ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Programación de redes SDN mediante el controlador POX

AUTOR: Pablo Yagües Fernández

DIRECTOR: José María Malgosa Sanahuja

Octubre / 2015

ÍNDICE DE LA MEMORIA

Agradecimientos

Me gustaría empezar este TFG agradeciendo el apoyo incondicional de mi familia y de todas las personas de mi entorno, las cuales han depositado su confianza en mí. Haciendo todo lo que estaba en su mano para que el mismo pudiera ser realizado.

También deseo agradecer al personal e infraestructura de la Universidad Politécnica de Cartagena que con sus números libros, equipos y distintas herramientas de aprendizaje facilitan a todos y cada uno de sus alumnos que puedan aprender y trabajar sin ningún tipo de limitación menos el que ellos mismos se impongan.

1 Introducción

Probablemente el lector de este texto sabe qué es un protocolo de red. Para los que no, en informática y telecomunicación, un protocolo de comunicaciones es un sistema de reglas que define la sintaxis, semántica y sincronización de la comunicación que permiten que dos o más entidades de un sistema de comunicación se comuniquen entre ellas para transmitir información.

En el presente nos encontramos en un estancamiento en la evolución de protocolos de red, el tema el cual se habla en este proyecto puede cambiar esta realidad cambiando la forma en la que funcionan las redes.

En este proyecto hay grandes empresas implicadas tales como *Google*, *Microsoft*, *Facebook* a los que se han ido añadiendo: Allied Telesis, Citrix, Cisco, Dell, HP, F5 Networks, IBM, NEC, Huawei, Juniper Networks, Oracle y VMware, todas ellas grandes empresas del sector de las telecomunicaciones.

En el presente la configuración de los componentes de la red se realiza mediante el software integrado en los mismos componentes, la idea principal de OpenFlow es que la programación de estos se haga desde un nodo central creando de esta forma redes definidas por software (Software-Defined Network, SDN)

Urs Hölzle (vicepresidente de Ingeniería de *Google)* afirma:

“OpenFlow puede provocar un giro de ciento ochenta grados. Una de las razones es que en las redes actuales se emplean una gran cantidad de protocolos y que no siempre interaccionan bien entre ellos. OpenFlow rompe ese modelo, toda la inteligencia estará en un punto central”

Nos planteamos la siguiente cuestión ¿Estamos ante uno de los cambios más importantes en relación a los protocolos de red?

1.1 Motivación y Objetivos

Este proyecto tiene por objetivo intentar comprender el concepto de **SDN** (Software Defined Network), junto con el protocolo con más relevancia en este campo, **OpenFlow**.
Antes de comenzar con el marco teórico expondremos una serie de objetivos que intentaremos abordar en este proyecto final de grado:

1. Conocer las características técnicas e históricas de la virtualización de redes y las causas de la aparición de Openflow.
2. Obtener una visión general de la tecnología Openflow.
3. Intentaremos comprender el uso de la herramienta MiniNet, un emulador que permite crear redes virtuales escalables, esta herramienta está enfocada principalmente a redes SDN.
4. Familiarizarnos con la herramienta POX, con el fin de implementar funcionalidades que puedan ser aplicables a un switch con tecnología OpenFlow.
5. Implantar un escenario OpenFlow virtualizado donde cargaremos distintos tipos de controladores ya presentes a los cuales añadiremos nuevas funcionalidades.
6. Conocer las diferentes aplicaciones de OpenFlow así como sus ventajas que aporta en este nuevo marco de las redes SDN

1.2 Virtualizacion de Redes

La virtualización de redes es la combinación de los recursos de red del hardware con los recursos de red del software en una única unidad administrativa. El objetivo de la virtualización de redes consiste en facilitar un uso compartido de recursos de redes eficaz, controlado y seguro para los usuarios y los sistemas.

La virtualización de red está destinada a mejorar la productividad y la eficiencia mediante la realización de tareas de forma automática, permitiendo que los archivos, imágenes y programas que se gestionen de forma centralizada desde un único sitio físico.

Los objetivos de diseño para el entorno de virtualización de red, son los siguientes:

- Flexibilidad y heterogeneidad
- Capacidad de gestión
- Aislamiento
- Programabilidad
- Facilidad en el desarrollo y en la investigación
- Soporte a sistemas tradicionales

Tecnologías

1. Red de área local virtual (VLAN)

Una VLAN (Red de área local virtual o LAN virtual) es un método para crear redes lógicas independientes dentro de una misma red física.
Gracias a las redes virtuales (VLAN), es posible liberarse de las limitaciones de la arquitectura física (geográficas, de direccionamiento, etc.), ya que se define una segmentación lógica basada en el agrupamiento de equipos según determinados criterios (direcciones MAC, número de puertos, protocolo, etc).

Tipos de VLAN

VLAN de nivel 1 (por puerto). También conocida como “port switching”. Se especifica qué puertos del switch pertenecen a la VLAN, los miembros de dicha VLAN son los que se conecten a esos puertos.

VLAN de nivel 2 por direcciones MAC. Se asignan hosts a una VLAN en función de su dirección MAC. Tiene la ventaja de que no hay que reconfigurar el dispositivo de conmutación si el usuario cambia su localización.

VLAN de nivel 2 por tipo de protocolo. La VLAN queda determinada por el contenido del campo tipo de protocolo de la trama MAC

VLAN de nivel 3 por direcciones de subred (subred virtual). La cabecera de nivel 3 se utiliza para mapear la VLAN a la que pertenece. En este tipo de VLAN son los paquetes, y no las estaciones, quienes pertenecen a la VLAN.

VLAN de niveles superiores. La pertenencia a una VLAN puede basarse en una combinación de factores como puertos, direcciones MAC, subred, hora del día, forma de acceso, condiciones de seguridad del equipo...

Las VLANs están definidas por los estándares IEEE 802.1D, 802.1p, 802.1Q y 802.10.

2. Redes Virtuales Privadas (VPN)

Una red privada virtual VPN de las siglas en inglés de Virtual Private Network, es una tecnología de red que permite una extensión segura de la red local (LAN) sobre una red pública o no controlada como Internet. Permite que la computadora en la red envíe y reciba datos sobre redes compartidas o públicas como si fuera una red privada con toda la funcionalidad, seguridad y políticas de gestión de una red privada.

Tipos de VPN

Básicamente existen cuatro arquitecturas de conexión VPN:

VPN de acceso remoto

Es quizás el modelo más usado actualmente, y consiste en usuarios que se conectan desde sitios remotos utilizando Internet como vínculo de acceso. Una vez autenticados tienen un nivel de acceso muy similar al que tienen en la red local de la empresa

VPN punto a punto

El servidor VPN, que posee un vínculo permanente a Internet, acepta las conexiones vía Internet provenientes de los sitios y establece el túnel VPN. Esto permite eliminar los costosos vínculos punto a punto tradicionales (realizados comúnmente mediante conexiones de cable físicas entre los nodos).

Tunneling

La técnica de tunneling consiste en encapsular un protocolo de red sobre otro (protocolo de red encapsulador) creando un túnel dentro de una red de computadoras. El establecimiento de dicho túnel se implementa incluyendo una PDU (unidades de datos de protocolo) determinada dentro de otra PDU con el objetivo de transmitirla desde un extremo al otro del túnel sin que sea necesaria una interpretación intermedia de la PDU encapsulada.

VPN over LAN

Una aplicación realmente desconocida pero muy útil y potente consiste en establecer redes privadas virtuales dentro de una misma red local. Una aplicación muy típica de este modelo se utiliza para aumentar la seguridad en redes de acceso inalámbrico, separándolas así de la red física para evitar posibles fugas de información o accesos no autorizados. Sirve para aislar zonas y servicios de la red interna.

3. Redes Activas y Programables

Redes activas y programables

Las redes activas (Active Networking) orientadas hacia el control de la red, conceptualizando una interfaz de programación (API) que expone los recursos (procesamiento, almacenamiento, colas de paquetes, etc) en nodos de red individuales y soporta la construcción de funcionalidades personalizadas para aplicar a un subconjunto de paquetes que pasan a través del nodo.

4. Redes Overlay

Una **red overlay** es una red virtual de nodos enlazados lógicamente, que está construida sobre una o más redes subyacentes. Su objetivo es implementar servicios de red que no están disponibles en la red subyacente. Las redes superpuestas pueden apilarse de forma que tengamos capas que proporcionen servicios a la capa superior.

1.3 Redes Actuales

Debido a la cantidad de dispositivos conectados y cada vez las mayores cantidades de flujos de datos circulando por la red, las arquitecturas de redes tradicionales están restringidas por algunas **limitaciones**:

- **La complejidad**: para enfrentar las demandas cada vez más exigentes de los usuarios, se han desarrollado una serie de protocoles de red que ofrecen un mayor rendimiento, confiabilidad y seguridad más estricta, sin embargo, estos han sido definidos de manera aislada, haciendo que cada solución esté diseñada para un problema específico dejando atrás los beneficios de las soluciones generales. Esta complejidad hace que las redes actuales no puedan adaptarse dinámicamente a los cambios en el tráfico de las aplicaciones, dándoles, por el contrario, una caracterización de redes estáticas. Esta naturaleza estática hace que la asignación de los recursos sea altamente manual, ya que se deben configurar equipos de cada fabricante por separado y ajustar ciertos parámetros en función de cada sesión o aplicación; por ejemplo, en redes que operan con tráfico IP convergente para voz, datos y vídeo en las cuales se proporcionan niveles diferenciados de QoS para el manejo de diferentes tipos en las aplicaciones.

- **Políticas inconsistentes**: Para implementar una política que abarque a la red completamente, los administradores de red, deben configurar miles de mecanismos y aparatos, para implementar políticas de acceso, seguridad, calidad de servicio, etc. Lo que deja a la red vulnerable debido al incumplimiento de regulaciones, configuraciones erróneas…

- **Imposibilidad de escalabilidad**: la red se vuelve más complicada con la adición de cientos o miles de dispositivos de red que se deben configurar y gestionar manualmente.

- **Dependencia de los fabricantes**: la implementación de nuevas capacidades y servicios, en respuesta rápida a las demandas del usuario, se dificulta por el ciclo de productividad de los equipos de los fabricantes y la falta de interfaces estandarizadas que limitan la adaptación de la red a entornos particulares

1.4 La necesidad de una nueva arquitectura de red

En un principio cuando internet nació, un paquete si no podía pasar por un camino predeterminado se encaminaba por otro lugar hasta llegar a su destino, en un principio los flujos de datos eran infinitamente pequeños comparados con los grandes contenidos en streaming que hay circulando de manera continuada por la red los cuales han dejado obsoleta la organización inicial.

Algunas de las tendencias que impulsan la creación de mecanismos alternativos que permitan volver a examinar la arquitectura de las redes tradicionales incluyen:

- **La gran aceptación de dispositivos móviles:** Esto provoca un cambio significativo en los patrones de tráfico, accedemos a diversas bases de datos y a diferentes servidores provocando la comunicación entre múltiples clientes desde una gran variedad de dispositivos, conectados desde cualquier lugar y en cualquier momento.

- **La amplia diversidad de contenidos que se manejan.**

- **Los servicios en la nube:** Este aumento, principalmente debido a la gran acogida por parte de las empresas en el plano público y privado, requieren una escalabilidad dinámica de la capacidad de cómputo, almacenamiento y recursos de red.

- **Virtualización de servidores:** Dependiendo de la función que esta deba de desempeñar en la organización, todas ellas dependen del hardware y dispositivos físicos, pero casi siempre trabajan como modelos totalmente independientes de este. Cada una de ellas con sus propias CPUs virtuales, tarjetas de red, discos etc. Lo cual podría especificarse como una compartición de recursos locales físicos entre varios dispositivos virtuales.

Debido a todo lo que se ha mencionado anteriormente, se argumenta que las futuras redes deberían proporcionar dos funciones: la capacidad de virtualizar a una red física en varias particiones (slice) y la capacidad para permitir el control independiente de programación de cada partición, todo esto lo proporcionan las SDN.

2 Software Defined Networking

Las redes definidas por software, en inglés Software Defined Networking (SDN), es un conjunto de técnicas relacionadas con el área de redes computacionales, cuyo objetivo es facilitar la implementación e implantación de servicios de red de una manera determinista, dinámica y escalable, evitando al administrador de red gestionar dichos servicios a bajo nivel. Todo esto se consigue mediante la separación del plano de control (software) del plano de datos (hardware).

Las SDN definen una nueva relación entre los dispositivos de la red y el software que los controla. Los dispositivos de red son gestionados remotamente y de forma centralizada por un programa de tal forma que no tienes que presentarte físicamente a configurar el router como si de un guardia que quiere cambiar una señal de tráfico se tratase, con las SDN, no es necesario y es posible cambiar las señales de una forma centralizada que agiliza y optimiza el tráfico.

La forma en la que el programa y el dispositivo de red se entienden tiene un nombre y es el protocolo más extendido para este fin, **OpenFlow.**

2.2 La propuesta que pone en marcha las redes SDN y sus beneficios

SDN es una gran alternativa para crear redes con un gran potencial debido a la separación de los planos de datos y de control, especialmente en el plano de control que permite a los administradores de red mantener el control de sus redes de forma centralizada mediante equipos servidores llamados controladores.

Además de ofrecer redes centralizadas programables que pueden atender dinámicamente las necesidades de las empresas, SDN provee los siguientes beneficios:

- **Reduce el** Capex **(*Capital Expenditures*)**: Mediante la posibilidad de reutilizar el hardware existente, SDN limita la necesidad de invertir en hardware nuevo.
- **Reduce el** Opex **(*Operating Expense*)**: SDN permite control algorítmico de la red de elementos de red, como *switches*/*routers* (*hardware* y *software*) que cada vez son más programables, haciendo más sencillo la configuración y gestión de las redes. Además, esto permite una reducción del tiempo de gestión por parte de los administradores, lo que reduce la probabilidad de error humano.
- **Agilidad y flexibilidad**: SDN permite a las organizaciones desplegar aplicaciones, servicios e infraestructuras rápidamente para alcanzar los objetivos propuestos por empresas en el menor tiempo posible.

- **Permite innovación**: Permite crear nuevos tipos de aplicaciones y modelos de negocio por parte de las empresas, que las beneficia y aumenta el valor de sus redes.

2.3 Arquitectura de SDN

Como hemos visto en apartados anteriores las SDN separan el plano de control que es gestionado de forma centralizada por el controlador (NOX) de el plano de reenvío de paquetes

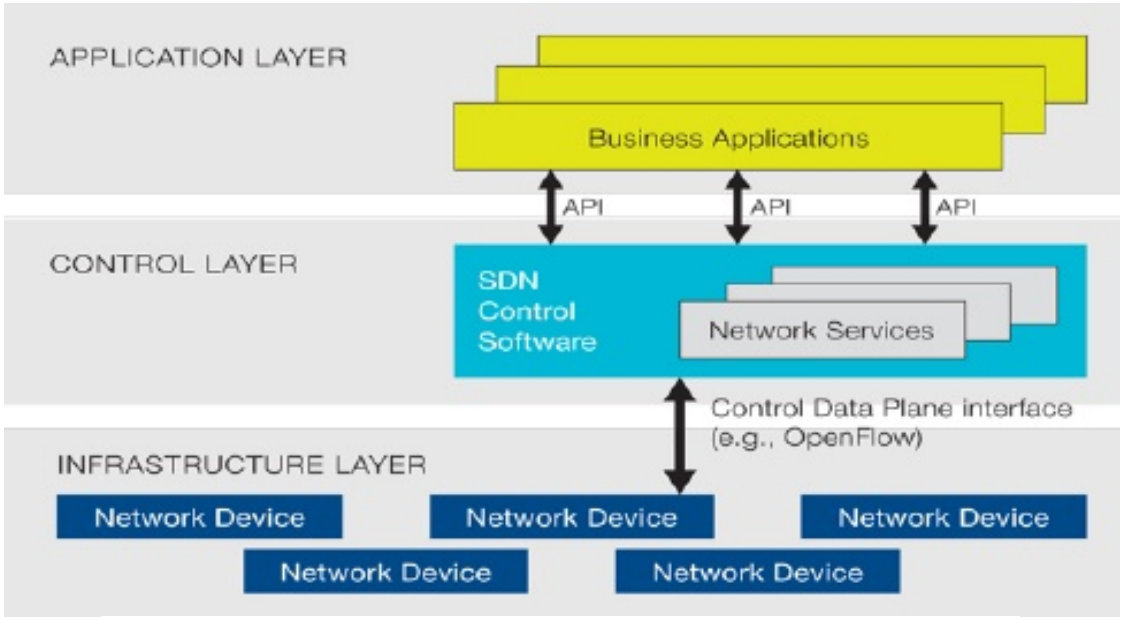


Diagrama general de la arquitectura SDN, el plano de control tiene toda la lógica SDN que separa la infraestructura de encaminamiento de las aplicaciones.

Como hemos comentado anteriormente en las redes tradicionales los routers y switches convencionales tenían una estrecha relación el plano de control y de datos, esto generaba una tarea de depuración de problemas de configuración y control muy complicada, para enfrentarse a dicha tarea, se realizó lo que tantas veces se ha aplicado a los problemas de ingeniería, separar la problemática en varias partes, la idea de dividir ambos planos empezó a florecer con las SDN.

2.3.1 Infraestructura

Formada por todos los hosts físicos o virtuales pertenecientes a la red así como todos los medios de transmisión que permiten la comunicación en la red, como routers y switches que están corriendo el protocolo SDN.

2.3.2 Plano de datos

Representa los datos reales de los usuarios. Por ejemplo, los bits de información contenidos en los flujos de datos de un circuito óptico que lleva un servicio o múltiples servicios.

Se ha realizado mediante la abstracción de capas (física, enlace de datos, red, transporte, sesión – Modelo OSI)

2.3.3 Plano de control

Es la entidad donde reside parte de la inteligencia de la red. Automatiza funcionalidades dentro de una red, como añadir o eliminar circuitos y restaurarlos una vez que se ha solventado el fallo que produjo su caída. Abarca protocolos de señalización, descubrimiento la topología, reserva de recursos, cálculo de caminos y cálculos de enrutamiento e información a intercambiar.

De esta parte en una arquitectura SDN se encarga el controlador centralizado, las aplicaciones de control las cuales permiten manejar flujos basándose en distintos patrones se ejecuta en dicho controlador, se hablará más de estos en secciones posteriores.

3 OpenFlow

3.1 Introduccion y antecedentes

OpenFlow es una tecnología de switching que surgió a raíz del proyecto de investigación *Ethane* de 2008 en la Universidad de Stanford a raíz de que los ingenieros de Standford y Berkeley pensaran que era necesario abstraer el control de datos del hardware de sus equipos de conmutación, debido a que esta configuración se hacía de forma individual, por esto implementaron un cerebro a la red, el cual se encarga de realizar las tareas que se programen y luego replicarla en todos los dispositivos (controlador)

Es un protocolo emergente y abierto que permite a un servidor llamado controlador determinar el camino de los paquetes que debería seguir una red.

OpenFlow podría llegar a las redes presentes en un futuro puesto que existe un gran compromiso con fabricantes de dispositivos de red como Cisco, HP, T-Mobile. En los routers podrían integrarse en su firmware mediante actualizaciones.

OpenFlow permite que la red sea gestionada como un todo, no como un número de dispositivos individuales.

En los switches OpenFlow la parte del datapath reside en el mismo switch pero es el controlador el que decide el encaminamiento de alto nivel, esto provoca una mayor eficiencia en el uso de los recursos de la red comparándola con una red tradicional.

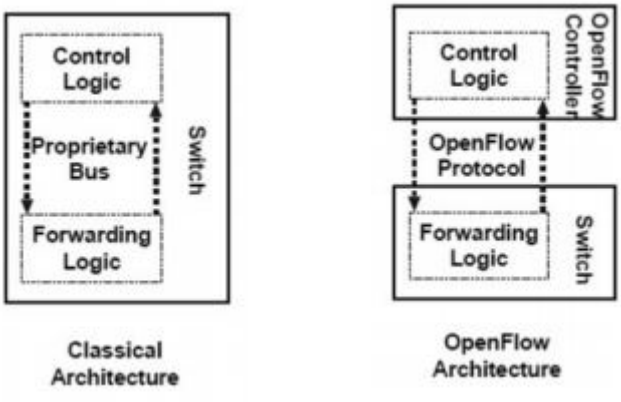
OpenFlow permite acceder directamente y manipular el plano de redireccionamiento de dispositivos de red como conmutadores y enrutadores, es decir facilita a la hora de programar permite configurar una capa de control para poder centralizar la inteligencia de la red y brinda la capacidad de programarla tal como lo anuncia la tecnología SDN.

3.2 SDN no es Openflow

A menudo se apunta a Openflow como sinónimo de SDN, pero en realidad, es simplemente un elemento que forma parte de la arquitectura SDN. Openflow es un estándar abierto para un protocolo de comunicaciones que permite al plano de control interactuar con el plano de datos (Openflow, sin embargo, no es el único protocolo disponible o en desarrollo para SDN, aunque sí está convirtiéndose en el modelo estándar de implementación de una SDN).

3.3 Arquitectura clásica contra arquitectura Openflow.

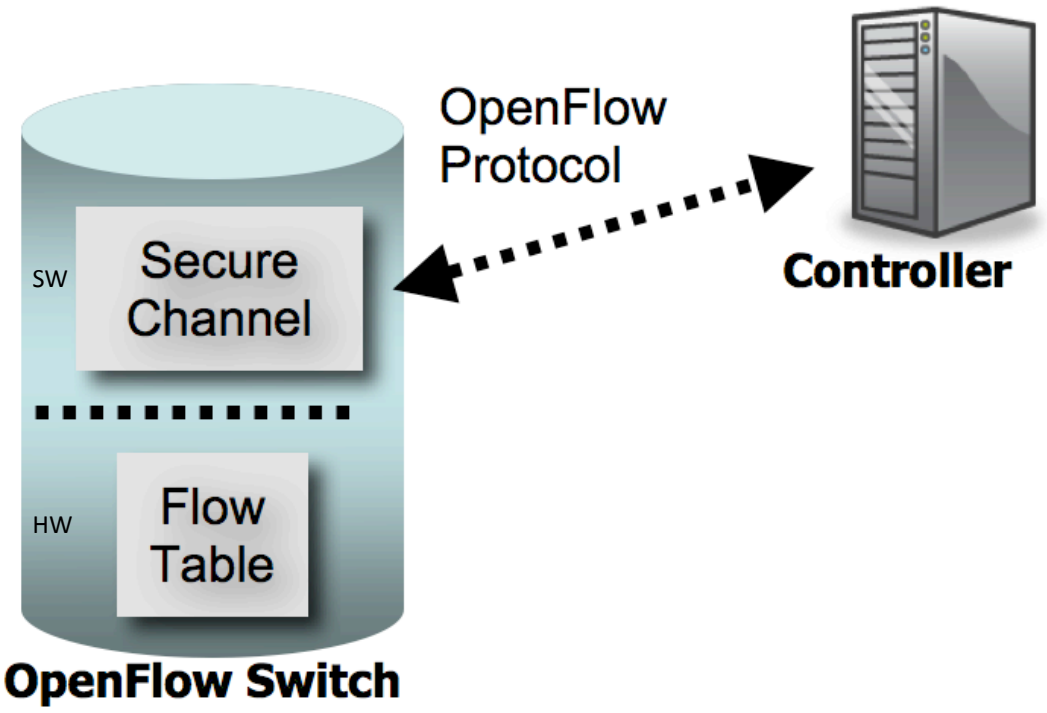
Los dispositivos actuales utilizan firwmare específico y propietario en el cual está predefinido como se van a tratar los paquetes que redireccione por lo que son difíciles de integrar.



Relativa al hardware tradicional, el protocolo OpenFlow traslada el datapath a un controlador externo

3.4 Openflow Switch

Hablaremos de las especificaciones de un OFS en su última versión en vigor de OpenFlow 1.4.0. Estos utilizan las tradicionales tablas de flujos (Flow-Tables) que implementaban firewalls, NAT, Qos, recolectar estadísticas, estas tablas eran propias de cada fabricante de switches, pero en un OFS se han abstraído las características y funciones comunes de todas ellas.



Arquitectura switch-controllador OpenFlow

Un OpenFlow Switch (OFS) consta de una o más flow tables y una group table, un canal OpenFlow (TCP/SSL) capa 4 y 5 para la comunicación entre el controlador y el plano de control del enrutador.Para todo esto el Switch OpenFlow se ve precisado de unos componentes los cuales se describen a continuación.

3.4.1 Componentes del Switch Openflow

- **Tabla de flujos**: Entradas asociadas a cada flujo y una acción que determina cómo debe tratarlo.
- **SSL** (Secure Sockets Layer): Capa de conexión segura, protocolo de conexión para la comunicación entre el controlador y el dispositivo.
- **OFP** (OpenFlow Protocol): Estándar abierto de comunicación entre el controlador y los dispositivos.

3.4.1.1 Tabla de flujos:

Contiene asociada una regla para identificar el flujo, una acción que aplicar al flujo y un contador para estadísticas

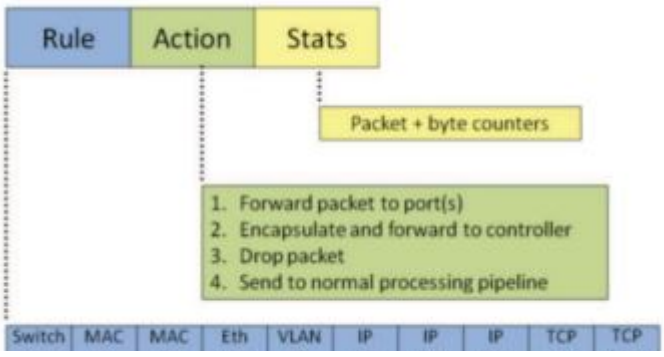


Figura. Campos en una entrada de flujo

Se identifica principalmente por sus coincidencias y prioridad. Ambos campos identifican un flujo único en la Tabla de Flujos. Podemos citar varios campos principales:

- **Match fields**: puerto y cabecera. Opcionalmente metadatos especificados en una tabla anterior.
- **Priority**: coincidencia con el flujo de entrada.
- **Counters**: Se actualiza cuando se encuentra una coincidencia y sirven para hacer estadísticas.
- **Instructions**: Acción a aplicar al flujo.
- **Timeouts**: tiempo máximo antes de que el switch descarte el flujo.
- **Cookie**: Valor que elige el controlador para filtrar las estadísticas, las modificaciones y el borrado de los flujos. No se usa: cuando se procesan los paquetes.

También los OFS contienen **tablas de grupos**, los cuales son una manera eficiente de indicar que el mismo set de instrucciones deben aplicarse a múltiples flujos, cada tabla de flujo contiene:

- **Identificador de grupo**: entero sin signo.
- **Tipo de grupo**: Para determinar su semántica.
- **Contador**: Para llevar el número de paquetes procesador por el grupo.
- **Set de acciones**: a aplicar.

3.4.1.2 Canal seguro SSL

El canal OpenFlow que conecta el switch y el controlador para actualizar la información del datapath se realiza a través de un canal seguro utiliza un protocolo criptográfico llamado SSL (Transport Layer Security) con el fin de hacer la red más segura y que sea realmente el controlador autorizado el que está cambiando la configuración del OFS.

3.4.2 Tipos de Switches Openflow

Lo distintos tipos de switches que soportan el protocolo OpenFlow son:

- Switches dedicados a OpenFlow: Funcionan únicamente a través de este protocolo, sus capas 2 y 3 no son compatibles con las redes tradicionales.
- Switches habilitados para Openflow: Switches tradicionales en los cuales se ha añadido una nueva acción, *forward* (reenviar) los paquetes pertenecientes a un flujo a través del procesamiento en paralelo de estos con los paquetes tradicionales.
- Switches “type 0”: Compatible con los formatos de cabecera y las acciones básicas descritas anteriormente

3.5 Funcionamiento de un Switch Openflow

Usando OpenFlow el controlador puede añadir, actualizar y borrar entradas en la flow table, cada flow table contiene un conjunto de tablas, contadores para estadísticas y un conjunto de instrucciones que indican la acción a aplicar al flujo.

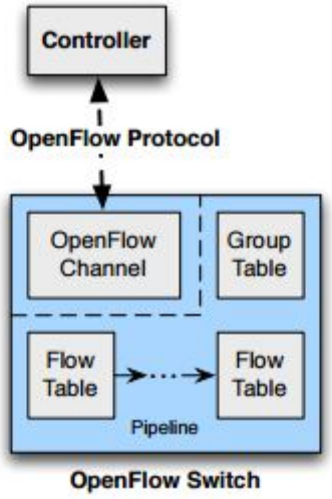
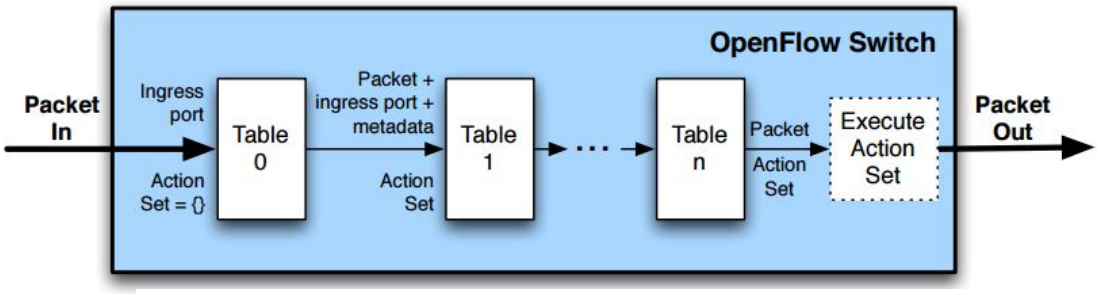


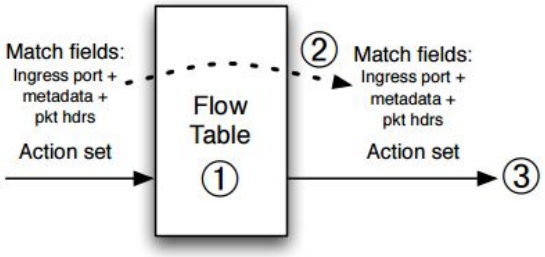
Figure 1: Main components of an OpenFlow switch.

3.5.1 Matching

Cuando llega un paquete a un OFS (Openflow Switch) se compara con todas las entradas de la primera flow table la 0, existe infinidad de parámetros para especificar un flujo (puertos, etiqueta vlan, destino, puerto ip….), si no encuentra coincidencia se actualiza el número de la tabla de flujo y salta a otra tabla de flujo dentro del pipeline, hasta encontrar la coincidencia la cual tendrá una acción asociada en su tabla de acuerdo a la política establecida desde el controlador, si no encuentra una coincidencia será enviado al controlador para crear un nuevo flujo o se descartará el paquete según la configuración del controlador.



Los paquetes encuentran su flujo en las diferentes tablas de flujo a través de una estructura en pipeline



1. Encuentra la coincidencia con mayor prioridad
2. Aplica instrucción:

a. Forward: encaminar el paquete a través de la red. En la mayoría de los switches sucede a la velocidad de la línea..

b. Encapsulación y reenvío: Cuando llega un paquete que no encuentra coincidencia sus tablas de flujo, se encapsula en SSL y se envía al controlador.
c. Descartar este flujo de paquetes. Puede ser usado por seguridad , para parar ataques de denegación de servicio o reducir el falso tráfico de descubrimiento broadcast desde los hosts finales.

3. Manda el paquete a la siguiente tabla.

3.5.1 Meter tables

Con estas tablas se mide la tasa de paquetes asignadas a ellas y se facilita el control de este. Consiste en entradas de medidas por flujo esto proporciona a OpenFlow implementar operaciones de QoS simples, como limitar el tráfico.

Band Type	Rate	Counters	Type specific arguments

Table 4: Main components of a meter band in a meter entry.

- **Band type**: Entero sin signo de 32 bytes que identifica unívocamente.
- **Rate**: contiene el ancho de banda mínimo asignado al flujo.
- **Counter**: Se actualiza cuando un paquete es procesado por la meter band
- **Type specific arguments**: Argumentos opcionales.

3.5.2 Counters

Se utilizan para hacer estadísticas de todo tipo asociadas a las tablas, entradas de tabla, puertos, colas, grupos, meter band, etc…

Un OFS no debe contener todos los contadores, sólo aquellos marcados como requeridos, si un contador especifico no está disponible en un switch su valor debe ser el máximo de este.

Counter	Bits	
Per Flow Table		
Reference Count (active entries)	32	*Required*
Packet Lookups	64	*Optional*
Packet Matches	64	*Optional*
Per Flow Entry		
Received Packets	64	*Optional*
Received Bytes	64	*Optional*
Duration (seconds)	32	*Required*
Duration (nanoseconds)	32	*Optional*
Per Port		
Received Packets	64	*Required*
Transmitted Packets	64	*Required*
Received Bytes	64	*Optional*
Transmitted Bytes	64	*Optional*
Receive Drops	64	*Optional*
Transmit Drops	64	*Optional*
Receive Errors	64	*Optional*
Transmit Errors	64	*Optional*
Receive Frame Alignment Errors	64	*Optional*
Receive Overrun Errors	64	*Optional*
Receive CRC Errors	64	*Optional*
Collisions	64	*Optional*
Duration (seconds)	32	*Required*
Duration (nanoseconds)	32	*Optional*
Per Queue		
Transmit Packets	64	*Required*
Transmit Bytes	64	*Optional*
Transmit Overrun Errors	64	*Optional*
Duration (seconds)	32	*Required*
Duration (nanoseconds)	32	*Optional*
Per Group		
Reference Count (flow entries)	32	*Optional*
Packet Count	64	*Optional*
Byte Count	64	*Optional*
Duration (seconds)	32	*Required*
Duration (nanoseconds)	32	*Optional*
Per Group Bucket		
Packet Count	64	*Optional*
Byte Count	64	*Optional*
Per Meter		
Flow Count	32	*Optional*
Input Packet Count	64	*Optional*
Input Byte Count	64	*Optional*
Duration (seconds)	32	*Required*
Duration (nanoseconds)	32	*Optional*
Per Meter Band		
In Band Packet Count	64	*Optional*
In Band Byte Count	64	*Optional*

Table 5: List of counters.

3.6 Servidor controlador

Como ya hemos comentado anteriormente el controlador centraliza todas las configuraciones de los elementos de la red, se podría decir que funciona como un sistema operativo de red, ya que tiene una visión global de los flujos que atraviesan la red. El controlador se comunica mediante el protocolo Openflow con los elementos de la red.

La configuración de los flujos pueden ser de dos maneras:

- **Proactiva**: Antes de que un paquete llegue al switch, dando lugar a insignificantes retrasos.
- **Reactiva**: Cuando un OFS recibe un paquete que no coincide con ninguna entrada tiene que enviarlo el controlador y es este quien decide qué hacer con el mismo, sumando 3 retrasos, el de envio, el de procesamiento por el controlador, y el tiempo que se escribe una regla en la tabla de flujo del OFS.

3.7 Tipos de controlador

Describiremos las diferentes herramientas de software que están implementadas para mantener el control de la red, mencionar que analizaremos todas ellas para elegir el software idóneo para los diferentes roles que puede tener una red.

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

Distintos controladores y algunas de sus características

3.7.1 Elementos para la selección de controlador

La selección del controlador es vital para el diseño de una red SDN.

- **Soporte OpenFlow**: Al elegir un controlador los administradores de red necesitan conocer las características de las versiones de OpenFlow que el controlador soporta, así como las posibilidades que ofrece el proveedor para migrar a las nuevas versiones del protocolo, tales como la v1.3 y la v1.4. Una razón por la que esto es necesario, es que algunas funciones importantes como, por ejemplo, el soporte de IPv6 no es parte de OpenFlow v1.0 pues se incluyen a partir del estándar OpenFlow v1.2.

- **Virtualizacion de red**: Debido a los beneficios que ofrece la virtualización de red, un controlador SDN debe soportarla. Esta característica permite a los administradores crear dinámicamente las redes virtuales basadas en políticas, disociadas de las redes físicas, para satisfacer una amplia gama de requisitos.

- **Funcionalidad de la red**: Para lograr mayor flexibilidad en términos de cómo los flujos son enrutados, es importante que el controlador SD pueda tomar decisiones de enrutamiento basado en múltiples campos de la cabecera de OpenFlow.

- **Escalabilidad**: En la actualidad se debe esperar que los controladores soporten un mínimo de 100 switches, pero en última instancia esto depende de las aplicaciones que soportan. Otro factor que limita la escalabilidad de una red SDN es la proliferación de entradas en la tabla de flujo, ya que sin algún tipo de optimización, se requiere de una entrada salto por alto para cada flujo. Al evaluar los controladores SDN, es necesario asegurarse que el controlador puede disminuir el impacto de sobrecarga de difusión de red, la cual limita la escalabilidad de la arquitectura de red implementada y reducir al mínimo la proliferación de las entradas de la tabla de flujo.

- **Rendimiento**: dos de los indicadores claves de rendimiento asociados con un controlador SDN son el tiempo de conformación de flujo y el número de flujos por segundo que puede establecer el controlador, esto está asociado en gran medida a la configuración reactiva, cuando por ejemplo los switches inician más flujos de los que el controlador puede soportar, creando grandes retrasos en la red.

Existen herramientas de evaluación de controladores como *Cbench* y *Hcprobe* para realizar pruebas d rendimiento, escalabilidad, disponibilidad y seguridad.

3.7.2 POX

Es un controlador desarrollado a partir de NOX para cubrir los requerimientos de las SDN usando Python en Windows, Mac o Linux. Es uno de los *frameworks* de desarrollo más creciente y el que utilizaremos en la línea de investigación de este TFG.

3.7.2.1 Los componentes de POX

En esta sección se detallan algunos de los componentes

Py

Es el encargado de arrancar el intérprete de Python para la depuración y ajuste interactivo.

forwarding.l2_learning

Permite que los switches OpenFlow actúen como un switch de capa 2.

forwarding.l2_pairs

Este componente permite que los *switches* OpenFlow actúen como un tipo L2, con la diferencia de que instala reglas basadas exclusivamente en las direcciones MAC.

forwarding.l3_learning

Este componente no define el comportamiento de un *router*, pero tampoco es un *switch* de nivel 2. POX usa una biblioteca de paquetes para examinar y elaborar solicitudes y respuestas ARP.

forwarding.l2_multi

Este componente se puede ver como un switch de aprendizaje. El aprendizaje de los otros switches se realiza sobre una base de conexión de switch a switch, tomando las decisiones de conmutación, como si cada switch tuviera solo información local, para aprender la topología de toda la red.

openflow.spanning_tree

Este componente utiliza el componente de descubrimiento para construir una vista de la topología de la red, construye un árbol de expansión[41], desactivando los *floodings* en los puertos del *switch* que no están en el árbol para que las topologías queden libres de bucles.

web.webcore

Este componente inicia un servidor web dentro del proceso POX. Otros componentes pueden interactuar con él para proporcionar su propio contenido web estático y dinámico.

messenger

Este componente proporciona una interfaz para interactuar con los procesos externos bidireccionales a través de mensajes basados en JSON[42]. Es una API para la comunicación a través de medios de transporte usando los *sockets* TCP y HTTP.

misc.arp_responder

Es un componente con el que se puede responder las peticiones ARP.

misc.dns_spy

Este componente supervisa las respuestas DNS y muestra sus resultados.

misc.mac_blocker

Este componente está destinado a ser utilizado junto con algunas aplicaciones de reenvío, tales como *l2_learning* y *l2_pairs*. Abre una interfaz gráfica de usuario que permite bloquear direcciones MAC.

openflow.of_01

Este componente se comunica con *switches* OpenFlow 1.0. Por lo general, se inicia de forma predeterminada (a menos que se especifique la opción *no-OpenFlow* en la línea de comandos).

openflow.discovery

Este componente envía mensajes LLDP de conmutadores OpenFlow para que pueda descubrir la topología de la red.

openflow.debug

Cargando este componente hará que POX pueda crear trazas con mensajes OpenFlow, que luego se pueden cargar en Wireshark para ser analizados.

openflow.keepalive

Este componente hace que POX pueda enviar solicitudes periódicas *echo* a los *switches* conectados.

4 MININET

4.1 Introduccion

Es una plataforma de emulación de red, que crea redes definidas por software (Tipo OpenFlow por ejemplo) totalmente escalables (cuyas dimensiones pueden ser de hasta cientos de nodos, según la configuración deseada) que están contenidas en una PC que utiliza procesamiento Linux.

MiniNet permite crear, interactuar, personalizar y compartir de forma rápida un prototipo de red definido mediante software al mismo tiempo proporcionar un camino fácilmente adaptable para la migración a hardware.

Los atributos en los que se basa MiniNet son los siguientes:

- **Flexible**: Soporta diferentes tecnologías y funcionalidades de las nuevas tecnologías de red basadas en software.
- **Desplegable**: La implantación correcta de un prototipo SDN no debe exigir cambios en el código.
- **Interactivo**: La gestión y la operación de la red debe realizarse en tiempo real, como si no se tratase de una red virtualizada.
- **Escalable**.

Topología	Hosts	Switches	Setup (sg)	Stop(sg)	Mem (MB)
Minimal	2	1	1.0	0.5	6
Linear(100)	100	100	70.7	70.0	112
VL2(4, 4)	80	10	31.7	14.9	73
FatTree(4)	16	20	17.2	22.3	66
FatTree(6)	54	45	54.3	56.3	103
Mesh(10, 10)	40	100	82.3	92.9	152
Tree (4^4)	256	85	168.4	83.9	233
Tree (16^2)	256	17	139.8	39.3	212
Tree (32^2)	1024	33	817.8	163.6	492

Benchmarks realizados sobre distintas topologías con Mininet: Setup time, stop time y memoria usada. Testeado bajo debían 5 / Linux 2.6.33.1 en un MacBook Pro (2,4 GHz / 6 GB)

- **Realista**: el comportamiento de la red debe coincidir con la real.
- **Compartible**: Compartición de los diferentes prototipos para poder realizar pruebas de todos ellos en diferentes experimentos.

La principal limitación de MiniNet es la pérdida de fidelidad en su rendimiento especialmente ante grandes cargas debido al multiplexado en el tiempo de los recursos de la CPU planificado por defecto en Linux.

Tipo de operación	Tiempo (ms)
Crear un nodo (host / switch / controlador)	10
Ejecutar un comando sobre un host	0.3
Añadir un enlace entre dos nodos	260
Eliminar un enlace entre dos nodos	416
Iniciar un switch en el espacio de usuario	29
Detener un switch en el espacio de usuario	290
Iniciar un switch en el kernel	332
Detener un switch en el kernel	540

Benchmarks realizados sobre las distintas aplicaciones de Mininet.
Testeado bajo debían 5 / Linux 2.6.33.1 en un MacBook Pro (2,4 GHz / 6 GB)

4.2 Conceptos previos

Wireshark: Wireshark es el analizador de protocolos más utilizado que hay actualmente, permite la captura y búsqueda interactiva del tráfico que atraviesa una red, actualmente se considera un estándar a nivel no solo educativo sino también industrial y comercial. Para ver el tráfico de control de OpenFlow ejecutamos *"$ sudo wireshark &"*

Iperf: Utilidad de comandos utilizara para la evaluación de rendimientos en las comunicaciones de una red local y posterior optimización de los parámetros, Con IPerf es posible medir el ancho de banda y rendimiento de una conexión entre dos host. Se trata, básicamente de una herramienta cliente-servidor.

SSH(SecureShell): SSH(o Secure SHell): es un protocolo que facilita las comunicaciones seguras entre dos sistemas usando una arquitectura cliente/servidor y que permite a los usuarios conectarse a un host remotamente. A diferencia de otros protocolos de comunicación remota tales como FTP o Telnet, SSH encripta la sesión de conexión, haciendo imposible que alguien pueda obtener contraseñas no encriptadas.

4.3 Crear topología de red

Disponemos de varias formas por las cuales podemos crear una red en MiniNet:

- A través de la utilidad mn
- Mediante aplicaciones gráficas.
- Mediante código usando una simple API Python

4.2.1 Utilidad mn

El comando `mn` arranca el entorno de MiniNet y crean topologías por defecto. La creación de estas topologías es sumamente fácil y rápida, pero limitada a la hora de pretender hacer una topología personalizada.

Con este comando se pueden crear topologías parametrizadas por defecto, invocan el CLI de MiniNet (prompt cambiará a "MiniNet>") y se ejecutan test.

Aunque este comando admite muchos más parámetros los cuales se describirán a continuación

- **h**

  Para mostrar la ayuda del comando mn

- **switch= ivs|ovsk|ovsl|user**

  Configurar el switch

- **controller= (none|nox|ovsc|ref|remote)**

  La sintaxis general del parámetro es:

  ```
  --controller remote, ip=[controller IP], port=[controller port]
  ```

  De esta manera, se podría instalar un controlador en cualquier lugar del mundo. Si no se especifican, se toman los valores IP 127.0.0.1 port 6633. Estos valores se corresponden con la máquina virtual.

- **topo= linear | Minimal | Reversed | Single | tree**

  Con este parámetro se crean una serie de topologías sencillas

  Linear,N

  Esta topología consta de un determinado número de conmutadores interconectados de forma lineal (host---switch----switch---host)

  Minimal

  Crea una topología muy simple con un conmutador y dos hosts conectados a él.

  Single, N

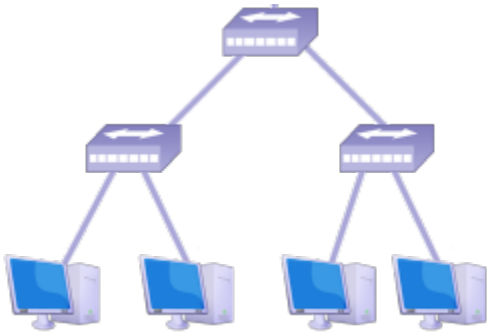
  Esta topología consiste en un único conmutador conectado a un número determinado de hosts (N).

  Tree

  En este caso es possible crear una topologia en forma de arbol como su propio nombre indica, el comando con más detalle se muestra a continuación:

  ```
  $sudo mn –-topo tree,N,M
  ```

  Se tienen dos variables N, que es la profundidad que desea tener el árbol, es decir, el número de conmutadores interconectados y M el esparcimiento o lo que es lo mismo, el número de hosts conectados a cada conmutador del último nivel.



Topologia en árbol con M y N = 2

- **`c`**
  Limpiar la topología y salir.

- **`custom=CUSTOM`**
  Leer y cargar un script de una topología MiniNet en python.

- **`test= cli|build|pingall|pingpair|iperf|all|iperfudp|none`**
  Realizar un test en cuanto se cargue la topología

- **`x`**
  Abrir una ventana de terminal de cada dispositivo de la red.

- **`i IPBASE`**
  IP base para los hosts.

- **`Mac`**
  Automaticamente asigna macs sencillas a las interfaces de la red.

- **`Arp`**
  Rellena todas las tablas arp de la red.

- **`v  info|warning|critical|error|debug|output`**
  Diferentes métodos de ejecución, mencionar aquí que el método debug es muy interesante para desarrollos, ya que nos imprime por pantalla bastante información la cual sin este parámetro se ocultaba.

- **`Versión`**
  Elegir versión MiniNet.

4.2.2 Aplicaciones Gráficas para la topología

El código fuente MiniNet incluye un script GUI en Python para la creación de topologías de red, MiniEdit que la comentamos en el siguiente apartado de aplicaciones MiniNet.

También he encontrado una aplicación web externa en la cual podremos exportar desde la web el script de creación de una topología de MiniNet.

http://www.ramonfontes.com/vnd/#

4.3 Aplicaciones MiniNet y comandos de utilidad

MiniNet ofrece un conjunto de aplicaciones en Python para apoyarte a la hora de comenzar a trabajar, en este apartado se mencionarán las más importantes.

Consoles.py:

Aplicación para generar ventanas, una para cada nodo, permitiendo la interacción con ellos.

Emptynet.py:

Esta aplicación genera una red vacía.

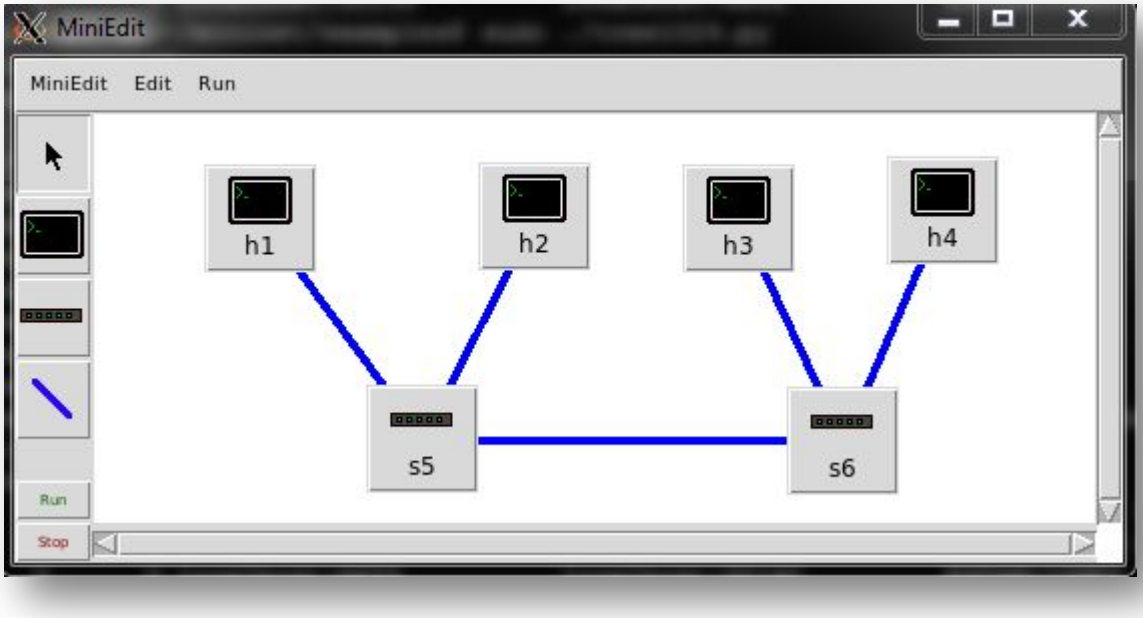
Linearbandwidth.py:

Aplicación que crea diferentes topologías de red y realiza pruebas sobre ellas.

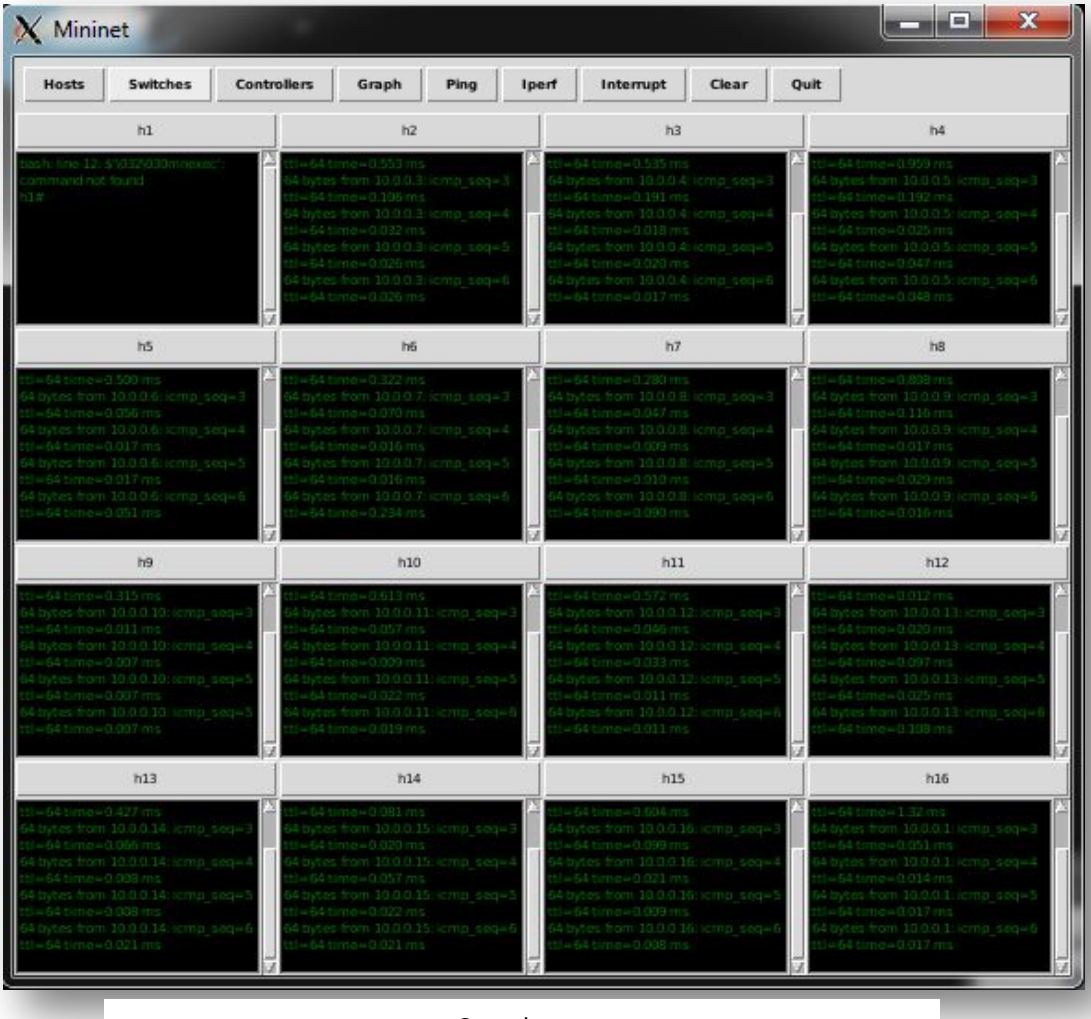
Miniedit.py:

Editor gráfico para generar topologías.

Destacar miniedit.py y consoles.py por su utilidad a la hora de crear topologías y su posterior estudio.



MiniEdit.py



Consoles.py

Una vez en MiniNet con una topología a simular, hay una serie de *comandos* sencillos que muestran información de la red:

mininet>nodes

Muestra los nodos disponibles

mininet>net

Muestra información sobre la red, nodos y enlaces disponibles.

mininet>dump

Muestra información sobre las direcciones IP de cada nodo

mininet>iperf src dst

Mininet también incorpora la herramienta Iperf. Es una herramienta que establece sesiones TCP (o UDP) para calcular el ancho de banda entre dos hosts, si se obvian src y dst se medirá entre el primer y último host.

mininet>xterm hosts

Una forma de acceder a un terminal para cada componente de la red es abrir un emulador de terminal Xterm desde MiniNet. Para ello tiene que estar corriendo un gestor de ventanas X11 (Xming para Windows).

mininet>link [node1][node2][up or down]

Crea o elimina un o link entre dos nodos

mininet>pingall

Prueba conectividad entre dos nodos

mininet>help

Ayuda, muestra lista de comandos de MiniNet

mininet> exit

Salir del CLI

5 Línea de investigación

Dado que las empresas modernas enfrentan grandes cantidades de datos cada vez más sofisticados, la eficiencia subyacente de la red del centro de datos se está convirtiendo en un problema. Los switches y routers han utilizado tradicionalmente protocolos estandarizados como OSPF (Open Shortest Path First), RIP (Routing Information Protocol) para mover paquetes de tráfico en la red, pero a medida que más cargas de trabajo compiten por el limitado ancho de banda, el énfasis se está desplazando hacia una mayor eficiencia de tráfico. El naciente protocolo OpenFlow tiene como objetivo proporcionar esta eficiencia.

5.1 Propuesta de implementación

Entre los puntos clave de este trabajo se encuentra implantar a pequeña escala una red basada en OpenFlow cargando distintos controladores ya definidos a los cuales se le hará una pequeña modificación aportando a la línea de investigación, esta modificación será un pequeño control de acceso muy utilizado en la tecnología WiFi, este control de acceso no es más que una White list de Mac's las cuales podrán enviar paquetes por la red, el resto no.

Explicaremos con detalle los siguientes controladores:

1. **forwarding.hub**
/home/mininet/pox/pox/forwarding/hub.py

2. **forwarding.l2_pairs**
/home/mininet/pox/pox/forwarding/l2_pairs.py

3. **forwarding.l3_learning**
/home/mininet/pox/pox/forwarding/l3_learning.py

4**. forwarding.l2_multi**
/home/mininet/pox/pox/forwarding/l2_multi.py

Después añadiremos alguna funcionalidad a alguno de los módulos forward, lo veremos más adelante.

5.2 Herramientas Software y Hardware necesarias

En el tema Hardware para el objetivo de implementación de este TFG no requiere de un vasto equipamiento, un simple equipo de propósito general con 2 GB de memoria RAM y al menos 10 GB de espacio libre de disco duro sería suficiente.

Las pruebas pueden ser llevadas a cabo en diferentes SO, aunque se recomienda Linux ya que necesitaremos un menor número de paquetes y herramientas a descargar dado que el mismo SO trae por defecto muchos de los requerimientos.

Las pruebas las he realizado en un equipo Windows virtual izando un SO Linux en su distribución de Ubuntu 12

Por otra parte, comentar las herramientas Software requeridas tanto de virtualización como de conexión SSH a la máquina virtual:

- **Un terminal que soporte SSH**, en nuestro caso utilizaremos Putty.
- **Un servidor X**, en nuestro caso utilizaremos Xming es un emulador de terminal para el sistema de ventanas Windows
- **X**, proporciona compatibilidad con terminales DEC VT102/VT220 y para programas que no pueden usar el sistema Windows de forma directa. El terminal Xming está conectado aun host en la red virtual.
- Un software de virtualización de SO como **Virtual Box**.
- **Una imagen virtual** en este caso una distribución Ubuntu con todos los paquetes necesarios instalados:
    - **MiniNet**: Un simulador de redes ligero y fácil de usar programado en Python.
    - **Ovswitch**: Acrónimo de Open Virtual Switch, que no es más que un conmutador implementado en software. Es el que utiliza MiniNet y además es 100% compatible con OpenFlow.
    - **POX**: Controlador SDN escrito en Python y compatible con OpenFlow como comentamos en líneas anteriores.

Dpctl

Es una herramienta de línea de comandos para monitorear y administrar datapaths OpenFlow. Es capaz de mostrar el estado actual de un camino de datos, incluidas las funciones, la configuración y las entradas de las tablas. Cuando se utiliza el módulo del kernel OpenFlow, dpctl se utiliza para agregar, eliminar, modificar y controlar datapaths.

dpctl [options] command [switch] [args&...]

La mayoría de los comandos dpctl toman un argumento que especifica el método para conectar a un conmutador OpenFlow. Los siguientes métodos de conexión son compatibles:

- **ssl**: host [: puerto] El puerto SSL especificado (por defecto: 6633) en el host remoto determinado.
- **tcp**: host [: puerto] El puerto TCP especificado (por defecto: 6633) en el host remoto determinado.

Los siguientes comandos para darnos información de los datapaths:

Show *switch*

Imprime configuración del Switch que incluye información sobre sus tablas de flujo y puertos.

Status *switch* [*key*]

Imprime a la consola una serie de pares de valores clave que informan el estado del interruptor.

show-protostat *switch*

Imprime información de estadísticas del protocolo OpenFlow

dump-tables switch

Imprime información estadísticas de cada una de las tablas de flujo que conforman el switch

dump-ports *switch* [*port number*]

Imprime datos estadísticos de las diferentes interfaces del Switch

mod-port *switch netdev action*

Modifica características de una interfaz del switch. Netdev puede ser referido por su número de puerto asignado a OpenFlow o por el nombre de interfaz, ejemplo eth0. Las acciones pueden ser una de las siguientes:

up

Levanta la interfaz. Equivalente “ifconfig up" en un Sistema Unix.

down

Apaga la interfaz. Equivale a “ifconfig down” en un Sistema Unix.

flood

Todo el tráfico del switch será enviado por esta interfaz, será un puerto de monitorización.

noflood

El tráfico del switch no será enviado por esta interfaz, esto tiene la utilidad de prevenir loops cuando el protocolo STP no está en uso.

dump-flows *switch* [*flows*]

Imprime por consola todas las entradas de flujo del datapath coincidentes con un flujo en concreto, si el argumento flows se omite se mostraran todos los flujos.

Desc *switch string*

Establece una descripción para un switch.

add-flow *switch flow*

Añade una entrada de flujo a las tablas del switch.

add-flows *switch file*

Añade entradas de flujo tal y como se describen en el archivo.

mod-flows *switch flow*

Modifica las acciones asociadas a cada entrada de flujo

del-flows *switch* [*flow*]

Borra entradas de flujo del datapath

Monitor *switch*

Se conecta a un switch e imprime por consola todos los mensajes OpenFlow recibidos.

5.4 API de POX

Las subsecciones siguientes proporcionan detalles sobre algunas de las API's POX que deberían ser útiles para la implementación de los controladores anteriormente descritos.

La documentación completa está disponible en la página web de POX.
https://openflow.stanford.edu/display/ONL/POX+Wiki

ofp_action_output class

Esta es una acción para su uso con `ofp_packet_out` y `ofp_flow_mod`. En él se especifica un puerto del switch por el que se desea enviar el paquete. También puede tomar varios números de puerto "especiales". Un ejemplo de esto sería OFPP_FLOOD que envía el paquete a todos los puertos excepto el paquete originalmente llegó.

```
out_action = of.ofp_action_output(port = of.OFPP_FLOOD)
```

ofp_match class

Esta clase contiene objetos que describen los campos de la cabecera de un paquete, estos son todos opcionales.

Los objetos más importantes de esta clase son:

dl_src: MAC fuente

dl_dst: MAC destino

in_port: El puerto de entrada.

ofp_packet_out OpenFlow message

Esta clase nos permite enviar paquetes desde el switch, ya sea construidos en el controlador, recibido, guardado en el buffer o reenviado al controlador

Los campos más interesantes son:

Buffer_id: identifica el buffer que deseas enviar.

Data: datos en bruto que deseas enviar.

Actions: diferentes acciones como `ofp_action_output`.

In_port: El número de puerto por el cual se recibió el paquete

ofp_flow_mod OpenFlow message

Esta instrucción sirve para instalar una entrada en una flow table.

Los campos más interesantes son:

idle_timeout: Tiempo de validez de una entrada que no se ha utilizado, por defecto no tiene.

hard_timeout: Tiempo en segundos antes de que se borre la entrada de flujo. Por defecto no tiene.

Acciones: Una lista de acciones a realizar (por ejemplo, ofp_action_output)

Prioridad: Especifica la prioridad para el emparejamiento. Los valores más altos son una prioridad más alta. No es importante para las entradas exactas.

buffer_id: El buffer_id especifica un buffer inequívocamente. Deja sin especificar para ninguno.

in_port: Si se utiliza un buffer_id, este es el puerto de entrada asociado.

match: Un objeto ofp_match. Por defecto, este coincide con todo, por lo que probablemente debería establecer algunos de sus campos (e.g dl_src, in_port)

Sending OpenFlow messages with POX

```
connection.send( ... ) #Mandar un mensaje OpenFlow al switch
```

Cuando se inicia una conexión con un Switch, un evento `ConnectionUp` se lanza.

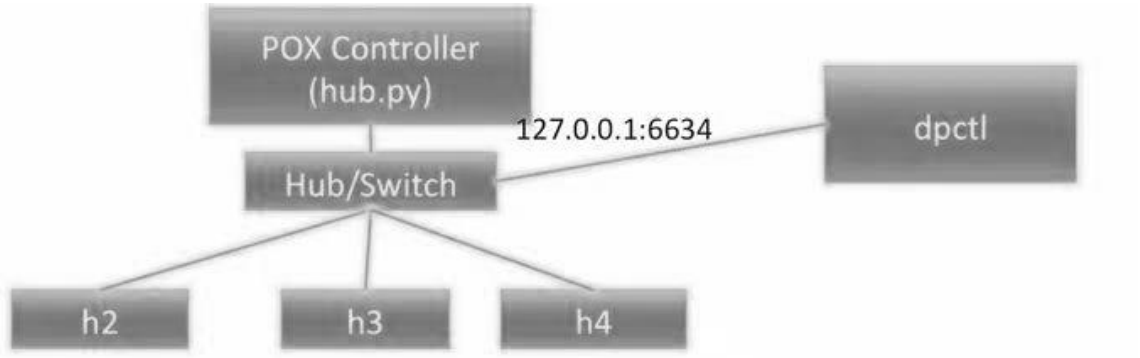
6 DESARROLLO DE LOS CONTROLADORES

6.1 forwarding.hub

El siguiente ejemplo configura una única regla de inundación, convirtiendo el OFS en un hub.

Cargamos una topología simple predeterminada

```
$ sudo mn –topo single,3 –mac –switch ovsk –controller remote
```



Topología simple 3 host, 1 hub

Mediante la herramienta net, nos dará un resumen del esquema de la topología

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0
c0
```

Hacemos ping mediante la utilidad "*pingall*" evidentemente los paquetes ICMP no llegarán al destino debido a que el switch no está configurado.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (0/6 received)
mininet>
```

Cargamos el controlador POX

```
./pox.py forwarding.hub
```

```
root@mininet-vm:~/pox# ./pox.py forwarding.hub
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:forwarding.hub:Hub running.
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
```

Y volvemos a hacer "*pingall*"

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet>
```

Mediante el comando "*dpctl dump-flows"* nos muestra todas las entradas de flujos

```
mininet@mininet-vm:~/pox$ dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x6614e981): flags=none type=1(flow)
  cookie=0, duration_sec=94s, duration_nsec=272000000s, table_id=0, priority=327
68, n_packets=0, n_bytes=0, idle_timeout=0,hard_timeout=0,actions=FLOOD
```

Como vemos, en la tabla *xid=0x6614e981* solo tiene una entrada, con una acción asignada, FLOOD, fluir por todos los puertos de salida menos por el que llegó.

Código en Python

```
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpidToStr

log = core.getLogger()


def _handle_ConnectionUp (event):
  msg = of.ofp_flow_mod()  #Crea un tipo de flujo

#Añade la acción, en este caso que fluya por todos los
#Puertos de salida menos por el que llegó el paquete.

  msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))

#Envia este mensaje OpenFlow al vSwitch

event.connection.send(msg)
  log.info("Hubifying %s", dpidToStr(event.dpid))

#La function launch() es automáticamente invocada, aquí es donde
se #registran todos los listeners y donde se crean los objetos
de la clase

def launch ():
#Crea un nuevo objeto "ConnectionUp" y ejecuta la función
  core.openflow.addListenerByName("ConnectionUp",
```

```
_handle_ConnectionUp)

  log.info("Hub running.")
```

6.2 Forwarding.l2_pairs

Este controlador transforma el OFS en un switch tipo L2. Sin embargo únicamente instala reglas basadas puramente en dirección Mac.

Cargamos el controlador POX

```
root@mininet-vm:~# cd pox
root@mininet-vm:~/pox# ./pox.py forwarding.l2_pairs
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:forwarding.l2_pairs:Pair-Learning switch running.
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
```

Mediante el comando "*dpctl dump-flows"* nos muestra todas las entradas de flujos

```
mininet@mininet-vm:~/pox/pox/forwarding$ dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0xffe5616d): flags=none type=1(flow)
```

Si nos fijamos, no tiene ninguna entrada de flujo debido a que todavía no ha habido tráfico fluyendo por la red.

Ejecutamos "*pingall*"

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet>
```

Y volvemos a mostrar todas las entradas de flujos

```
mininet@mininet-vm:~/pox/pox/forwarding$ dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x1f076fbb): flags=none type=1(flow)
  cookie=0, duration_sec=3s, duration_nsec=371000000s, table_id=0, priority=3276
8, n_packets=2, n_bytes=196, idle_timeout=0,hard_timeout=0,dl_src=00:00:00:00:00
:01,dl_dst=00:00:00:00:00:02,actions=output:2
  cookie=0, duration_sec=3s, duration_nsec=266000000s, table_id=0, priority=3276
8, n_packets=3, n_bytes=238, idle_timeout=0,hard_timeout=0,dl_src=00:00:00:00:00
:03,dl_dst=00:00:00:00:00:01,actions=output:1
  cookie=0, duration_sec=3s, duration_nsec=331000000s, table_id=0, priority=3276
8, n_packets=3, n_bytes=238, idle_timeout=0,hard_timeout=0,dl_src=00:00:00:00:00
:02,dl_dst=00:00:00:00:00:01,actions=output:1
  cookie=0, duration_sec=3s, duration_nsec=198000000s, table_id=0, priority=3276
8, n_packets=3, n_bytes=238, idle_timeout=0,hard_timeout=0,dl_src=00:00:00:00:00
:03,dl_dst=00:00:00:00:00:02,actions=output:2
  cookie=0, duration_sec=3s, duration_nsec=304000000s, table_id=0, priority=3276
8, n_packets=2, n_bytes=196, idle_timeout=0,hard_timeout=0,dl_src=00:00:00:00:00
:01,dl_dst=00:00:00:00:00:03,actions=output:3
  cookie=0, duration_sec=3s, duration_nsec=236000000s, table_id=0, priority=3276
8, n_packets=2, n_bytes=196, idle_timeout=0,hard_timeout=0,dl_src=00:00:00:00:00
:02,dl_dst=00:00:00:00:00:03,actions=output:3
mininet@mininet-vm:~/pox/pox/forwarding$
```

Ahora la tabla de flujo tendrá 6 entradas con su acción asociada correspondientes a cada par de direcciones.

El código en Python es el siguiente:

```
#Importa librerias
from pox.core import core

import pox.openflow.libopenflow_01 as of

#Usamos un log major que un print
log = core.getLogger()

#Creamos una table con un par (switch, MAC-addr) cada vez que entre un
#paquete
table = {}


def _handle_PacketIn (event):
  packet = event.parsed

  # Rellena la tabla
  table[(event.connection,packet.src)] = event.port

#Recoge el puerto de destino
  dst_port = table.get((event.connection,packet.dst))

#Si no sabemos cual es el destino enviamos el paquete por todos los puertos
#menos por el que entró

  if dst_port is None:
    msg = of.ofp_packet_out(data = event.ofp)
    msg.actions.append(of.ofp_action_output(port = of.OFPP_ALL))
    event.connection.send(msg)
  else:
#Al conocer los puertos de la MAC origen y destino podemos crear reglas #para
ambas direcciones
    msg = of.ofp_flow_mod()
    msg.match.dl_dst = packet.src
    msg.match.dl_src = packet.dst
    msg.actions.append(of.ofp_action_output(port = event.port))
    event.connection.send(msg)

#Tratamos el paquete que acaba de llegar, vamos a configurar la regla y
#reenviar el paquete
    msg = of.ofp_flow_mod()
    msg.data = event.ofp   #Reenviamos el paquete que ha llegado
    msg.match.dl_src = packet.src
    msg.match.dl_dst = packet.dst
    msg.actions.append(of.ofp_action_output(port = dst_port))
    event.connection.send(msg)
```

```
    log.debug("Installing %s <-> %s" % (packet.src, packet.dst))

#La función launch() es automáticamente invocada, aquí es donde se #registran
todos los listeners y donde se crean los objetos de la clase

def launch ():
#Crea un nuevo objeto “ConnectionUp” y ejecuta la función
  core.openflow.addListenerByName("PacketIn", _handle_PacketIn)

  log.info("Pair-Learning switch running.")
```

6.3.l3_learning.py (L3-learning-switchy-thing)

Este controlador no convierte al OFS del todo en un router, pero tampoco es un switch L2. Tal vez el aspecto más útil de todo es que sirve a modo de ejemplo del uso de la librería de POX para examinar y construir peticiones y respuestas ARP.
L3_learning.py realmente no se preocupa de cosas convencionales como subredes, solo aprende donde está la dirección IP, aunque los hosts ya se preocupen de esto.
Las utilidades más importantes de este controlador son:

1. Aprende la correspondencia entre direcciones IP y MAC.
2. Utiliza esta información para instalar la regla que reemplaza la MAC para que los hosts se puedan comunicar con distintas subredes.
3. Genera peticiones ARP en el caso si no se conoce la IP de destino.
4. Respuestas a solicitudes ARP.

Para parar la topología anterior y cerrar los procesos, en el CLI de MiniNet ejecutamos exit.
Cargamos el nuevo controlador con:

```
root@mininet-vm:~# cd pox
root@mininet-vm:~/pox# ./pox.py forwarding.l3_learning
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
```

Mediante el comando “*dpctl dump-flows”* nos muestra todas las entradas de flujos

```
mininet@mininet-vm:~/pox/pox/forwarding$ dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0xffe5616d): flags=none type=1(flow)
```

Si nos fijamos, no tiene ninguna entrada de flujo debido a que todavía no ha habido tráfico fluyendo por la red.

Ejecutamos “*pingall*”

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet>
```

Y volvemos a mostrar todas las entradas de flujos

```
root@mininet-vm:~# sudo dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x95a23b5a): flags=none type=1(flow)
  cookie=0, duration_sec=1s, duration_nsec=819000000s, table_id=0, priority=6553
5, n_packets=1, n_bytes=98, idle_timeout=10,hard_timeout=0,icmp,in_port=3,dl_vla
n=0xffff,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:02,nw_src=10.0.0.3,nw_ds
t=10.0.0.2,nw_tos=0x00,icmp_type=8,icmp_code=0,actions=mod_dl_dst:00:00:00:00:00
:02,output:2
  cookie=0, duration_sec=1s, duration_nsec=818000000s, table_id=0, priority=6553
5, n_packets=1, n_bytes=98, idle_timeout=10,hard_timeout=0,icmp,in_port=2,dl_vla
n=0xffff,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:03,nw_src=10.0.0.2,nw_ds
t=10.0.0.3,nw_tos=0x00,icmp_type=0,icmp_code=0,actions=mod_dl_dst:00:00:00:00:00
:03,output:3
  cookie=0, duration_sec=1s, duration_nsec=853000000s, table_id=0, priority=6553
5, n_packets=1, n_bytes=98, idle_timeout=10,hard_timeout=0,icmp,in_port=1,dl_vla
n=0xffff,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,nw_src=10.0.0.1,nw_ds
t=10.0.0.2,nw_tos=0x00,icmp_type=8,icmp_code=0,actions=mod_dl_dst:00:00:00:00:00
:02,output:2
  cookie=0, duration_sec=1s, duration_nsec=846000000s, table_id=0, priority=6553
5, n_packets=1, n_bytes=98, idle_timeout=10,hard_timeout=0,icmp,in_port=1,dl_vla
n=0xffff,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03,nw_src=10.0.0.1,nw_ds
t=10.0.0.3,nw_tos=0x00,icmp_type=8,icmp_code=0,actions=mod_dl_dst:00:00:00:00:00
:03,output:3
  cookie=0, duration_sec=1s, duration_nsec=844000000s, table_id=0, priority=6553
5, n_packets=1, n_bytes=98, idle_timeout=10,hard_timeout=0,icmp,in_port=3,dl_vla
n=0xffff,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01,nw_src=10.0.0.3,nw_ds
t=10.0.0.1,nw_tos=0x00,icmp_type=0,icmp_code=0,actions=mod_dl_dst:00:00:00:00:00
:01,output:1
  cookie=0, duration_sec=1s, duration_nsec=822000000s, table_id=0, priority=6553
5, n_packets=1, n_bytes=98, idle_timeout=10,hard_timeout=0,icmp,in_port=1,dl_vla
n=0xffff,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03,nw_src=10.0.0.1,nw_ds
t=10.0.0.3,nw_tos=0x00,icmp_type=0,icmp_code=0,actions=mod_dl_dst:00:00:00:00:00
:03,output:3
  cookie=0, duration_sec=1s, duration_nsec=828000000s, table_id=0, priority=6553
5, n_packets=1, n_bytes=98, idle_timeout=10,hard_timeout=0,icmp,in_port=3,dl_vla
n=0xffff,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:02,nw_src=10.0.0.3,nw_ds
t=10.0.0.2,nw_tos=0x00,icmp_type=0,icmp_code=0,actions=mod_dl_dst:00:00:00:00:00
:02,output:2
  cookie=0, duration_sec=1s, duration_nsec=830000000s, table_id=0, priority=6553
5, n_packets=1, n_bytes=98, idle_timeout=10,hard_timeout=0,icmp,in_port=2,dl_vla
n=0xffff,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:03,nw_src=10.0.0.2,nw_ds
t=10.0.0.3,nw_tos=0x00,icmp_type=8,icmp_code=0,actions=mod_dl_dst:00:00:00:00:00
:03,output:3
  cookie=0, duration_sec=1s, duration_nsec=823000000s, table_id=0, priority=6553
5, n_packets=1, n_bytes=98, idle_timeout=10,hard_timeout=0,icmp,in_port=3,dl_vla
n=0xffff,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01,nw_src=10.0.0.3,nw_ds
t=10.0.0.1,nw_tos=0x00,icmp_type=8,icmp_code=0,actions=mod_dl_dst:00:00:00:00:00
:01,output:1
  cookie=0, duration_sec=1s, duration_nsec=853000000s, table_id=0, priority=6553
5, n_packets=1, n_bytes=98, idle_timeout=10,hard_timeout=0,icmp,in_port=2,dl_vla
n=0xffff,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,nw_src=10.0.0.2,nw_ds
t=10.0.0.1,nw_tos=0x00,icmp_type=0,icmp_code=0,actions=mod_dl_dst:00:00:00:00:00
:01,output:1
  cookie=0, duration_sec=1s, duration_nsec=840000000s, table_id=0, priority=6553
5, n_packets=1, n_bytes=98, idle_timeout=10,hard_timeout=0,icmp,in_port=2,dl_vla
n=0xffff,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,nw_src=10.0.0.2,nw_ds
t=10.0.0.1,nw_tos=0x00,icmp_type=8,icmp_code=0,actions=mod_dl_dst:00:00:00:00:00
:01,output:1
  cookie=0, duration_sec=1s, duration_nsec=837000000s, table_id=0, priority=6553
5, n_packets=1, n_bytes=98, idle_timeout=10,hard_timeout=0,icmp,in_port=1,dl_vla
n=0xffff,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,nw_src=10.0.0.1,nw_ds
t=10.0.0.2,nw_tos=0x00,icmp_type=0,icmp_code=0,actions=mod_dl_dst:00:00:00:00:00
:02,output:2
```

Ahora tendremos una entrada para cada par de conexiones y para cada par de tipos de

mensaje ICMP, request y reply.

```
“””

Para cada Switch:

1) Mantener una tabla que asigna direcciones IP a direcciones MAC y puertos de switch.

  Se rellena la tabla utilizando la información de paquetes ARP e IP

2) Cuando vea una consulta ARP responde con la información de su tabla, si la entrada es vieja, pasa la consulta ARP.

3) Fluye su tabla ARP.

4) Cuando llega un paquete IP, si tiene una entrada con la ip destino puerto y mac, fluye el paquete por ella, sino, fluye por todas.

5) Borra una entrada si no llegan frames pertenecientes a esa entrada en un cierto tiempo.

6) El destino es multicast? Flood

7) Si el puerto de salida es el mismo puerto de entrada borra el paquete.


"""

#Importamos las librerías necesarias

from pox.core import core

import pox

log = core.getLogger()   #Creamos un log


from pox.lib.packet.ethernet import ethernet, ETHER_BROADCAST

from pox.lib.packet.ipv4 import ipv4

from pox.lib.packet.arp import arp

from pox.lib.addresses import IPAddr, EthAddr

from pox.lib.util import str_to_bool, dpidToStr

from pox.lib.recoco import Timer


import pox.openflow.libopenflow_01 as of
```

```
from pox.lib.revent import *

import time

# Timeout para los flujos
FLOW_IDLE_TIMEOUT = 10
# Timeout para las entradas de la tabla ARP
ARP_TIMEOUT = 60 * 2

# Maximo numero de paquetes en el buffer del switch para una IP desconocida
MAX_BUFFERED_PER_IP = 5

     #Maximo tiempo de espera en el buffer para una IP desconocida en
     segundos
MAX_BUFFER_TIME = 5



class Entry (object):
       """
     No es estrictamente una entrada ARP
     Usamos el puerto para determinar que puerto reenvia trafico fuera
     Usamos la MAC para responder  a peticiones ARP
     Usamos el timeout para que si una entrada es más vieja que ARP_TIMEOUT,
     generamos una petición ARP
       """
     #Funciones
     #Iniciar una peticion
  def __init__ (self, port, mac):
```

```
    self.timeout = time.time() + ARP_TIMEOUT
    self.port = port
    self.mac = mac
      #Son iguales dos peticiones?
  def __eq__ (self, other):
    if type(other) == tuple:
      return (self.port,self.mac)==other
    else:
      return (self.port,self.mac)==(other.port,other.mac)
      #Son diferentes dos peticiones?
  def __ne__ (self, other):
    return not self.__eq__(other)
      #Ha expirado?
  def isExpired (self):
    if self.port == of.OFPP_NONE: return False
    return time.time() > self.timeout


def dpid_to_mac (dpid):
  return EthAddr("%012x" % (dpid & 0xffffffffffff,))


class l3_switch (EventMixin):
  def __init__ (self, fakeways = [], arp_for_unknowns = False):
      #Respondemos a las peticiones ARP con las puertas falsas
    self.fakeways = set(fakeways)
     # buffer de IPs Entradas, una por Switch (identificado por dpid)
      # Si es verdadero y vemos un paquete con un host desconocido,
      #mandaremos un arp
    self.arp_for_unknowns = arp_for_unknowns
```

```
    # (dpid,IP) -> expire_time
    #Utilizamos esto para evitar hacer spam ARP
  self.outstanding_arps = {}

    # (dpid,IP) -> [(expire_time,buffer_id,in_port), ...]
    # Estos son los buffers que tenemos en este datapath para esta ip que
    no podemos entregar porque no conocemos el destino
  self.lost_buffers = {}

    # Para cada Switch, mapeamos las direcciones IP a la entrada
  self.arpTable = {}

    # Esto es el manejador de tiempo que se ocupa de expirar
  self._expire_timer = Timer(5, self._handle_expiration, recurring=True)

  self.listenTo(core)

def _handle_expiration (self):
    #Llamado por el timer para que podamos eliminar elementos antiguos.
  empty = []
  for k,v in self.lost_buffers.iteritems():
    dpid,ip = k

    for item in list(v):
      expires_at,buffer_id,in_port = item
      if expires_at < time.time():
    # Este paquete es antiguo, le comunicamos al interruptor que lo elimine
        v.remove(item)
        po = of.ofp_packet_out(buffer_id = buffer_id, in_port = in_port)
```

```
            core.openflow.sendToDPID(dpid, po)
        if len(v) == 0: empty.append(k)

      # Borramos los buffer vacíos
    for k in empty:
      del self.lost_buffers[k]

def _send_lost_buffers (self, dpid, ipaddr, macaddr, port):
    """
    Quizá eliminamos paquetes que no sabíamos donde iban y los cuales
    sabemos ahora.
    """
    if (dpid,ipaddr) in self.lost_buffers:
      bucket = self.lost_buffers[(dpid,ipaddr)]
      del self.lost_buffers[(dpid,ipaddr)]
      log.debug("Sending %i buffered packets to %s from %s"
                % (len(bucket),ipaddr,dpidToStr(dpid)))
      for _,buffer_id,in_port in bucket:
        po = of.ofp_packet_out(buffer_id=buffer_id,in_port=in_port)
        po.actions.append(of.ofp_action_dl_addr.set_dst(macaddr))
        po.actions.append(of.ofp_action_output(port = port))
        core.openflow.sendToDPID(dpid, po)

def _handle_GoingUpEvent (self, event):
    self.listenTo(core.openflow)
    log.debug("Up...")

def _handle_PacketIn (self, event):
    dpid = event.connection.dpid # identificacion del switch
```

```
    inport = event.port # identificacion del puerto
    packet = event.parsed # paquete recibido
    if not packet.parsed:
      log.warning("%i %i ignoring unparsed packet", dpid, inport)
      return

    if dpid not in self.arpTable:
     #Nuevo Switch, creamos una table vacía.
      self.arpTable[dpid] = {}
      for fake in self.fakeways:
        self.arpTable[dpid][IPAddr(fake)] = Entry(of.OFPP_NONE,
         dpid_to_mac(dpid))

    if packet.type == ethernet.LLDP_TYPE:
     # Ignoramos los paquetes LLDP
      return

    if isinstance(packet.next, ipv4):
      log.debug("%i %i IP %s => %s", dpid,inport,
                packet.next.srcip,packet.next.dstip)

     # Estos paquetes serán enviados más tarde, una vez que el controlador
     #aprende IPs aún desconocidas.
      self._send_lost_buffers(dpid, packet.next.srcip, packet.src, inport)

     # Aprendemos o actualizamos la table MAC/IP
      if packet.next.srcip in self.arpTable[dpid]:
        if self.arpTable[dpid][packet.next.srcip] != (inport, packet.src):
          log.info("%i %i RE-learned %s", dpid,inport,packet.next.srcip)
```

```
      else:
        log.debug("%i %i learned %s", dpid,inport,str(packet.next.srcip))
      self.arpTable[dpid][packet.next.srcip] = Entry(inport, packet.src)
     # El packet.next.srcip es una dirección IP de origen, mientras que
     packet.src es su dirección MAC de origen.
     # Intentamos reenviar.

      dstaddr = packet.next.dstip
      if dstaddr in self.arpTable[dpid]:
     # Tenemos información sobre el puerto de salida.

        prt = self.arpTable[dpid][dstaddr].port
        mac = self.arpTable[dpid][dstaddr].mac
        if prt == inport:
          log.warning("%i %i not sending packet for %s back out of the " +
"input port" % (dpid, inport, str(dstaddr)))
        else:
          log.debug("%i %i installing flow for %s => %s out port %i" %
(dpid, inport, packet.next.srcip, dstaddr, prt))

          actions = []
     # Acción responsable de reemplazar un destino MAC.
          actions.append(of.ofp_action_dl_addr.set_dst(mac))
          actions.append(of.ofp_action_output(port = prt))
          match = of.ofp_match.from_packet(packet, inport)
          match.dl_src = None # Wildcard Fuente Mac

          msg = of.ofp_flow_mod(command=of.OFPFC_ADD,
                                idle_timeout=FLOW_IDLE_TIMEOUT,
```

```
                                hard_timeout=of.OFP_FLOW_PERMANENT,
                                buffer_id=event.ofp.buffer_id,
                                actions=actions,
                                match=of.ofp_match.from_packet(packet,
                                                               inport))
        event.connection.send(msg.pack())
    elif self.arp_for_unknowns:
    # No conocemos el destino
    #En primer lugar tratamos de introducirlo en el buffer para más tarde
    # Cuando lo conozcamos enviaremos un ARP cuya respuesta debe serl
    # igual al que acabamos de aprender
    # Lo añadimos al buffer
      if (dpid,dstaddr) not in self.lost_buffers:
        self.lost_buffers[(dpid,dstaddr)] = []
      bucket = self.lost_buffers[(dpid,dstaddr)]
    #En el caso en el que el pu
      entry = (time.time() + MAX_BUFFER_TIME,event.ofp.buffer_id,inport)
      bucket.append(entry)
      while len(bucket) > MAX_BUFFERED_PER_IP: del bucket[0]

    # Expira excepcionalmente entradas de nuestra lista ARP
      self.outstanding_arps = {k:v for k,v in
       self.outstanding_arps.iteritems() if v > time.time()}

    # Check si ya hemos hecho un ARP recientemente
      if (dpid,dstaddr) in self.outstanding_arps:
    # Oop, lo hicimos recientemente.
        return
```

```
    # ARP
      self.outstanding_arps[(dpid,dstaddr)] = time.time() + 4

      r = arp()
      r.hwtype = r.HW_TYPE_ETHERNET
      r.prototype = r.PROTO_TYPE_IP
      r.hwlen = 6
      r.protolen = r.protolen
      r.opcode = r.REQUEST
      r.hwdst = ETHER_BROADCAST
      r.protodst = dstaddr
      r.hwsrc = packet.src
      r.protosrc = packet.next.srcip
      e = ethernet(type=ethernet.ARP_TYPE, src=packet.src,
                   dst=ETHER_BROADCAST)
      e.set_payload(r)
      log.debug("%i %i ARPing for %s on behalf of %s" % (dpid, inport,
       str(r.protodst), str(r.protosrc)))
      msg = of.ofp_packet_out()
      msg.data = e.pack()
      msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
      msg.in_port = inport
      event.connection.send(msg)

  elif isinstance(packet.next, arp):
    a = packet.next
    log.debug("%i %i ARP %s %s => %s", dpid, inport,
     {arp.REQUEST:"request",arp.REPLY:"reply"}.get(a.opcode,
```

```
  'op:%i' % (a.opcode,)), str(a.protosrc), str(a.protodst))

 if a.prototype == arp.PROTO_TYPE_IP:
   if a.hwtype == arp.HW_TYPE_ETHERNET:
     if a.protosrc != 0:

# Aprendemos o actualizamos port/MAC info
       if a.protosrc in self.arpTable[dpid]:
         if self.arpTable[dpid][a.protosrc] != (inport, packet.src):
           log.info("%i %i RE-learned %s", dpid,inport,str(a.protosrc))
       else:
         log.debug("%i %i learned %s", dpid,inport,str(a.protosrc))
       self.arpTable[dpid][a.protosrc] = Entry(inport, packet.src)

# Enviamos cualquier paquete en espera
       self._send_lost_buffers(dpid, a.protosrc, packet.src, inport)

       if a.opcode == arp.REQUEST:
# Quizá podemos responder

         if a.protodst in self.arpTable[dpid]:
# Tenemos la respuesta

           if not self.arpTable[dpid][a.protodst].isExpired():
#y relativamente actualizada, podemos responder.

             r = arp()
             r.hwtype = a.hwtype
```

```
                    r.prototype = a.prototype
                    r.hwlen = a.hwlen
                    r.protolen = a.protolen
                    r.opcode = arp.REPLY
                    r.hwdst = a.hwsrc
                    r.protodst = a.protosrc
                    r.protosrc = a.protodst
                    r.hwsrc = self.arpTable[dpid][a.protodst].mac
                    e = ethernet(type=packet.type, src=dpid_to_mac(dpid),
dst=a.hwsrc)
                    e.set_payload(r)
                    log.debug("%i %i answering ARP for %s" % (dpid, inport,
                     str(r.protosrc)))
                    msg = of.ofp_packet_out()
                    msg.data = e.pack()
                    msg.actions.append(of.ofp_action_output(port =
                                                            of.OFPP_IN_PORT))
                    msg.in_port = inport
                    event.connection.send(msg)
                    return

      # No sabemos como tratar este ARP, inundamos
       log.debug("%i %i flooding ARP %s %s => %s" % (dpid, inport,
        {arp.REQUEST:"request",arp.REPLY:"reply"}.get(a.opcode,
        'op:%i' % (a.opcode,)), str(a.protosrc), str(a.protodst)))

       msg = of.ofp_packet_out(in_port = inport, action =
of.ofp_action_output(port = of.OFPP_FLOOD))
       if event.ofp.buffer_id is of.NO_BUFFER:
```

```
      # Intentamos enviar el paquete incomplete en bruto.
        msg.data = event.data
      else:
        msg.buffer_id = event.ofp.buffer_id
      event.connection.send(msg.pack())



def launch (fakeways="", arp_for_unknowns=None):
  fakeways = fakeways.replace(","," ").split()
  fakeways = [IPAddr(x) for x in fakeways]
  if arp_for_unknowns is None:
    arp_for_unknowns = len(fakeways) > 0
  else:
    arp_for_unknowns = str_to_bool(arp_for_unknowns)
  core.registerNew(l3_switch, fakeways, arp_for_unknowns)
```

6.4 forwarding.l2_multi

Este componente puede ser visto como un switch de aprendizaje, pero tiene una diferencia, no recoge información únicamente local, l2_multi utiliza openflow.discovery para aprender de la topología de toda la red. Funciona con openflow.spanning_tree.

Con esto consigue ser una aplicación de camino más corto.

```
"""

Es una aplicacion para encontrar la ruta más corta, este conmutador es
nivel 2 y aprende direcciones Ethernet en toda la red para tomar
caminos cortos entre ellos.

Depende de OpenFlow.discovery y funciona con openflow.spanning_tree

"""


from pox.core import core

import pox.openflow.libopenflow_01 as of

from pox.lib.revent import *

from pox.lib.recoco import Timer

from collections import defaultdict

from pox.openflow.discovery import Discovery

from pox.lib.util import dpid_to_str

import time


log = core.getLogger()


# Matiz de adyacencia, si dos switches están .  [sw1][sw2] -> port from
# sw1 to sw2

adjacency = defaultdict(lambda:defaultdict(lambda:None))


# Switches que conocemos.  [dpid] -> Switch

switches = {}


# ethaddr -> (switch, port) | Su dirección mac de origen se utiliza
```

```
# como una clave, además se guarda el puerto origen.
mac_map = {}


# [sw1][sw2] -> (distance, intermediate)
# Mapa de camino, las posiciones son los switches y contiene un vector
# (distancia, intermedio)
path_map = defaultdict(lambda:defaultdict(lambda:(None,None)))


“””
Waiting path.  (dpid,xid)->WaitingPath
Lista de respuestas Barrier identificadas mediante un XID, cuando la lista esté vacía será porque todos los switches que conforman el camino están listos para recibir el paquete y encaminarlo por el camino más corto.
“””
waiting_paths = {}


# tiempo en el que no fluyen en segundos
FLOOD_HOLDDOWN = 5


# Flow timeouts
FLOW_IDLE_TIMEOUT = 10
FLOW_HARD_TIMEOUT = 30


# Cuanto tiempo disponemos para establecer un camino?
PATH_SETUP_TIME = 4



def _calc_paths ():
  """
```

```
El algoritmo de Floyd-Marshal compara todos los posibles caminos a
través del grafo entre cada par de vértices
  """

#Funcion para imprimir el path_map
  def dump ():
    for i in sws:
      for j in sws:
        a = path_map[i][j][0]
        if a is None: a = "*"
        print a,
        print
  sws = switches.values()
  path_map.clear()
  for k in sws:
    for j,port in adjacency[k].iteritems():
      if port is None: continue
      path_map[k][j] = (1,None)
    path_map[k][k] = (0,None) # distance, intermediate


  #dump()


  for k in sws:
    for i in sws:
      for j in sws:
        if path_map[i][k][0] is not None:
          if path_map[k][j][0] is not None:
            # i -> k -> j existe
            ikj_dist = path_map[i][k][0]+path_map[k][j][0]
        if path_map[i][j][0] is None or ikj_dist < path_map[i][j][0]:
```

```
            # i -> k -> j es mejor que el anterior.
            path_map[i][j] = (ikj_dist, k)


  #print "--------------------"
  #dump()



def _get_raw_path (src, dst):
  """
  Da el camino en bruto, la lista de nodos que atraviesa el paquete.
"""
  if len(path_map) == 0: _calc_paths()
  if src is dst:
    # Esta aqui!
    return []
  if path_map[src][dst][0] is None:
    return None
  intermediate = path_map[src][dst][1]
  if intermediate is None:
    # Directamente conectado
    return []
  return _get_raw_path(src, intermediate) + [intermediate] + \
         _get_raw_path(intermediate, dst)



def _check_path (p):
  """
  Se asegura que el camino es una serie de nodos con puertos conectados
entre sí, da true si el camino es correcto-
  """
```

```
  for a,b in zip(p[:-1],p[1:]):
    if adjacency[a[0]][b[0]] != a[2]:
      return False
    if adjacency[b[0]][a[0]] != b[2]:
      return False
  return True



def _get_path (src, dst, first_port, final_port):
  """
  Devuelve el destino más corto mediante el algoritmo de Floyd-Warshall
-- a list of (node,in_port,out_port) -- Para después con esta
información se encuentran los puertos de salida.
  """
  # Comienza con la lista de nodos que atraviesa el paquete...
  if src == dst:
    path = [src]
  else:
    path = _get_raw_path(src, dst)
    if path is None: return None
    path = [src] + path + [dst]


  #Añadimos los puertos de salida
  r = []
  in_port = first_port
  for s1,s2 in zip(path[:-1],path[1:]):
    out_port = adjacency[s1][s2]
    r.append((s1,in_port,out_port))
    in_port = adjacency[s2][s1]
  r.append((dst,in_port,final_port))
```

```
#Comprobamos que el camino existe.
  assert _check_path(r), "Illegal path!"

  return r



class WaitingPath (object):
  """
  Es la caché de paquetes que están esperando para establecerse en su
camino.
"""
  def __init__ (self, path, packet):
    """
   Xids es una secuencia de (dpid, XID)
   first_switch es el DPID de donde llegó el paquete
    """
# Configura el tiempo de expiración del paquete
    self.expires_at = time.time() + PATH_SETUP_TIME
    self.path = path
    self.first_switch = path[0][0].dpid # Primer Switch
    self.xids = set()
    self.packet = packet

    #Si la cola es de más de Mil, se eliminan caminos…
    if len(waiting_paths) > 1000:
      WaitingPath.expire_waiting_paths()

  def add_xid (self, dpid, xid):
```

```
    self.xids.add((dpid,xid))
    waiting_paths[(dpid,xid)] = self

  @property
  #Devuelve si el evento ha expirado ya.
  def is_expired (self):
    return time.time() >= self.expires_at

  def notify (self, event):
    """
    Cuando llega una respuesta Barrier es notificada
    """
    self.xids.discard((event.dpid,event.xid))
    if len(self.xids) == 0:
      # Hecho!
      if self.packet:
        log.debug("Sending delayed packet out %s"
                  % (dpid_to_str(self.first_switch),))
        msg = of.ofp_packet_out(data=self.packet,
            action=of.ofp_action_output(port=of.OFPP_TABLE))
        core.openflow.sendToDPID(self.first_switch, msg)

      core.l2_multi.raiseEvent(PathInstalled(self.path))


  @staticmethod
  #Método estático, para eliminar los caminos que han expirado
  def expire_waiting_paths ():
    packets = set(waiting_paths.values())
```

```
    killed = 0 #Contador de paquetes que se han tirado.
    for p in packets:
      if p.is_expired:
        killed += 1
        for entry in p.xids:
          waiting_paths.pop(entry, None)
    if killed:
      log.error("%i paths failed to install" % (killed,))



class PathInstalled (Event):
  """
Es un evento que se dispara una vez que las normas se instalan para un
destino MAC particular
  """
  def __init__ (self, path):
    Event.__init__(self)
    self.path = path


#Representa un switch físico y contiene la lógica para manejar eventos
# PacketIn y la instalación de reglas de reenvío
class Switch (EventMixin):
  def __init__ (self):
    self.connection = None
    self.ports = None
    self.dpid = None
    self._listeners = None
    self._connected_at = None

  def __repr__ (self):
```

```
    return dpid_to_str(self.dpid)
#En primer lugar WaitingPath es creado para que nuestro paquete pueda
#ser enviado después de que recibamos Barrier reply. Entonces se crea
#una regla para cada switch a lo largo del camino más corto acompañado
#de un Barrier request.
#Las reglas son básicamente el reenvio que tenemos que seguir

  def _install (self, switch, in_port, out_port, match, buf = None):
    msg = of.ofp_flow_mod()
    msg.match = match
    msg.match.in_port = in_port
    msg.idle_timeout = FLOW_IDLE_TIMEOUT
    msg.hard_timeout = FLOW_HARD_TIMEOUT
    msg.actions.append(of.ofp_action_output(port = out_port))
    msg.buffer_id = buf
    switch.connection.send(msg)
#Tan pronto como el camino es calculado se convierte en las reglas
#apropiadas para instalarse en todos los Switches que pertenecen al
#camino.
  def _install_path (self, p, match, packet_in=None):
    wp = WaitingPath(p, packet_in)
    for sw,in_port,out_port in p:
      self._install(sw, in_port, out_port, match)
      msg = of.ofp_barrier_request()
      sw.connection.send(msg)
      wp.add_xid(sw.dpid,msg.xid)

  def install_path (self, dst_sw, last_port, match, event):
    """
```

```
  Instala la regla para que el paquete llegue a su destino siguiendo el
camino más corto.
    """
    p = _get_path(self, dst_sw, event.port, last_port)
    if p is None:
      log.warning("Can't get from %s to %s", match.dl_src,
match.dl_dst)


      import pox.lib.packet as pkt


      if (match.dl_type == pkt.ethernet.IP_TYPE and
          event.parsed.find('ipv4')):
        #  Destino inalcanzable
        log.debug("Dest unreachable (%s -> %s)",
                  match.dl_src, match.dl_dst)


        from pox.lib.addresses import EthAddr
        e = pkt.ethernet()
        e.src = EthAddr(dpid_to_str(self.dpid)) #FIXME: Hmm...
        e.dst = match.dl_src
        e.type = e.IP_TYPE
        ipp = pkt.ipv4()
        ipp.protocol = ipp.ICMP_PROTOCOL
        ipp.srcip = match.nw_dst #FIXME: Ridiculous
        ipp.dstip = match.nw_src
        icmp = pkt.icmp()
        icmp.type = pkt.ICMP.TYPE_DEST_UNREACH
        icmp.code = pkt.ICMP.CODE_UNREACH_HOST
        orig_ip = event.parsed.find('ipv4')
```

```
        d = orig_ip.pack()
        d = d[:orig_ip.hl * 4 + 8]
        import struct
        d = struct.pack("!HH", 0,0) + d #FIXME: MTU
        icmp.payload = d
        ipp.payload = icmp
        e.payload = ipp
        msg = of.ofp_packet_out()
        msg.actions.append(of.ofp_action_output(port = event.port))
        msg.data = e.pack()
        self.connection.send(msg)

      return

    log.debug("Installing path for %s -> %s %04x (%i hops)",
        match.dl_src, match.dl_dst, match.dl_type, len(p))

    # Al tener el camino, lo podemos instalar.
    self._install_path(p, match, event.ofp)

    p = [(sw,out_port,in_port) for sw,in_port,out_port in p]
    self._install_path(p, match.flip())



      """
Manejador para cuando llega un paquete
      """
def _handle_PacketIn (self, event):
    def flood ():
```

```
     #Envia el paquete por todos sus puertos menos por el de entrada
      if self.is_holding_down:
        log.warning("Not flooding -- holddown active")
      msg = of.ofp_packet_out()
      msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
      msg.buffer_id = event.ofp.buffer_id
      msg.in_port = event.port
      self.connection.send(msg)

    def drop ():
      # Purga el buffer
      if event.ofp.buffer_id is not None:
        msg = of.ofp_packet_out()
        msg.buffer_id = event.ofp.buffer_id
        event.ofp.buffer_id = None # Mark is dead
        msg.in_port = event.port
        self.connection.send(msg)

    packet = event.parsed
# localización de esta direccion MAC
    loc = (self, event.port)
# Último lugar de esta dirección MAC
    oldloc = mac_map.get(packet.src)
    #Si el paquete es tipo LLDP se purga
    if packet.effective_ethertype == packet.LLDP_TYPE:
      drop()
      return

    if oldloc is None:
```

```
    # Si el paquete no es Multicast se aprende la MAC
      if packet.src.is_multicast == False:
        mac_map[packet.src] = loc # Learn position for ethaddr
        log.debug("Learned %s at %s.%i", packet.src, loc[0], loc[1])
    elif oldloc != loc:
      #La localizacion no es la que nosotros sabiamos!
      if loc[1] not in adjacency[loc[0]].values():
        # El nuevo lugar es otro puerto probablemente.
        log.debug("%s moved from %s.%i to %s.%i?", packet.src,
                  dpid_to_str(oldloc[0].connection.dpid), oldloc[1],
                  dpid_to_str(   loc[0].connection.dpid),    loc[1])
        if packet.src.is_multicast == False:
          mac_map[packet.src] = loc # Aprendemos esta ethaddr
          log.debug("Learned %s at %s.%i", packet.src, loc[0], loc[1])
      elif packet.dst.is_multicast == False:
        log.warning("Packet from %s arrived at %s.%i without flow",
                    packet.src, dpid_to_str(self.dpid), event.port)
        #drop()
        #return



    if packet.dst.is_multicast:
      log.debug("Flood multicast from %s", packet.src)
      flood()
    else:
      if packet.dst not in mac_map:
        log.debug("%s unknown -- flooding" % (packet.dst,))
        flood()
      else:
```

```
"""
En caso de que ya contenta la ubicación destino, el camino más corto se
puede instalar, lo que significa que todos los switches desde el origen
y el destino están programados adecuadamente para que el paquete siga
por el camino más corto.
"""
        dest = mac_map[packet.dst]
        match = of.ofp_match.from_packet(packet)
        self.install_path(dest[0], dest[1], match, event)

  def disconnect (self):
    if self.connection is not None:
      log.debug("Disconnect %s" % (self.connection,))
      self.connection.removeListeners(self._listeners)
      self.connection = None
      self._listeners = None

  def connect (self, connection):
    if self.dpid is None:
      self.dpid = connection.dpid
    assert self.dpid == connection.dpid
    if self.ports is None:
      self.ports = connection.features.ports
    self.disconnect()
    log.debug("Connect %s" % (connection,))
    self.connection = connection
    self._listeners = self.listenTo(connection)
    self._connected_at = time.time()

  @property
```

```
#Devuelve false si lleva más tiempo  el paquete de lo permitido.
  def is_holding_down (self):
    if self._connected_at is None: return True
    if time.time() - self._connected_at > FLOOD_HOLDDOWN:
      return False
    return True

  def _handle_ConnectionDown (self, event):
    self.disconnect()
#La principal entidad responsable de los eventos procedentes de la
#bilioteca OpenFlow, es decir, connectionUp, BarrierIn y discovery.

class l2_multi (EventMixin):

  _eventMixin_events = set([PathInstalled,])

  def __init__ (self):
    # listado de dependencias.
    def startup ():
      core.openflow.addListeners(self, priority=0)
      core.openflow_discovery.addListeners(self)
    core.call_when_ready(startup, ('openflow','openflow_discovery'))

  def _handle_LinkEvent (self, event):
    def flip (link):
      return Discovery.Link(link[2],link[3], link[0],link[1])

    l = event.link
    sw1 = switches[l.dpid1]
```

```
    sw2 = switches[l.dpid2]

    # Para cada link que se añade, se asegura si hay un nuevo Path que
    #sea mejorado debido a este
    clear = of.ofp_flow_mod(command=of.OFPFC_DELETE)
    for sw in switches.itervalues():
      if sw.connection is None: continue
      sw.connection.send(clear)
    path_map.clear()

    if event.removed:
      #El link no nos sirve.
      if sw2 in adjacency[sw1]: del adjacency[sw1][sw2]
      if sw1 in adjacency[sw2]: del adjacency[sw2][sw1]

      #Pero puede ser que haya otra manera de comunicarse ambos
      #switches
     for ll in core.openflow_discovery.adjacency:
        if ll.dpid1 == l.dpid1 and ll.dpid2 == l.dpid2:
          if flip(ll) in core.openflow_discovery.adjacency:
            #  Enlace encontrado y elegido para conectarlos
            adjacency[sw1][sw2] = ll.port1
            adjacency[sw2][sw1] = ll.port2
            break
    else:
      # Si ya tenemos estos nodos conectados podemos ignorar este
      #enlace, aunque podríamos estar interesados.
      if adjacency[sw1][sw2] is None:
        # Estos no  estaban conectados previamente
```

```
      if flip(l) in core.openflow_discovery.adjacency:
        #  si existe el enlace en ambas direcciones los consideramos
        #conectados.!
        adjacency[sw1][sw2] = l.port1
        adjacency[sw2][sw1] = l.port2

    #Si aprendemos una MAC por un puerto que ya sabiamos, lo
    #desaprendemos.
    bad_macs = set()
    for mac,(sw,port) in mac_map.iteritems():
      if sw is sw1 and port == l.port1:
        if mac not in bad_macs:
          log.debug("Unlearned %s", mac)
          bad_macs.add(mac)
      if sw is sw2 and port == l.port2:
        if mac not in bad_macs:
          log.debug("Unlearned %s", mac)
          bad_macs.add(mac)
    for mac in bad_macs:
      del mac_map[mac]

def _handle_ConnectionUp (self, event):
  sw = switches.get(event.dpid)
  if sw is None:
    #Nuevo Switch
    sw = Switch()
    switches[event.dpid] = sw
    sw.connect(event.connection)
  else:
```

```
      sw.connect(event.connection)


  def _handle_BarrierIn (self, event):
"""
WaitingPath utiliza una lista de identificadores de peticiones Barrier,
llamados XIDs, para esperar a que todos los nodos responda, Cada vez
que una respuesta Barrier llega de un switch que conforma el camino, el
XID de la lista waitingPath es borrado de la lista, Cuando la lista
está vacía, es porque la regla está instalada en todos los switches y
es seguro enviar el paquete que estaba en caché.
"""
    wp = waiting_paths.pop((event.dpid,event.xid), None)
    if not wp:
      #log.info("No waiting packet %s,%s", event.dpid, event.xid)
      return
        #Tan pronto como se recibe la respuesta de BarrierIn, se
        #notifica
    wp.notify(event)



def launch ():
  core.registerNew(l2_multi)


  timeout = min(max(PATH_SETUP_TIME, 5) * 2, 15)
  Timer(timeout, WaitingPath.expire_waiting_paths, recurring=True)
```

6.5 Modificacion control de acceso

A modo de ejemplo vamos a añadir una característica al componente ***l3_learning***.
Esta característica de seguridad no es más que un control de acceso de los hosts en base a sus direcciones MACs.

Se creará una White list de MACs en un fichero llamado mac.txt, provocando que sólo estas MACs tienen permitido el acceso a la red.
Esta característica de seguridad es funcional principalmente en redes WiFi, donde se requiere tener un control sobre los hosts conectados a ella. En las redes cableadas este problema no es importante ya que se necesita el propio acceso al switch y ello aporta la suficiente seguridad.

La principal ventaja en esta característica de seguridad, es que el archivo Whitelist está centralizado, y no tenemos que entrar al CLI de los cientos de puntos de acceso que pueda tener una red medianamente grande para añadir una a una las MACs autorizadas.

Como vimos anteriormente cuando explicamos este controlador, su función es procesar paquetes que no encuentran afinidad en ninguna de sus flow tables, para ello de manera reactiva el switch deriva el paquete al controlador para que este procese y decida si este creará una nueva entrada de flujo en la tabla del switch.

Aquí es donde entra nuestra modificación, hemos añadido un par de líneas de código y nos bastará para que el controlador decida o no añadir una nueva entrada de flujo.

Esta decisión la toma en su manejador de eventos PacketIn, y no es más que un if comprobando si la mac destino del paquete está o no en un atributo que después explicaremos, si está se crea un nuevo flujo en el switch, si no está únicamente en el debug se guarda un report.

```
        if mac in self.whitelist_mac:
          log.debug("%i %i installing flow for %s => %s out port %i"
                    % (dpid, inport, packet.next.srcip, dstaddr, prt))

          actions = []
          actions.append(of.ofp_action_dl_addr.set_dst(mac))
          actions.append(of.ofp_action_output(port = prt))
          match = of.ofp_match.from_packet(packet, inport)
          match.dl_src = None # Wildcard source MAC

          msg = of.ofp_flow_mod(command=of.OFPFC_ADD,
                                idle_timeout=FLOW_IDLE_TIMEOUT,
                                hard_timeout=of.OFP_FLOW_PERMANENT,
                                buffer_id=event.ofp.buffer_id,
                                actions=actions,
                                match=of.ofp_match.from_packet(packet,
```

```
                                                                inport))
            event.connection.send(msg.pack())
        elif mac not in self.whitelist_mac:
            log.debug("MAC no autorizada, contacte con el administrador de
la red")
```

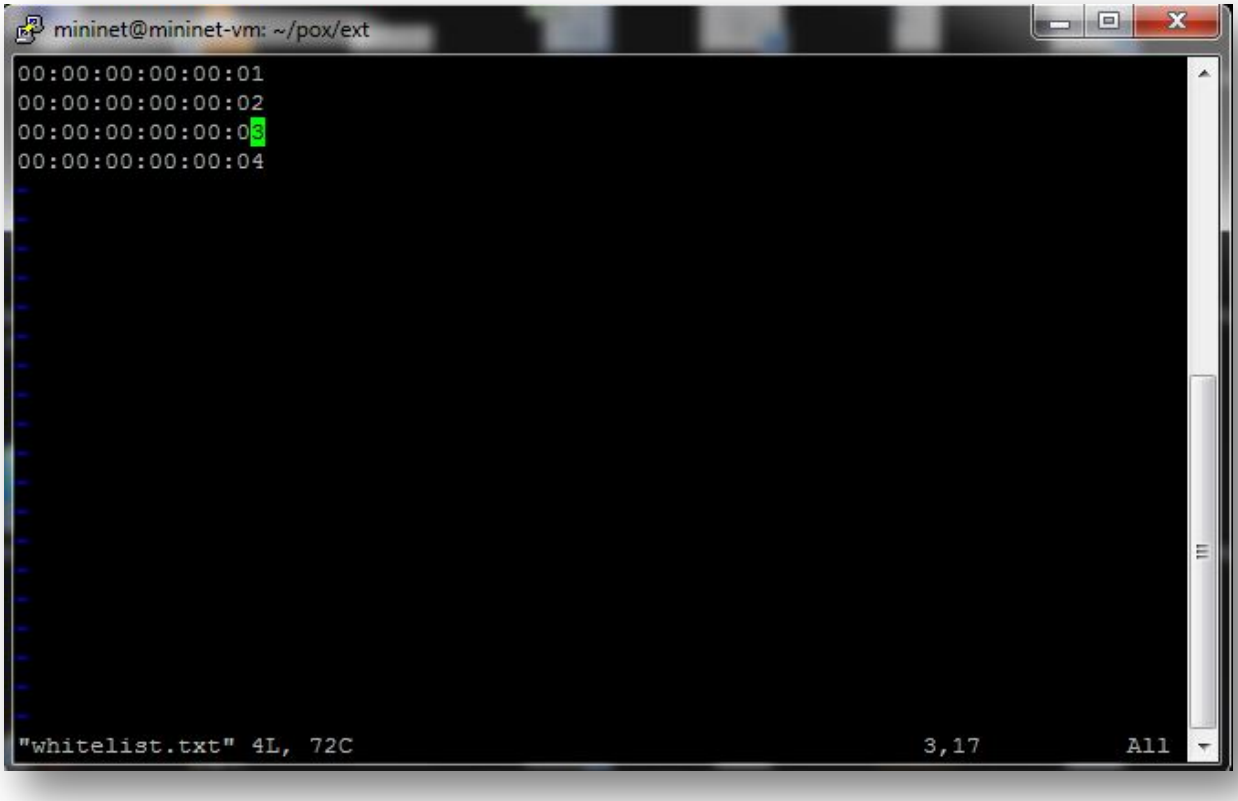
El atributo whitelist_mac no es más que un array el cual mediante los métodos de apertura y lectura de ficheros de Python se rellena en cada posición con cada MAC autorizada que pueden reenviar por la red.

```
self.whitelist_mac=[]
    infile=open('ext/whitelist.txt','r')
    for line in infile:
        mac=EthAddr (line)
        self.whitelist_mac.append(mac)
    infile.close()
```

6.5.1 Comprobando su funcionalidad

Para comprobar que la modificación a l3_learning para añadirle la característica de seguridad mediante una whitelist vamos a proceder a hacer una serie de pruebas.

Primero añadiremos en su archivo ubicado en ext una serie de líneas con cada MAC autorizada a navegar por la red



Después creamos una topología básica usando el comando mn:

```
$sudo mn –topo single,5 –mac –switch ovsk –controller remote
```

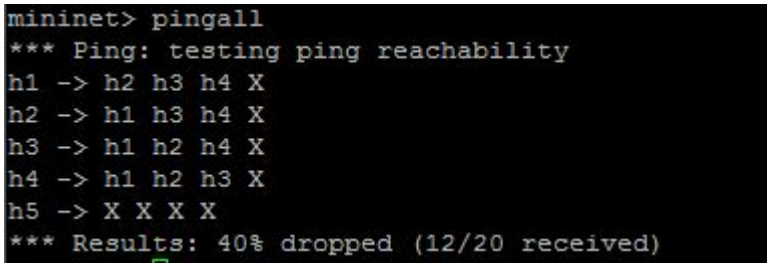
Esto creará una topología simple de 5 hosts, con 5 mac de 00:00:00:00:00:01 hasta 00:00:00:00:00:02 en orden ya que así se lo especificamos mediante el parámetro –mac al comando mn

```
mininet@mininet-vm:~$ sudo mn --topo single,5 --mac --switch ovsk --controller r
emote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4 h5
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1) (h5, s1)
*** Configuring hosts
h1 h2 h3 h4 h5
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

Una vez ya estamos en el CLI de MiniNet con la topología ya iniciada cargamos el controlador.
En este caso yo he hecho la modificación sobre el mismo archivo pox/pox/forwarding/l3_learning, lo correcto sería crear un nuevo controlador ya que añade una nueva característica.

Una vez el OVS funcionando con el controlador, veremos mediante la herramienta pingall como todos los hosts tendrán ping menos el h5, ya que su mac no está en la

whitelist.



Realmente el mensaje ICMP request si sale de h5 pero no llega el ICMP reply, ya que el switch lo que capa es el mensaje que va destinado a h5.
Esto quiere decir, que si intentamos cargar una página web desde h5, al servidor donde tiene alojada la web si que recibirá un http get, pero al devolver la web, el switch desechará este mensaje y h5 nunca podrá cargarla en su navegador.

Si nos damos cuenta, no hay límite a la hora de desarrollar componentes, los cuales proporcionan un punto de partida para futuros desarrollos.
No hay límite, el objetivo es poder crear componentes que hagan que la red se comporte como desee, siempre abstrayéndola como un todo, centralizada, sin estar sujeta a los estrictos RFC ni al firmware propietario de los fabricantes de equipos.

7. Bibliografía y referencias

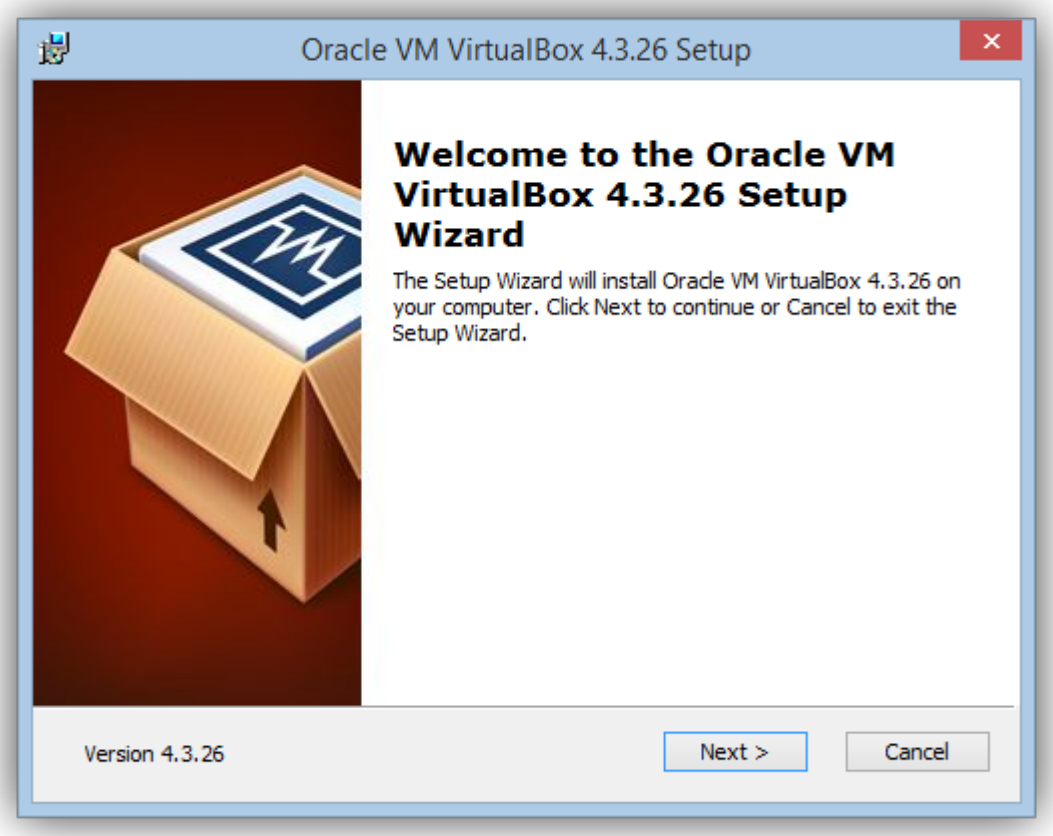
[1]http://myslide.es/download/link/openflow-protocolo-del-futuro
[2]http://repositorio.unican.es/xmlui/bitstream/handle/10902/1165/Sergio%20Rodriguez%20Santamaria.pdf?sequence=1
[3]http://bibdigital.epn.edu.ec/bitstream/15000/7360/1/CD-5509.pdf
[4]https://www.fayerwayer.com/2012/04/google-revela-su-plan-para-mejorar-las-redes-del-mundo/
[5]http://revistatelematica.cujae.edu.cu
[6]http://searchdatacenter.techtarget.com/es/respuesta/Como-ayuda-el-protocolo-OpenFlow-al-trafico-de-red-del-centro-de-datos
[7]http://repositorio.uta.edu.ec/bitstream/123456789/10587/1/Tesis_982ec.pdf
[8]http://ranosgrant.cocolog-nifty.com/openflow/dpctl.8.html
[9]http://mininet.org/walkthrough/
[10]https://es.scribd.com/doc/211359097/TFM-josebastida-1-2
[11]http://archive.openflow.org/
[12]https://www.opennetworking.org/
[13]http://www.dit.upm.es/~posgrado/doc/TFM/TFMs2011-2012/TFM_Vanessa_Moreno_2012.pdf
[14]http://sdnhub.org/tutorials/pox/
[15]https://github.com
[16]https://openflow.stanford.edu

8. ANEXO

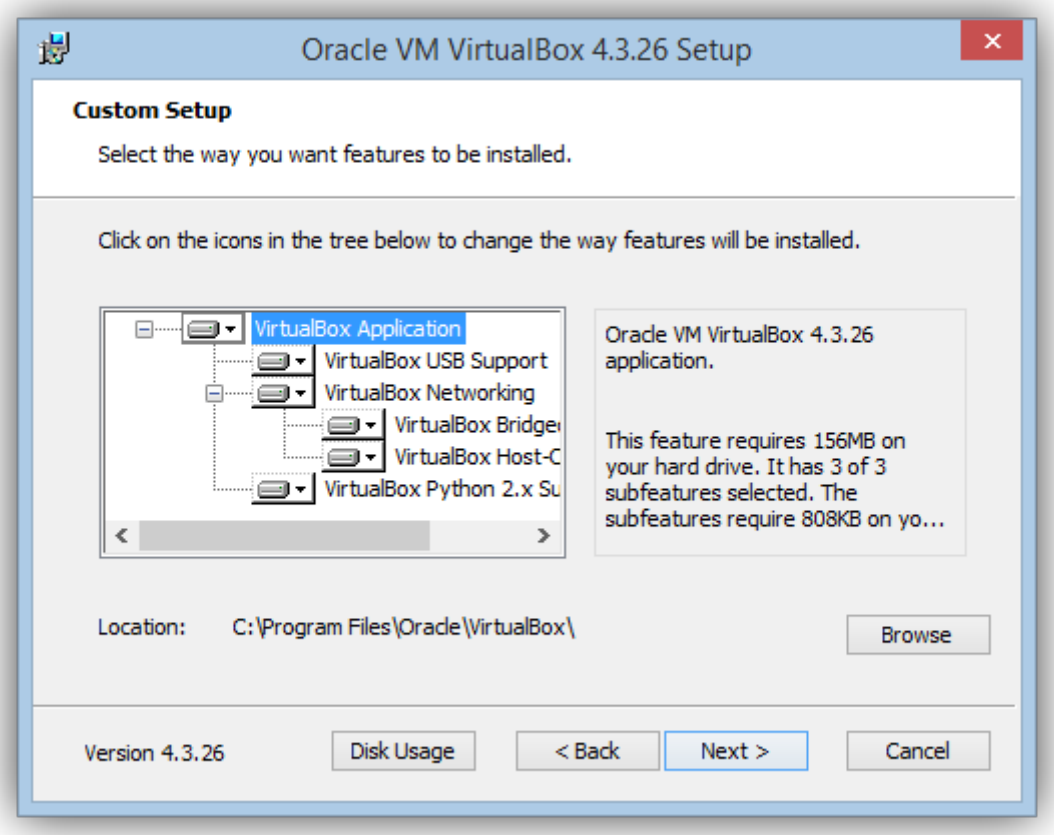
8.1 Instalación y configuración de las herramientas necesarias.

Descargar el paquete VirtualBox para tu plataforma desde su página oficial aceptando los términos y condiciones de su respectiva licencia (https://www.virtualbox.org/wiki/Downloads)

Ejecutaremos el lanzador del programa y se nos abrirá la primera pantalla tal y como aparece a continuación.

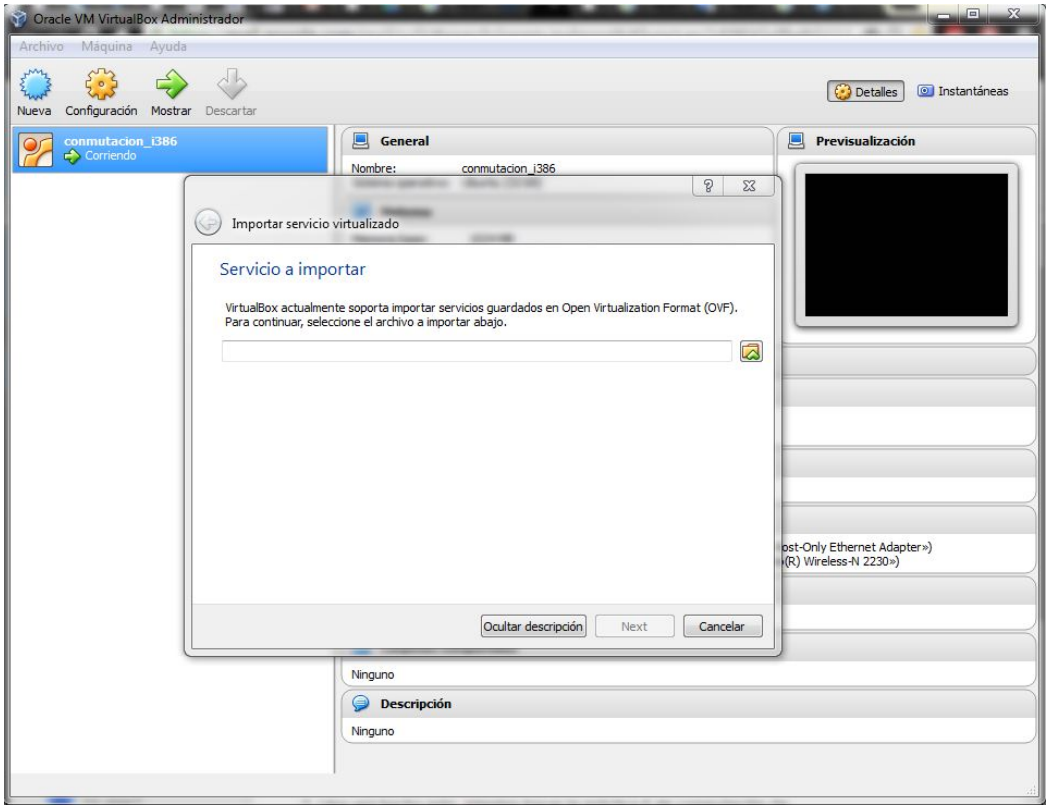


Nos da la bienvenida y nos informa de que si continuamos se instalara virtual box en nuestro sistema, pulsamos en “next” y en la siguiente pantalla deberemos elegir los componentes y la ruta donde lo instaremos.



Una vez instalado Oracle VM VirtualBox procedemos a descargar una imagen virtual de VirtualBox de un ubuntu con todos los paquetes necesarios instalados (MiniNet, ovswitch, POX) (2GB)
http://labit201.upct.es/VIOVyl9QYcS0bmF5DGwaeNVQG9/

Tras esto, lo primero es importarla dejando todos los parámetros de asignación de recursos por defecto.

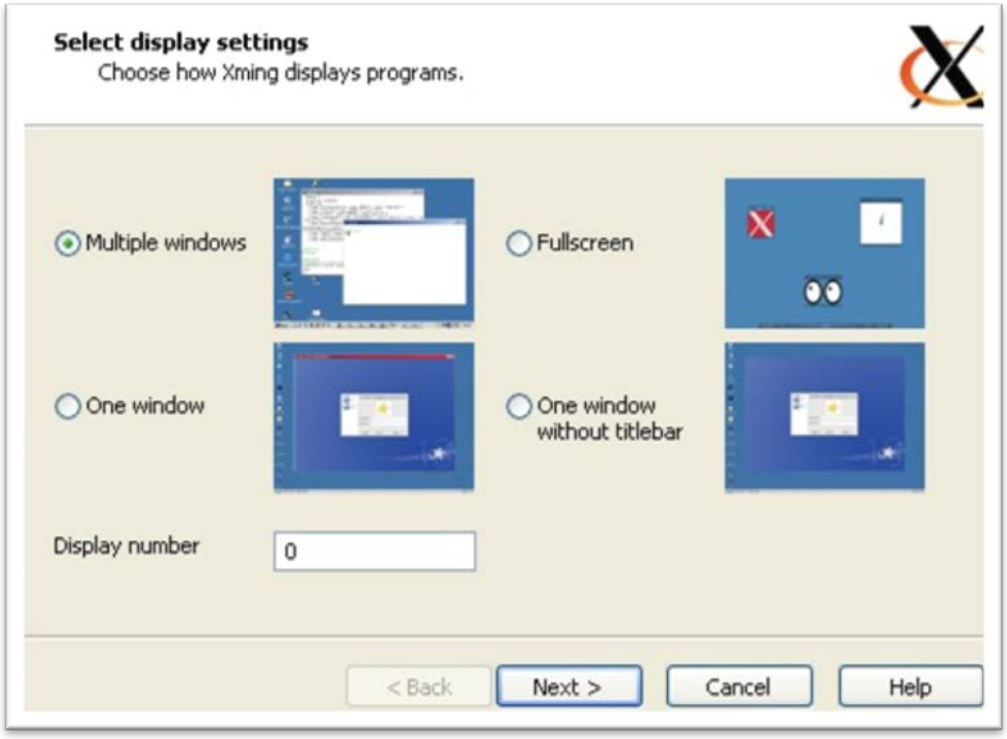


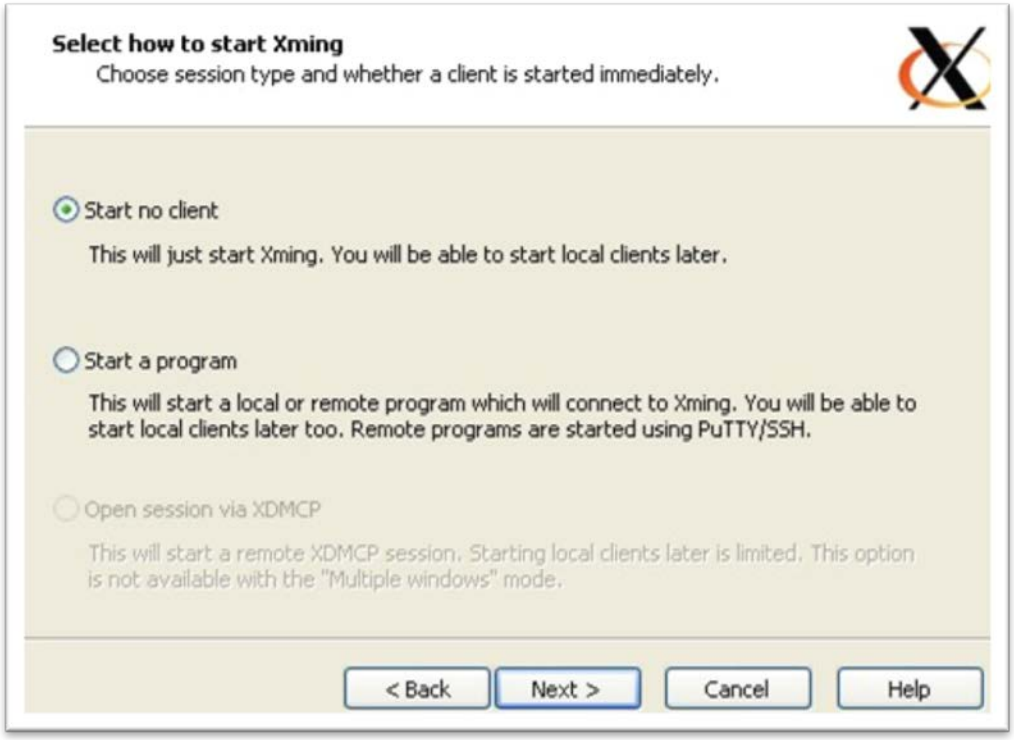
Una vez importada toca configurar las la parte de red, creando los adaptadores necesarios para conectarnos por SSH o si queremos que la red virtualizada tenga acceso al exterior.

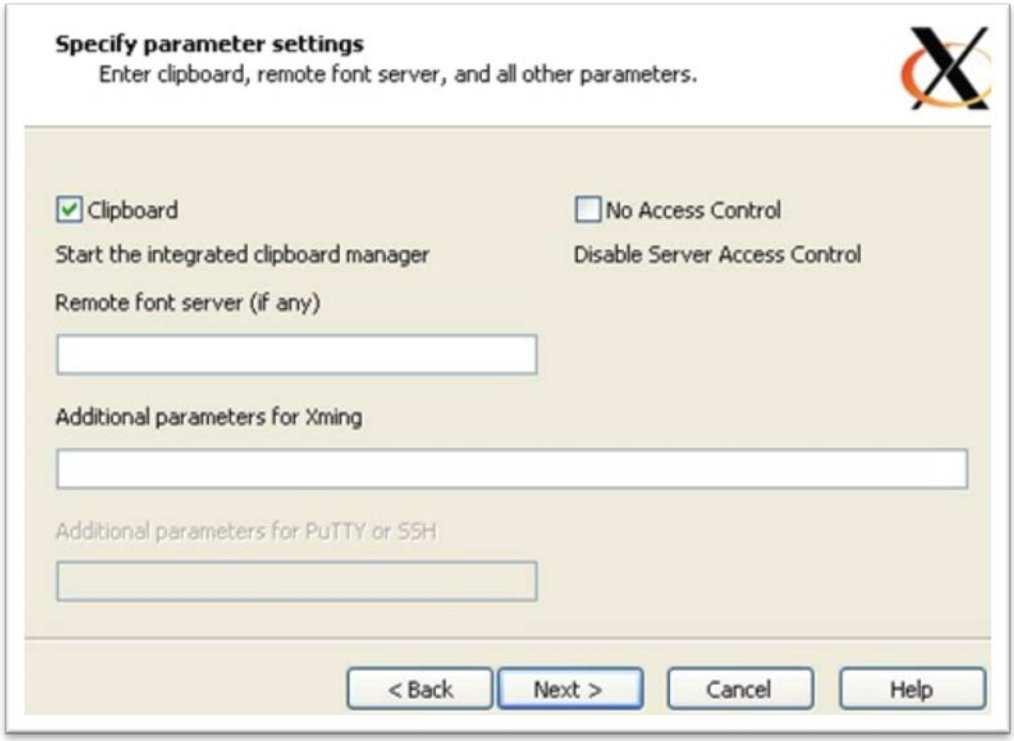
Tras esto toca instalar ***Putty*** es un cliente SSH, Telnet, rlogin, y TCP raw con licencia libre y ***Xming*** que es un emulador de terminal para el sistema de ventanas Windows.

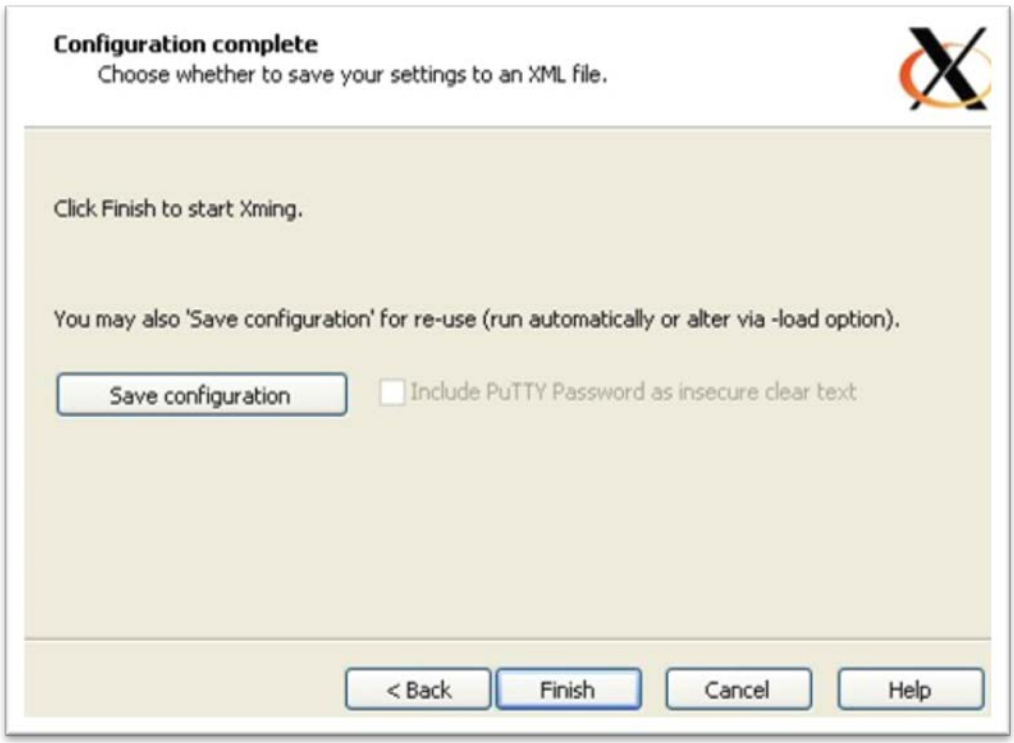
Xming
Descargamos el paquete Xming para la plataforma que estamos utilizando de la página oficial
http://www.straightrunning.com/XmingNotes/

Tras el sencillo instalador en el que elegimos todas las opciones predeterminadas toca configurar Xming corriendo la aplicación '***XLaunch'*** y verificamos que las opciones están como a continuación:


Select display settings
Choose how Xming displays programs.
Multiple windows
Fullscreen
One window
One window without titlebar
Display number
0
< Back
Next >
Cancel
Help


Select how to start Xming
Choose session type and whether a client is started immediately.
Start no client
This will just start Xming. You will be able to start local clients later.
Start a program
This will start a local or remote program which will connect to Xming. You will be able to start local clients later too. Remote programs are started using PuTTY/SSH.
Open session via XDMCP
This will start a remote XDMCP session. Starting local clients later is limited. This option is not available with the "Multiple windows" mode.
< Back
Next >
Cancel
Help





Putty

Descargamos e instalamos putty de su página oficial http://www.putty.org/

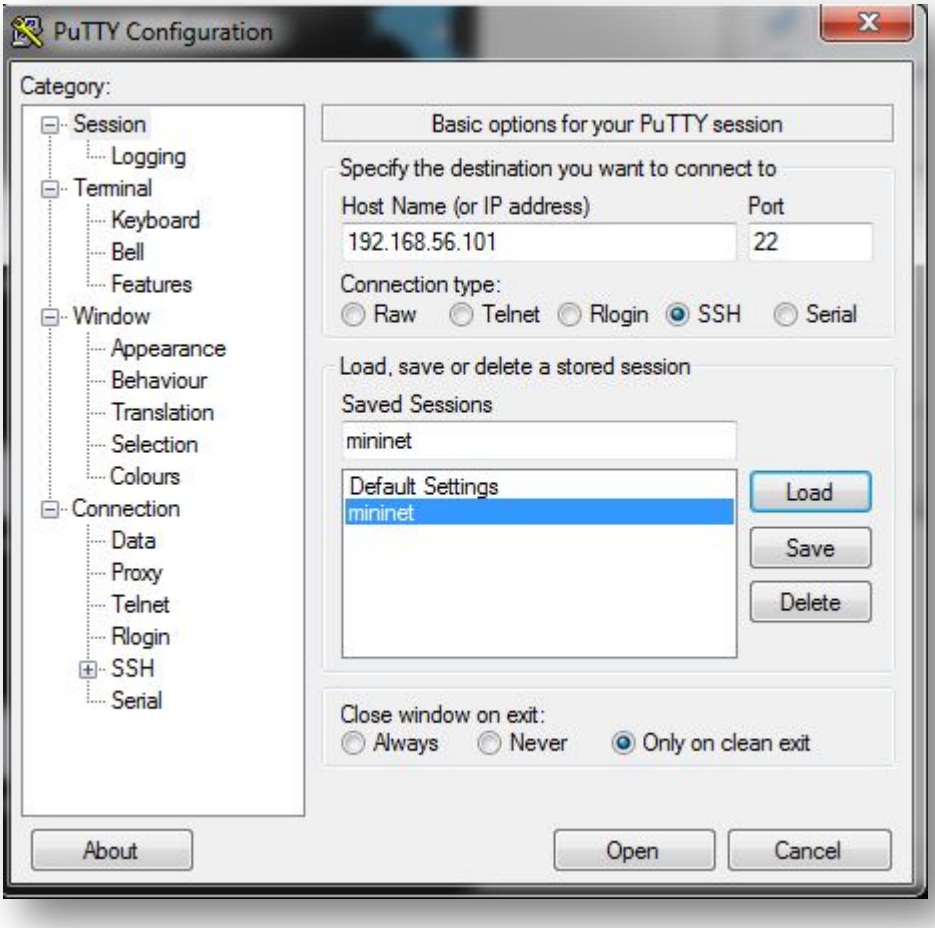
En mi caso he utilizado un cliente portable por comodidad, únicamente ejecutamos.

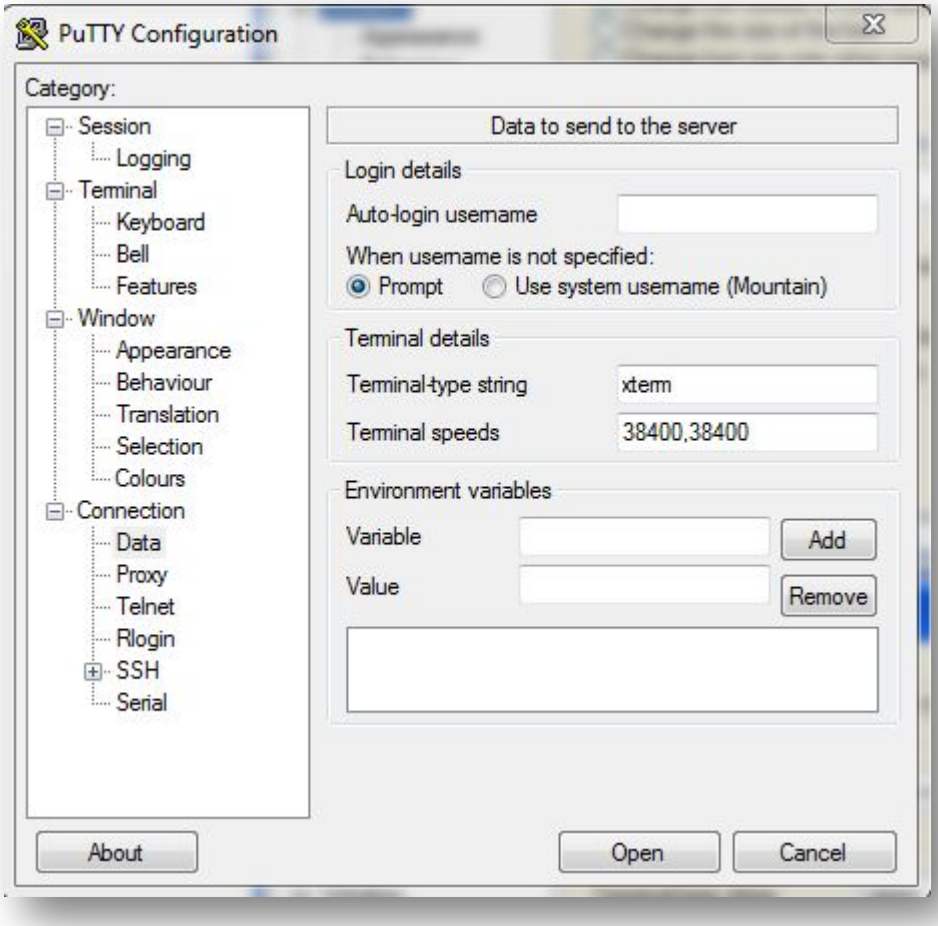
Por la parte de configuración debemos crear una sesión nueva para conectarnos por SSH a MiniNet.

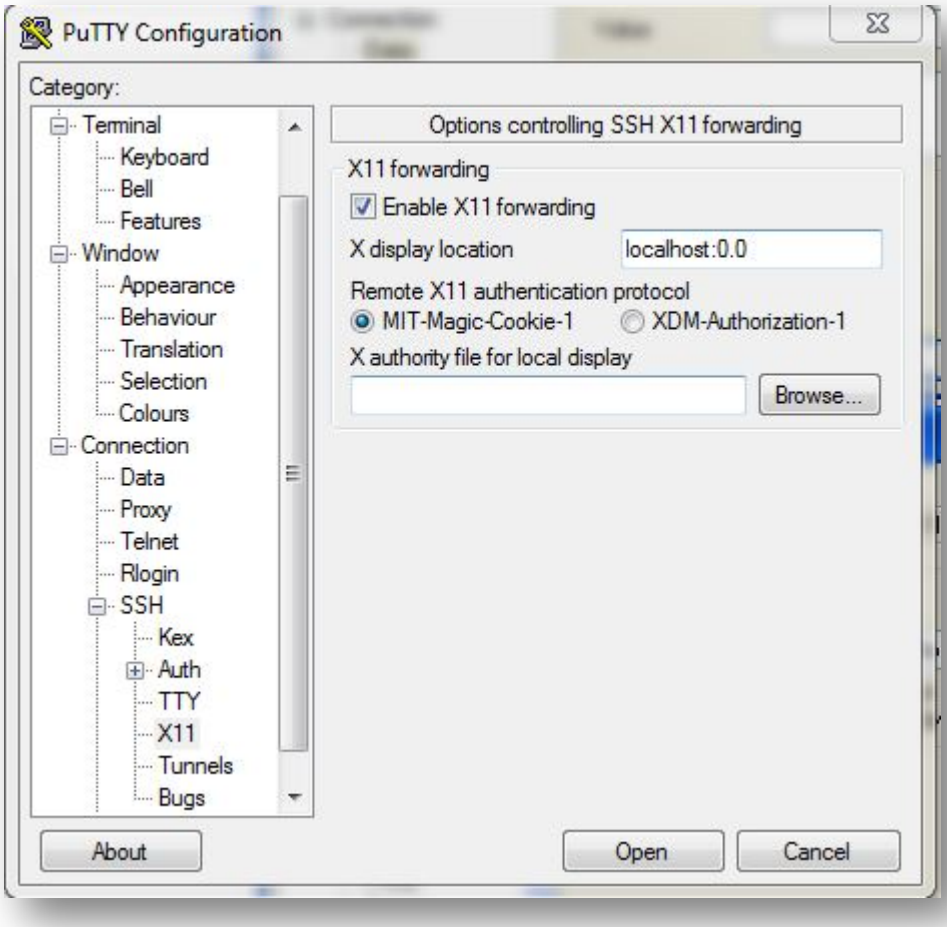
En IP address colocaremos la IP de la interfaz asignada mediante dhcp con el siguiente comando en Ubuntu.

```
sudo dhclient eth1
```

Configuramos PuTTY como muestra:


PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
Data
Proxy
Telnet
Rlogin
SSH
Serial
Basic options for your PuTTY session
Specify the destination you want to connect to
Host Name (or IP address)
Port
192.168.56.101
22
Connection type:
Raw
Telnet
Rlogin
SSH
Serial
Load, save or delete a stored session
Saved Sessions
mininet
Default Settings
mininet
Load
Save
Delete
Close window on exit:
Always
Never
Only on clean exit
About
Open
Cancel


PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
Data
Proxy
Telnet
Rlogin
SSH
Serial
Data to send to the server
Login details
Auto-login username
When username is not specified:
Prompt
Use system username (Mountain)
Terminal details
Terminal-type string
xterm
Terminal speeds
38400,38400
Environment variables
Variable
Add
Value
Remove
About
Open
Cancel


PuTTY Configuration
Category:
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
Data
Proxy
Telnet
Rlogin
SSH
Kex
Auth
TTY
X11
Tunnels
Bugs
Options controlling SSH X11 forwarding
X11 forwarding
Enable X11 forwarding
X display location
localhost:0.0
Remote X11 authentication protocol
MIT-Magic-Cookie-1
XDM-Authorization-1
X authority file for local display
Browse...
About
Open
Cancel

Por último no olvidar guardar la sesión para guardar la configuración, para más tarde cargar el perfil y ya estaremos listos para interactuar con nuestra red SDN virtualizada.