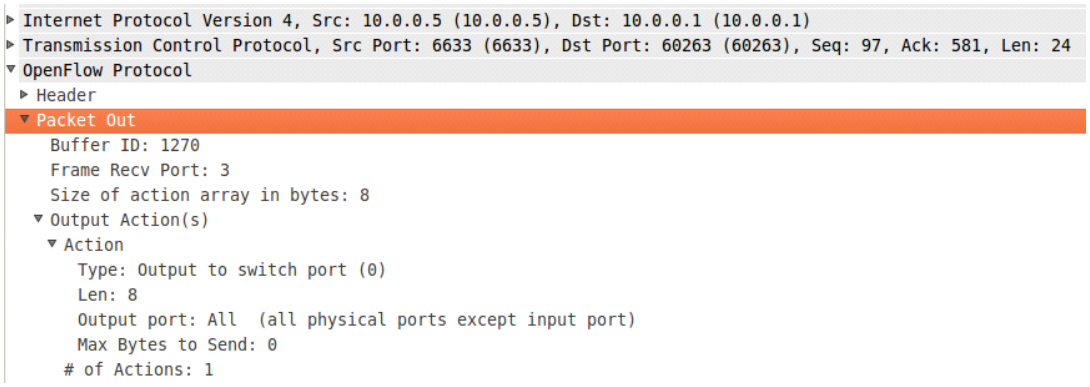

Figura 32. Captura Wireshark Packet Out

En este caso como se está ejecutando el componente que permite que el switch funcione como un *hub*, el controlador indica que se genere un mensaje de salida (*Packet Out*) y que la acción que debe ejecutar es el envío de dicho paquete por todos los puertos *(Output port: all*).

Si ahora se corre el componente *fordwarding.l2_learning*, se tendrá una captura similar a la de la figura 33.

5	5.005458	10.0.0.5	10.0.0.1	OFP	74	Echo Reply (SM) (8B)
7	9.267845	02:fd:00:00:03:01	02:fd:00:00:02:01	OFP+ARP	126	Packet In (AM) (BufID=4595) (60B) => Who
8	9.277876	10.0.0.5	10.0.0.1	OFP	146	Flow Mod (CSM) (80B)
10	9.278767	02:fd:00:00:02:01	02:fd:00:00:03:01	OFP+ARP	126	Packet In (AM) (BufID=4596) (60B) => 10.
11	9.280138	10.0.0.5	10.0.0.1	OFP	146	Flow Mod (CSM) (80B)
13	13.129731	10.0.0.2	10.0.0.4	OFP+ICMP	182	Packet In (AM) (BufID=4597) (116B) => Ec
14	13.158148	10.0.0.5	10.0.0.1	OFP	146	Flow Mod (CSM) (80B)
16	13.277071	10.0.0.3	10.0.0.2	OFP+ICMP	182	Packet In (AM) (BufID=4598) (116B) => Ec
17	13.278466	10.0.0.5	10.0.0.1	OFP	146	Flow Mod (CSM) (80B)

Figura 33. Mensajes OFP+ARP y OFP+ICMP

En este caso como se utiliza el componente que permite emular el funcionamiento de un switch normal, se puede visualizar las direcciones IP de origen y destino desde donde se realiza el requerimiento, lo que no sucedía en el componente que emula un hub. Se puede observar mensajes de establecimiento de conexión (*Echo Reply / Request*) y de mensajes *Flow mod*. Al iniciar la comunicación el controlador realizará una consulta ARP para preguntar la dirección MAC de una IP específica, de igual forma que un switch l2 convencional. También aparecen los mensajes ICMP.

Del lado de la interfaz del controlador, al momento de correr el componente se verá algo similar a la figura 34.

```
root@C1:/home/vnx/pox# ./pox.py log.level --DEBUG forwarding.l2_learning
POX 0.1.0 (betta) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.1.0 (betta) going up...
DEBUG:core:Running on CPython (2.7.3/Aug 1 2012 05:16:07)
DEBUG:core:Platform is Linux-3.2.0-38-generic-pae-i686-with-Ubuntu-12.04-precise
INFO:core:POX 0.1.0 (betta) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[5e-38-fb-61-af-4c 1] connected
DEBUG:forwarding.l2_learning:Connection [5e-38-fb-61-af-4c 1]
DEBUG:forwarding.l2_learning:Port for 5e:38:fb:61:af:4c unknown -- flooding
DEBUG:forwarding.l2_learning:installing flow for 5e:38:fb:61:af:4c.65534 -> 02:f
d:00:00:02:01.3
DEBUG:forwarding.l2_learning:installing flow for 02:fd:00:00:04:01.5 -> 02:fd:00
:00:02:01.3
DEBUG:forwarding.l2_learning:installing flow for 02:fd:00:00:02:01.3 -> 02:fd:00
:00:04:01.5
```

Figura 34. Controlador funcionando con l2_learning

Esta captura indica que componente está utilizando el controlador y adicionalmente refleja las consultas realizadas para el aprendizaje del controlador.

Finalmente se prueba el componente forwarding.l3_learning, como se visualiza en la figura 35.

```
root@C1:/home/vnx/pox# ./pox.py log.level --DEBUG forwarding.l3_learning
POX 0.1.0 (betta) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.1.0 (betta) going up...
DEBUG:core:Running on CPython (2.7.3/Aug 1 2012 05:16:07)
DEBUG:core:Platform is Linux-3.2.0-38-generic-pae-i686-with-Ubuntu-12.04-precise
DEBUG:forwarding.l3_learning:Up...
INFO:core:POX 0.1.0 (betta) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[5e-38-fb-61-af-4c 1] connected
DEBUG:forwarding.l3_learning:103598828662604 3 ARP request 10.0.0.2 => 10.0.0.3
DEBUG:forwarding.l3_learning:103598828662604 3 learned 10.0.0.2
DEBUG:forwarding.l3_learning:103598828662604 3 flooding ARP request 10.0.0.2 =>
10.0.0.3
DEBUG:forwarding.l3_learning:103598828662604 4 ARP reply 10.0.0.3 => 10.0.0.2
DEBUG:forwarding.l3_learning:103598828662604 4 learned 10.0.0.3
DEBUG:forwarding.l3_learning:103598828662604 4 flooding ARP reply 10.0.0.3 => 10
.0.0.2
DEBUG:forwarding.l3_learning:103598828662604 3 IP 10.0.0.2 => 10.0.0.3
DEBUG:forwarding.l3_learning:103598828662604 3 installing flow for 10.0.0.2 => 1
0.0.0.3 out port 4
DEBUG:forwarding.l3_learning:103598828662604 4 IP 10.0.0.3 => 10.0.0.2
DEBUG:forwarding.l3_learning:103598828662604 4 installing flow for 10.0.0.3 => 1
```

Figura 35. Controlador funcionando con l3_learning

Se puede observar que la instalación de flujos se realiza en función de las direcciones IP.

4.4 Escenario 2: Escenario con dos controladores

El segundo escenario corresponde a una ampliación del escenario número 1. Lo que se intenta en éste es incluir redundancia en caso de falla, para lo cual se ha introducido al esquema un segundo controlador. En ambos controladores se utiliza el mismo *root filesystem*. La asignación de los dos controladores al bridge Net0 se realiza a través del siguiente comando:

ovs-vsctl set-controller Net0 tcp:10.0.0.5:6633 tcp:10.0.0.6:6633

ovs-vsctl set-fail-mode Net0 secure

Si se realiza un show del OVS, se verá la configuración tal como consta en la figura 36:

```
8d6fae9f-9314-450a-99ae-fd8382233cc6
    Bridge "Net0"
        Controller "tcp:10.0.0.5:6633"
            is_connected: true
        Controller "tcp:10.0.0.6:6633"
            is_connected: true
        fail_mode: secure
        Port "h4-e1"
```

Figura 36. Configuración OVS Escenario 2

En este caso ambos controladores se encuentran operativos y cada uno de ellos recibe los diferentes requerimientos por parte del bridge Net0. Uno de los grandes inconvenientes en este escenario fue que dependiendo del tipo de componente que se ponga en funcionamiento en el controlador, se produce paquetes duplicados en la red. Como se había explicado, *OpenFlow* solamente define los mensajes que deberán intercambiarse en este tipo de escenarios, mas no ofrece un mecanismo para este fin. La consistencia entre los controladores se encuentra en etapas de investigación y en los trabajos presentados hasta ahora no se entrega resultados que puedan ser considerados 100% exitosos. Los mecanismos de consistencia por el momento son realizados a nivel de la programación de los diferentes controladores.

Para verificar el tráfico *OpenFlow* en el escenario se realiza un ping desde el host h2 (10.0.0.2) al host h3 (10.0.0.3) y se toman las capturas de pantalla de cada uno de los controladores como se puede ver en las figuras 37 y 38.

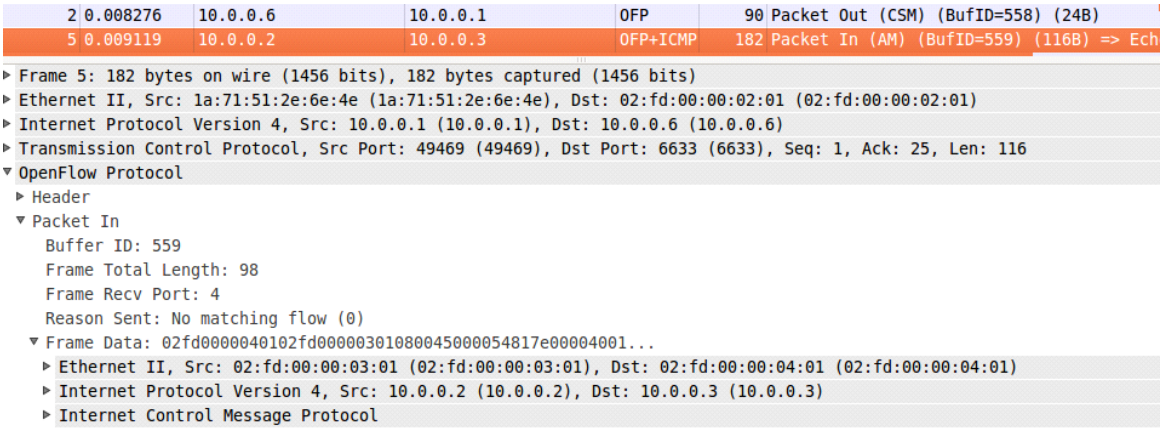

Figura 37. Wireshark Controlador C2

En la figura 37 se muestra el paquete de entrada generado por el ping desde h2 a h3. Así como la dirección IP del controlador que actúa, en este caso el 10.0.0.6 (C2). Por otra parte en la figura 38 consta el paquete de entrada ICMP pero en el primer controlador (10.0.0.5).

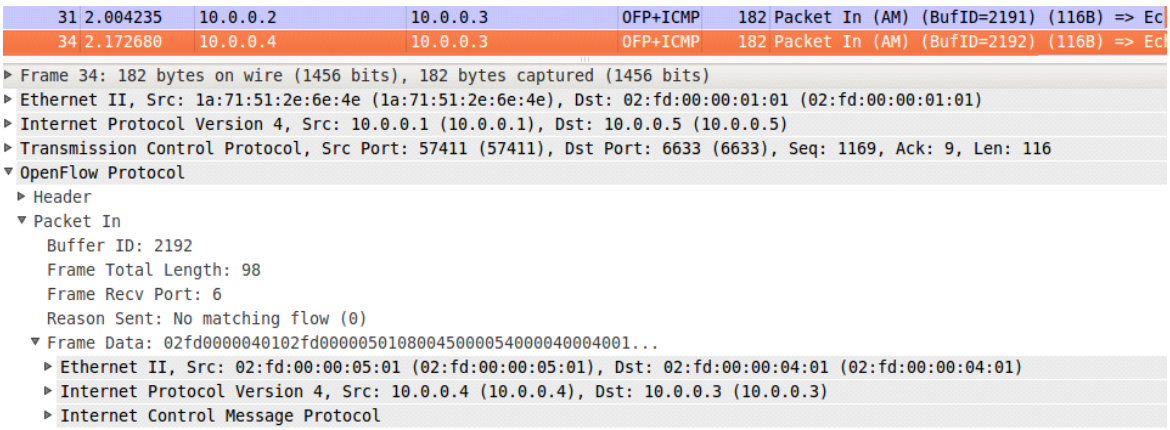

Figura 38. Wireshark Controlador C2

En este escenario se realizó una prueba con *tcpdump* en paralelo en los host h4 y h2, para lo cual se realiza un ping desde el host h3 al controlador c2 (10.0.0.3 a 10.0.0.6) y ejecutando el componente misc.of_tutorial a nivel de controlador.

 # tcpdump -XX -n -i h2-eth0

Cuando funciona como hub se produce un proceso de inundación, se envía todos los paquetes a cada puerto dentro de la red excepto por el puerto de entrada. Las figuras 39 y 40 muestran una salida exactamente igual del comando *tcpdump*. Esto se debe a que el switch al estar funcionando como un hub, hará una inundación en toda la red, en consecuencia cada host verá reflejado dicho tráfico.

```
root@h4:/home/vnx# tcpdump -XX -n -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
21:09:46.134843 IP 10.0.0.3 > 10.0.0.6: ICMP echo request, id 1590, seq 1, le
ngth 64
        0x0000:  02fd 0000 0201 02fd 0000 0401 0800 4500  ..............E.
        0x0010:  0054 0000 4000 4001 26a1 0a00 0003 0a00  .T..@.@.&.......
        0x0020:  0006 0800 bdaa 0636 0001 79f5 a851 19d4  .......6..y..Q..
        0x0030:  0d00 0809 0a0b 0c0d 0e0f 1011 1213 1415  ................
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
21:09:46.137560 IP 10.0.0.6 > 10.0.0.3: ICMP echo reply, id 1590, seq 1, leng
th 64
```

Figura 39. Tcpdump en host h4

De igual forma ocurre en el host h2, como se muestra en la figura 40.

```
root@h2:/home/vnx# tcpdump -XX -n -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
21:09:46.414244 IP 10.0.0.3 > 10.0.0.6: ICMP echo request, id 1590, seq 1, len
gth 64
        0x0000:  02fd 0000 0201 02fd 0000 0401 0800 4500  ..............E.
        0x0010:  0054 0000 4000 4001 26a1 0a00 0003 0a00  .T..@.@.&.......
        0x0020:  0006 0800 bdaa 0636 0001 79f5 a851 19d4  .......6..y..Q..
        0x0030:  0d00 0809 0a0b 0c0d 0e0f 1011 1213 1415  ................
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
21:09:46.417444 IP 10.0.0.6 > 10.0.0.3: ICMP echo reply, id 1590, seq 1, lengt
h 64
```

Figura 40. Tcpdump en host h2

4.5 Escenario 3: Escenario

Para el escenario tres es posible realizar todas las configuraciones y pruebas de los dos escenarios anteriores. Lo que se intenta demostrar en este escenario es que si se incrementa el tamaño de la red se puede de igual forma seguir trabajando con el protocolo *OpenFlow*. La configuración aplicada fue la siguiente:

 # ovs-vsctl set-controller Net0 tcp:10.0.0.2:6633 tcp:10.0.1.3:6633

 # ovs-vsctl set-controller Net1 tcp:10.0.0.2:6633 tcp:10.0.1.3:6633

 # ovs-vsctl set-controller Net2 tcp:10.0.0.2:6633 tcp:10.0.1.3:6633

 # ovs-vsctl set-fail-mode Net0 secure

 # ovs-vsctl set-fail-mode Net1 secure

 # ovs-vsctl set-fail-mode Net2 secure

Si se realiza la configuración bajo estos parámetros y cada uno de los controladores se encuentra operativo, al realizar un show del OVS se podrá ver una pantalla similar a la de la figura 41.

```
Bridge "Net0"
    Controller "tcp:10.0.0.2:6633"
        is_connected: true
    Controller "tcp:10.0.1.3:6633"
        is_connected: true
    fail_mode: secure

Bridge "Net1"
    Controller "tcp:10.0.0.2:6633"
        is_connected: true
    Controller "tcp:10.0.1.3:6633"
        is_connected: true
    fail_mode: secure

Bridge "Net2"
    Controller "tcp:10.0.1.3:6633"
        is_connected: true
    Controller "tcp:10.0.0.2:6633"
        is_connected: true
    fail_mode: secure
```

Figura 41. Configuración Escenario 3

El controlador C2 se encuentra trabajando con el componente forwarding.hub que permite la emulación de un hub. En la figura 42 se puede visualizar los tres bridges conectados al controlador, cada uno representado por su dirección MAC. de igual forma sucederá en el controlador C1.

```
root@C2:/home/vnx/pox# ./pox.py log.level -DEBUG forwarding.hub
POX 0.1.0 (betta) / Copyright 2011-2013 James McCauley, et al.
INFO:forwarding.hub:Hub running.
DEBUG:core:POX 0.1.0 (betta) going up...
DEBUG:core:Running on CPython (2.7.3/Aug 1 2012 05:16:07)
DEBUG:core:Platform is Linux-3.2.0-38-generic-pae-i686-with-Ubuntu-12.04-precise
INFO:core:POX 0.1.0 (betta) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[52-c6-6d-d2-c8-44 1] connected
INFO:forwarding.hub:Hubifying 52-c6-6d-d2-c8-44
INFO:openflow.of_01:[0e-fe-d9-7b-f9-4b 3] connected
INFO:forwarding.hub:Hubifying 0e-fe-d9-7b-f9-4b
INFO:openflow.of_01:[62-18-8d-56-df-42 2] connected
INFO:forwarding.hub:Hubifying 62-18-8d-56-df-42
```

Figura 42. Funcionamiento Controlador Escenario 3

Para la comprobación del funcionamiento del escenario se tomó una captura del tráfico con la herramienta Wireshark en el controlador C1 (10.0.0.2), como puede verse en la figura 43.

26	4.088734	10.0.0.2	10.0.2.2	OFP	74	Echo Reply (SM) (8B)
27	4.088827	10.0.0.2	10.0.2.2	OFP	74	Echo Reply (SM) (8B)
28	4.101774	02:fd:00:00:03:01	02:fd:00:00:01:01	OFP+ARP	126	Packet In (AM) (BufID=268) (60B) => Who
29	4.105569	10.0.0.2	10.0.2.2	OFP	90	Packet Out (CSM) (BufID=268) (24B)
32	4.106380	02:fd:00:00:01:01	02:fd:00:00:03:01	OFP+ARP	126	Packet In (AM) (BufID=269) (60B) => 10.0
33	4.108450	10.0.0.2	10.0.2.2	OFP	90	Packet Out (CSM) (BufID=269) (24B)
37	4.425297	10.0.0.3	10.0.1.2	OFP+ICMP	182	Packet In (AM) (BufID=270) (116B) => Ech
39	5.038602	10.0.1.2	10.0.0.3	OFP+ICMP	182	Packet In (AM) (BufID=274) (116B) => Ech
41	5.425545	10.0.0.3	10.0.1.2	OFP+ICMP	182	Packet In (AM) (BufID=271) (116B) => Ech

Figura 43. Wireshark Escenario 3

De manera semejante se verá si se realiza una captura en el controlador C2. Mediante las capturas con wireshark queda totalmente comprobado el correcto funcionamiento de un escenario virtual generado con VNX con soporte para el protocolo OpenFlow.

Si por algún motivo el OVS pierde su configuración, o si se ha aplicado el comando *ovs-vsctl emer-reset*, se producirá el cierre de la conexión con el controlador, tal como se ve en la figura 44.

```
INFO:openflow.of_01:[62-18-8d-56-df-42 3] closed
INFO:openflow.of_01:[0e-fe-d9-7b-f9-4b 1] closed
INFO:openflow.of_01:[52-c6-6d-d2-c8-44 2] closed
```

Figura 44. Controlador Cerrado

Ahora si el componente que se encuentra en funcionamiento en el controlador es detenido, la conexión pasará a un estado "*disconnected*" de desconexión, como se puede apreciar en la figura 45.

```
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[5e-38-fb-61-af-4c 1] connected
DEBUG:misc.of_tutorial:Controlling [5e-38-fb-61-af-4c 1]
^CINFO:core:Going down...
INFO:openflow.of_01:[5e-38-fb-61-af-4c 1] disconnected
INFO:core:Down.
```

Figura 45. Controlador Desconectado

Al momento de ir probando las diferentes configuraciones y los componentes de POX, se pudo ver los siguientes funcionamientos particulares.

- ✓ Cuando se trabaja en modo *standalone*, una vez que un controlador es cerrado o desconectado el switch entra en funcionamiento normal de forma automática, sin pérdida de conectividad.
- ✓ En caso de pérdida de conexión con un controlador que esté funcionando en modo hub y si el OVS se encuentra configurado en modo seguro, automáticamente la comunicación se pierde.
- ✓ Si se realiza la configuración en modo seguro y el controlador está funcionando con alguno de los componentes que emulan un switch l2 o l3, y si la conexión al controlador falla la comunicación no se pierde automáticamente. Se tendrá un tiempo de respuesta adicional de conexión.

5 Conclusiones y trabajos futuros

5.1 Conclusiones

El presente trabajo propone un escenario para pruebas del protocolo OpenFlow, para lo cual en base al estado del arte se proporciona los principales conceptos relacionados con *Software Defined Networking* y OpenFlow. Se determina en base a la arquitectura de SDN la necesidad de switches y software controlador que tengan soporte para OpenFlow. Es así que se escoge la herramienta *Open vSwitch*, la cual es integrada con VNX, reemplazando la utilidad de bridge tradicional que viene incluida en los sistemas operativos Linux.

En el caso del software controlador se decide añadir a la imagen *del root filesystem* Ubuntu 12.04 tres controladores diferentes, NOX, Beacon y POX; siendo este último el utilizado en las pruebas de los diferentes escenarios.

Una vez listos los componentes de la arquitectura OpenFlow se diseñan e implementan tres escenarios para experimentar con SDN, a partir de una topología básica idéntica a *OpenFlow Tutorial*. Se procede a realizar diferentes configuraciones del switch, con sus dos modos de funcionamiento: *standalone y secure*. Por otro lado se utiliza algunos componentes que vienen incluidos con el controlador POX, por ejemplo para que el *brigde* de los escenarios definidos funcionen en modo *hub*.

Se presentan las pruebas realizadas a través de la captura de tráfico con la herramienta *wireshark* y adicionalmente con *tcpdump*. Al ser OpenFlow un protocolo tan reciente Wireshark no lo tiene identificado en consecuencia fue necesario la instalación de un disector para *Openflow*.

Se introduce cierto nivel de seguridad a la red en lo que respecta a disponibilidad a través de la inclusión de un segundo controlador. Se generan problemas de duplicidad de la información, debido a la falta de mecanismos que ayuden a conservar la consistencia de la información entre los controladores.

SDN al ser una tecnología totalmente nueva todavía no tiene definido que funciones le corresponden al dispositivo de red y cuales residen en el controlador, las investigaciones se encuentran en etapas iniciales por tanto existe un gran campo de estudio hasta tener una versión totalmente estable. Sin lugar a dudas SDN y OpenFlow darán mucho de qué hablar en un futuro.

5.2 Trabajo futuros

Los escenarios presentados han ayudado a probar el funcionamiento del protocolo *OpenFlow*. Gracias a las funcionales de VNX se podrá desplegar grandes redes en función del *root filesystem* generado y la correcta instalación de Open vSwitch. Sin embargo se podría mejorar los escenarios implementando otros elementos del stack SDN, tal es el caso de la herramienta *FlowVisor* o alguna de los proyectos para monitoreo y *debug*.

Dentro de las líneas de investigación del proyecto VNX y sus posibles avances se encuentra el mejorar la emulación de redes, para lo cual se había planteado la inclusión de Open vSwitch como base para la creación de sus escenarios. En el presente trabajo se ha fusionado el uso de VNX con Open vSwitch de manera exitosa. No obstante, el trabajo solamente se ha enfocado en las funcionalidades para *OpenFlow*, en consecuencia se deja la posibilidad de experimentar con las demás bondades que proporciona Open vSwitch.

El uso del controlador POX se ha basado en la ejecución de algunos de los componentes incluidos en la herramienta. El estudiante podrá hacer uso de ella y crear sus propias líneas de código. También se ha instalado en el *root filesystem* base dos controladores adicionales NOX y Beacon, con lo cual se brinda dos lenguajes de programación diferentes.

Uno de los más grandes retos de este proyecto fue la tolerancia a fallos de los controladores *OpenFlow*. El mayor problema surge en la consistencia entre los controladores. Dentro de las pruebas realizadas con dos controladores simultáneos, se ha tenido el inconveniente de mensajes duplicados, lo cual crea cierta ineficiencia en el sistema. El protocolo OpenFlow no proporciona ningún mecanismos de control entre varios controladores en caso de fallas, por tanto queda abierta la posibilidad de crear mecanismos o código que permita mantener la consistencia de la red, para que en caso de falla del controlador principal, el secundario entre automáticamente en funcionamiento sin perder el estado de la red.

6 Bibliografía

[1] B. Heller. *OpenFlowTutorial_ONS_Heller.pdf* . Available: http://www.stanford.edu/~brandonh/ONS/OpenFlowTutorial_ONS_Heller.pdf.

[2] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown and S. Shenker, "Ethane: Taking control of the enterprise," in *ACM SIGCOMM Computer Communication Review,* 2007, pp. 1-12.

[3] N. M. K. Chowdhury and R. Boutaba, "Network virtualization: state of the art and research challenges," *Communications Magazine, IEEE,* vol. 47, pp. 20-26, 2009.

[4] J. Sahoo, S. Mohapatra and R. Lath, "Virtualization: A survey on concepts, taxonomy and associated security issues," in *Computer and Network Technology (ICCNT), 2010 Second International Conference on,* 2010, pp. 222-226.

[5] F. Galán, D. Fernández, W. Fuertes, M. Gómez and de Vergara, Jorge E López, "Scenario-based virtual network infrastructure management in research and educational testbeds with VNUML," *Annals of Telecommunications-Annales Des Télécommunications,* vol. 64, pp. 305-323, 2009.

[6] *VNX* . Available: http://web.dit.upm.es/vnxwiki/index.php/Main_Page.

[7] *Libvirt. The virtualization API* . Available: http://libvirt.org/.

[8] D. Fernández, A. Cordero, J. Somavilla, J. Rodriguez, A. Corchero, L. Tarrafeta and F. Galán, "Distributed virtual scenarios over multi-host linux environments," in *Systems and Virtualization Management (SVM), 2011 5th International DMTF Academic Alliance Workshop on,* 2011, pp. 1-8.

[9] Open Networking Foundation. *Software-Defined Networking: The New Norm for Networks*. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[10] NOXRepo. *About NOX*. Available: http://www.noxrepo.org/nox/about-nox/.

[11] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. McKeown and G. Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium, Tech.Rep,* 2009.

[12] B. Lantz, B. Heller and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks,* 2010, pp. 19.

[13] SDNCentral. *Software Defined Networking in 2018*. Available: http://www.sdncentral.com/sdn-market-sizing-2013-april/.

[14] W. Stallings. "Software Defined Networks and OpenFlow". Internet Protocol Journal. *16(1),* pp. 02-14. 2013.

[15] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review,* vol. 38, pp. 69-74, 2008.

[16] P. Fonseca, R. Bennesby, E. Mota and A. Passito, "A replication component for resilient OpenFlow-based networking," in Network Operations and Management Symposium (NOMS), 2012 IEEE, 2012, pp. 933-939.

[17] McKeown Group Wiki, "Mininet " *Foswiki's OpenFlow Web,* 12-05, 2012.

[18] OpenFlow Wiki. *OpenFlow Tutorial*. Available: http://www.openflow.org/wk/index.php/OpenFlow_Tutorial.

[19] J. Pettit, J. Gross, B. Pfaff, M. Casado and S. Crosby, "Virtual switching in an era of advanced edges," in *2nd Workshop on Data Center–Converged and Virtual Ethernet Switching (DC-CAVES),* 2010, .

[20] Internet-Draft. *The Open vSwitch Database Management Protocol Draft*. Available: http://tools.ietf.org/pdf/draft-pfaff-ovsdb-proto-02.pdf.

[21] Open vSwitch. *Why Open vSwitch*. Available: http://openvswitch.org/cgi-bin/gitweb.cgi?p=openvswitch;a=blob_plain;f=WHY-OVS;hb=HEAD.

[22] *Features | Open vSwitch* . Available: http://openvswitch.org/features/.

[23] A. Al-Shabibi. *POX Wiki*. https://openflow.stanford.edu/display/ONL/POX+Wiki

[24] N. McKeown, "Software-Defined Networking," *INFOCOM Keynote Talk*, Apr, 2009.

[25] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue and T. Hama, "Onix: A distributed control platform for large-scale production networks," *in Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, 2010, pp. 1-6.

[26] OpenFlow. *Enabling Innovation in Your Network*. Available: http://www.openflow.org/.