Escuela Politécnica Superior de Linares

Grado en Ingeniería Telemática

Universidad de Jaén
Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

_____

Universidad de Jaén
Escuela Politécnica Superior de Linares

ESTUDIO Y DESPLIEGUE DE REDES DEFINIDAS POR SOFTWARE

Alumno: **Ángela María Sánchez – Valdepeñas López**

Tutor: Antonio Jesús Yuste Delgado y Juan Carlos Cuevas Martínez

Depto.: Ingeniería de Telecomunicación

Octubre, 2017

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

1. RESUMEN

Las Redes definidas por software (SDN) son un nuevo concepto que simplifica la creación y gestión de redes. El desacoplamiento entre el plano control y el plano de reenvío de la red permite el control de todo el comportamiento de la red mediante un elemento lógico centralizado denominado controlador. Esta separación de los planos permite la virtualización de redes.

Además, las Redes Definidas por Software se sirven del protocolo OpenFlow, el cual permite la programación remota del plano de control. Este protocolo es un elemento vital en las Redes Definidas por Software

En este proyecto se puede ver en qué están basados SDN y OpenFlow en profundidad y sus herramientas de desarrollo. Además, también se verán unos ejemplos de SDN y OpenFlow.

ABSTRACT

Software Defined Networks are a new concept that simplifies the network creation and management. The decoupling between the control plane and the forwarding plane of the network allows the control of the network behavior through a centralized logic element, which is the controller. This separation of the planes allows the virtualization of networks.

In addition, Software Defined Networks offers the OpenFlow protocol that allows remote programming of the control plane. This protocol is a vital element in Software Defined Networks

In this project it is shown a deep description of SDN and OpenFlow and their development tools. In addition, some examples of SDN and OpenFlow are shown below.

2. INTRODUCCIÓN

En este proyecto se explica qué son las Redes Definidas por Software (SDN, siglas del inglés *Software Defined Networking*), las cuales vienen precedidas por la llegada del '*big data*'[1], la entrada del '*cloud computing*'[2] y por el hecho de que las comunicaciones de hoy en día ya no son únicamente entre servidor y cliente. Además, se debe tener en cuenta que cada vez se genera una mayor cantidad de tráfico entre máquinas para cumplir con la función de devolver los datos al usuario final, haciendo esto que se esté busquen nuevos protocolos para afrontar los problemas de las redes modernas.

SDN propone el control de los dispositivos que componen la red desde un software externo, apoyándose en el protocolo desarrollado para este fin llamado OpenFlow. SDN propone el desacoplamiento entre plano de control y el plano de reenvío para así controlar todo el comportamiento de la red mediante un elemento lógico centralizado, llamado controlador. Esta separación de los planos permite la virtualización de redes, proporcionando a los usuarios una abstracción lógica de los recursos subyacentes de red.

En el presente trabajo fin de grado se presenta un estado del arte donde se detallan las tecnologías anteriores a SDN y cómo han ido evolucionando las redes de ordenadores ante los problemas que han ido surgiendo en dichas redes, y cómo han desembocado en este nuevo concepto. A continuación, se explica cómo funciona y en qué está basado SDN y el protocolo que lo rige que es OpenFlow.

En cuanto al protocolo OpenFlow, se ha hecho una búsqueda exhaustiva de información sobre sus características, funcionalidad, arquitectura, etc., que ha permitido conocer con detalle esta tecnología. Además, se presentan los resultados de la búsqueda de información sobre todos los proyectos que utilizan OpenFlow y sus plataformas de desarrollo, para poder elegir así un simulador y poder ver cómo se usa y exponer algunos ejemplos.

En la parte de pruebas y discusión de resultados se presentan ejemplos, que muestran las características más prácticas de SDN y OpenFlow. Y por último, se hace una conclusión del trabajo realizado y las líneas futuras de trabajo que esta tecnología puede ofrecer.

[1] Es un concepto que hace referencia a un conjunto de datos tan grandes que aplicaciones informáticas tradicionales de procesamiento de datos no son suficientes para tratar con ellos y los procedimientos usados para encontrar patrones repetitivos dentro de esos datos.
[2] Es un paradigma que permite ofrecer servicios de computación a través de una red, que usualmente es Internet.

3. OBJETIVOS

Los objetivos para llevar a cabo el estudio y despliegue de redes definidas por software (SDN) y cuáles son los proyectos desarrollados hasta el momento. Además del estudio de los simuladores y la descripción detallada de uno de ellos.

Los objetivos principales del presente estudio son los siguientes:

1. Estudio de las técnicas relacionadas con Software Defined Network (SDN)
2. Estudio de simuladores que permitan usar escenarios con esta técnica.
3. Elección y descripción detallada de un simulador.
4. Estudio de un caso práctico.

4. METODOLOGÍA A DESARROLLAR

4.1 ESTADO DEL ARTE

El hecho de intentar resolver muchos problemas de forma distribuida a través de diferentes protocolos en el plano de control dio lugar al nacimiento de las redes definidas por software o SDN. Esto ha dado lugar a la inercia en la evolución de la conmutación de red en comparación con las tecnologías de cálculo y almacenamiento.

En general, la evolución de la conmutación de red ha evolucionado en una solución demasiado cerrada y por tanto existe la necesidad de un modelo más cercano al desarrollo en código abierto (como lo que representa Linux respecto a los sistemas operativos bajo licencia de pago como Windows).

También hay que tener en cuenta que los fabricantes de equipos de red o NEM (del inglés *Network Equipment Manufacturer*) son incapaces de innovar a la velocidad requerida por la evolución actual de las redes. Además, el presente modelo de negocio de los equipos de redes es un mercado con alto margen de beneficio, motivo por el que los principales comerciantes no quieren que su estatus cambie drásticamente.

Por lo tanto, la necesidad de SDN se ha hecho evidente para muchos usuarios de equipos de red, especialmente los que son usuarios de grandes centros de datos. En este capítulo se va a ver cómo comenzó el movimiento SDN y para ello se verá la evolución de las redes hasta la llegada de SDN.

4.2 LA EVOLUCIÓN DE LA TECNOLOGÍA DE LAS REDES.

Antes de OpenFlow, y ciertamente antes del nacimiento del término *Software Defined Networking*, investigadores y tecnólogos estaban considerando cambios fundamentales en el mundo actual de los dispositivos autónomos e independientes y la inteligencia y control de las redes distribuidas.

En la Tabla 1 se muestran los períodos de tiempo aproximados en los que las respectivas tecnologías fueron desarrolladas y aplicadas.

PROYECTO	DESCRIPCIÓN
Señalización Abierta [1]	Separación de los planos de control y reenvío en la conmutación ATM en 1999
Activación de redes [2]	Switches de control y de separación de control a finales

	de los 90
DCAN [3]	Separación de los planos de control y reenvío en la conmutación ATM en 1999
Conmutación IP [4]	Switches de control de la capa 2 como la capa 3 de enrutamiento a finales de los 90
MPLS [RFC3031] [5]	Separando el software de control, estableciendo caminos semi-estáticos de reenvío para los flujos en los routers tradicionales después de los 90
RADIUS [6] Y COPS [7]	Uso del control de acceso para proporcionar dinámicamente políticas en 2010
Orquestación [8]	Uso de SNMP y CLI para ayudar a automatizar la configuración de equipos de red en 2008
Gestión de la virtualización [9]	Uso de módulos para realizar la reconfiguración de red para dar soporte a la virtualización de servidores.
ForCES [10]	Separación de los planos de reenvío y control en 2003
4D [11]	Localizar la inteligencia del plano de control en un sistema centralizado en 2005
Ethane [11]	Lograr el acceso y el control completos del trabajo y la red utilizando los planos de control y de reenvío separados y utilizando un controlador centralizado

Tabla 1. Precursores de SDN

(Fuente: Software Defined Networks A Comprehensive Approach)

Dos ejemplos de los primeros trabajos anteriores a SDN fueron el control descentralizado de las redes ATM (DCAN) y su señalización abierta. DCAN [12] dispuso la separación del control y la gestión de ATM en los mismos switches. El control sería asumido por un dispositivo externo que haría un papel similar al del controlador en redes SDN.

La señalización abierta [1] propuso un conjunto de interfaces abiertas y programables para el hardware de conmutación de ATM. El concepto clave era separar el software de control del hardware de conmutación, que dio lugar al *General Switch*

Management Protocol (GSMP) [13]. En GSMP, el controlador centralizado es capaz de establecer y liberar conexiones en un switch ATM, así como más funciones que se pueden lograr a través de protocolos distribuidos a través de un router tradicional.

Después las mejoras sobre ATM apareció la conmutación de etiquetas que finalmente se conoció como *Mutiprotocol Label Switching* (MPLS). MPLS es un mecanismo de transporte de datos que fue diseñado para unificar el servicio de transporte de datos para las redes basadas en paquetes y circuitos, que permite el transporte de diferentes tipos de tráfico.

A finales de los años 90, se empezó a utilizar el protocolo GSMP para configurar y desmantelar conexiones internas. Los switches presentaban externamente interfaces de enrutador de Internet normales, pero su conmutación interna podía utilizar conmutadores ATM para flujos persistentes. Los flujos fueron definidos como un conjunto de paquetes relativamente persistentes entre los mismos dos extremos, donde el punto final era determinado por la dirección IP y el puerto TCP o UDP. Dado que había algunos gastos generales en el establecimiento de la conexión ATM que llevaría ese flujo, este esfuerzo adicional se subsanaría sólo si el switch que formaba parte de la red IP creía que un número relativamente grande de los paquetes se intercambiarían, entre los extremos, en un corto periodo de tiempo. Esta definición de flujo es algo coherente con el flujo en redes SDN.

El proyecto de activación de redes [2,14,15] también incluye el concepto de switches que podrían ser programados por protocolos fuera de banda. Estos protocolos son un tipo de protocolo de comunicación diferente al de administración, como el por ejemplo el protocolo simple de administración de red o SNMP (del inglés *Simple Network Management Protocol)*, u otro protocolo propietario específico del distribuidor. Las redes activas incluyeron una propuesta muy novedosa en forma de pequeños programas descargables que viajaban en paquetes a los routers y que podrían reprogramar el comportamiento de un router sobre la marcha.

Los productos de control de acceso a la red [16] (NAC) controlan el acceso a una red basada en políticas establecidas por la administración de la red. El control más básico se basa en determinar si admitir o no al usuario en la red. Esta decisión suele llevarse a cabo mediante un intercambio de credenciales entre los usuarios y la red. Si es admitido, sus derechos de acceso serán restringidos según las políticas establecidas. A veces, la política de red, más allá del control de admisión en la red, se aprovisionó dinámicamente a través de métodos NAC, como el Servicio de Usuarios de Acceso Telefónico de Autenticación Remota (RADIUS) [16].

RADIUS se ha utilizado para proporcionar la reconfiguración automática de la red. Esta solución fue de la manera en la que RADIUS fue visto como un precursor de SDN.

Considerando que RADIUS fue diseñado originalmente para procesos de autenticación, autorización y contabilidad, que están relacionados con la concesión de acceso de un usuario a una red. Podemos ver que el caso original está bien relacionado con el problema de la reconfiguración de la red. Por tanto, la identidad del recurso que se conecta a la red sirve para identificar el recurso en la base de datos de RADIUS y la autorización de los atributos devueltos de esa base de datos (BD) podría utilizarse para cambiar los atributos de red. Este tipo de soluciones lograron la reconfiguración automática de la red, pero nunca obtuvieron plena confianza de los administradores de telecomunicaciones.

Los primeros intentos de automatización involucraron aplicaciones que eran llamadas orquestadoras. Tales aplicaciones podrían tomar un comando u objetivo y aplicarlo a través de una amplia gama de dispositivos. Además, utilizaban dispositivos con interfaces de programación de aplicaciones (API) como la interfaz de línea de comandos (CLI, del inglés *Command Line Interface*) o el protocolo simple de administración de red (SNMP). Una solución de este tipo puede tener ciertas políticas de nivel superior que son ejecutadas en niveles inferiores por los complementos adecuados. Los complementos específicos del proveedor se utilizan para convertir las solicitudes de las políticas de nivel superior en solicitudes de CLI específicas para cada proveedor. Las aplicaciones de gestión de orquestación fueron útiles para las tareas de actualización de *firmware*, pero no para automatizar los centros de datos actuales.

El concepto de administrador de virtualización de la red se basa en la orquestación y los intentos de automatizar las actualizaciones de red que se requieren en un entorno de virtualización. Las herramientas dirigidas específicamente para el centro de datos, podría implicar muy a menudo los módulos de administrador de virtualización, es decir complementos para el VCenter[18], que sería configurado para actuar en caso de un cambio de servidor, como un VMotion[19]. Estos módulos tomaban las decisiones adecuadas en los dispositivos de red que controlaban que el servidor siguiese en funcionamiento. El mecanismo para realizar cambios en los dispositivos de red se hizo a través de comandos SNMP o la interfaz de línea de comandos CLI. Estos módulos, al utilizar capacidades de configuración de SNMP y CLI, suelen ser difíciles de manejar y propensos a errores.

En la Figura 1 se muestra una visión general del diseño y el funcionamiento de una solución de este tipo para automatizar el proceso de creación de conectividad de red en un servidor virtual.

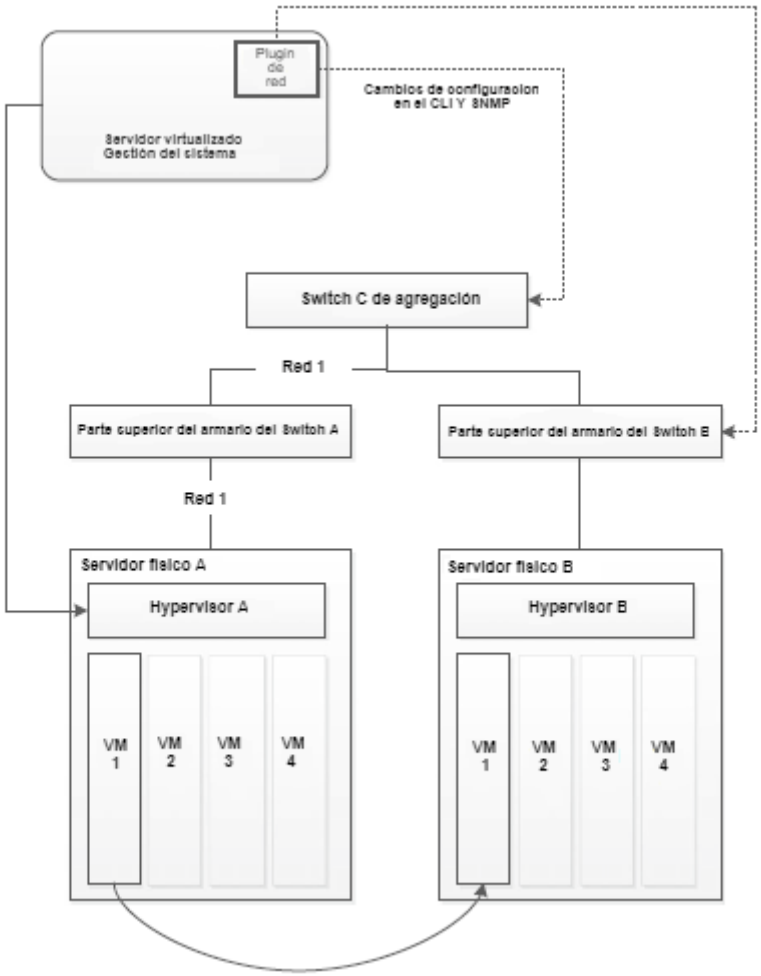


Figura 1. Primeros intentos de SDN: módulos

La Separación de Elementos de Transmisión y Control (ForCES) [10] fue una de las propuestas que recomendaban el desacoplamiento de los planos de trasmisión y control. La idea general de ForCES era proporcionar una entidad sencilla de reenvío basada en hardware. La base de software de control tiene la responsabilidad de las tareas más amplias, que a menudo implican la coordinación de múltiples dispositivos de red. Los componentes funcionales de ForCES son:

- **Elemento de reenvío** (FE, del inglés *Forward Element*). Se implementa normalmente en hardware y está situado en la red. Además, es responsable de la aplicación de los procedimientos de reenvío y filtrado de las reglas que recibe del controlador.
- **Elemento de control** (CE, del inglés *Control Element*). se usa par a la coordinación entre los dispositivos de la red individuales y para el reenvío de comunicaciones y la información de enrutamiento a los FEs.

ForCES propone la separación del plano de reenvío del plano de control. La fuerza real de ForCES reside en su modelo, que permite la descripción de una nueva funcionalidad de ruta de datos sin cambiar el protocolo entre el plano de control y el plano de reenvío. Durante el estudio de cómo trasladar la tecnología redes de elementos, se puedo estudiar como pasar de las redes distribuidas a un controlador centralizado. De esta manera, apareció el 4D [11] que lleva el nombre de los cuatro planos de su arquitectura: decisión, difusión, descubrimiento y datos. Esta nueva arquitectura propone una refactorización completa de la red dejando fuera dispositivos autónomos y que se dirige hacia la idea de concentrar todas las operaciones en el plano de control en un sistema separado dedicado solamente a ese propósito.

El 4D sostiene que el estado de la red de hoy en día es frágil debido a su diseño actual basado en sistemas distribuidos y autónomos. Esto hace que un pequeño evento local, o una configuración errónea de un protocolo de enrutamiento, puedan generar problemas graves. La causa es que el plano de control se está ejecutando sobre los mismos elementos de red. El 4D se centra alrededor de los siguientes tres principios:

- **Objetivos a nivel de red.** Las metas y objetivos del sistema de red deben comenzar en términos de nivel de red, basados en el dominio completo de la red y de los elementos separados de la red en lugar de en términos relacionados con el rendimiento individual del dispositivo de red.
- **Visión de toda la red.** Debe haber una visión completa de la red. La topología, el tráfico y los eventos de todo el sistema debe ser la base sobre la cual se toman las decisiones.
- **Control directo.** Los sistemas de control y gestión deberían poder ejercer un control directo sobre los elementos de red, con la capacidad de programar las tablas de reenvío.

La propuesta 4D propone algunos de los desafíos a los que se enfrenta la arquitectura centralizada. Estos retos siguen siendo relevantes hoy en día en SDN como son la latencia, la adaptabilidad, la seguridad y la alta disponibilidad.

ForCES y 4D contribuyeron en gran medida a la evolución de los conceptos que subyacen a SDN con ideas y soluciones tales como la separación de los planos de control y reenvío de ForCES, o el controlador centralizado responsable del enrutamiento y reenvío de decisiones de 4D.

Ethane, introducido por Rethinking Enterprise Network Control, es una solución basada en políticas que permite a los administradores de red definir políticas a nivel de red

de acceso para los usuarios, que incluye la autenticación y la cuarentena para los usuarios que no hacen un buen uso. Ethane se basa en tres principios básicos:

- **La red debe estar dirigida por políticas de alto nivel.** Ethane apoya la idea de que la red sea dirigida por esas políticas en lugar de que la dirija un dispositivo.
- **El enrutamiento de la red debe ser consciente de estas políticas.** Las rutas que los paquetes toman, se eligen a través de las políticas de nivel superior, no como hoy en día que los caminos se eligen basados en directivas de nivel inferior. Estas políticas de alto nivel son pautas muy poderosas para dirigir y organizar el tráfico.
- **La red debe hacer cumplir la vinculación entre los paquetes y su origen.** Si las decisiones de las políticas dependen de conceptos de nivel superior, como el concepto de usuario, entonces los paquetes que circulan en la red pueden ser rastreables hasta su punto de origen (por ejemplo, el usuario o máquina que es la fuente real del paquete).

En la Figura 2 se muestran los fundamentos de la solución de Ethane. Hay muchas similitudes entre esta solución y OpenFlow. Para probar Ethane [11] se desarrollaron switches para que implementaran el protocolo y el comportamiento del switch, es decir que permitiera la funcionalidad del plano de control por una entidad externa (el controlador) y que se comunica con esa entidad externa a través del protocolo, que le permite que las entradas de flujo se configuren en sus tablas de flujo locales, con las que luego realizarán las funciones de reenvío a medida que lleguen los paquetes.

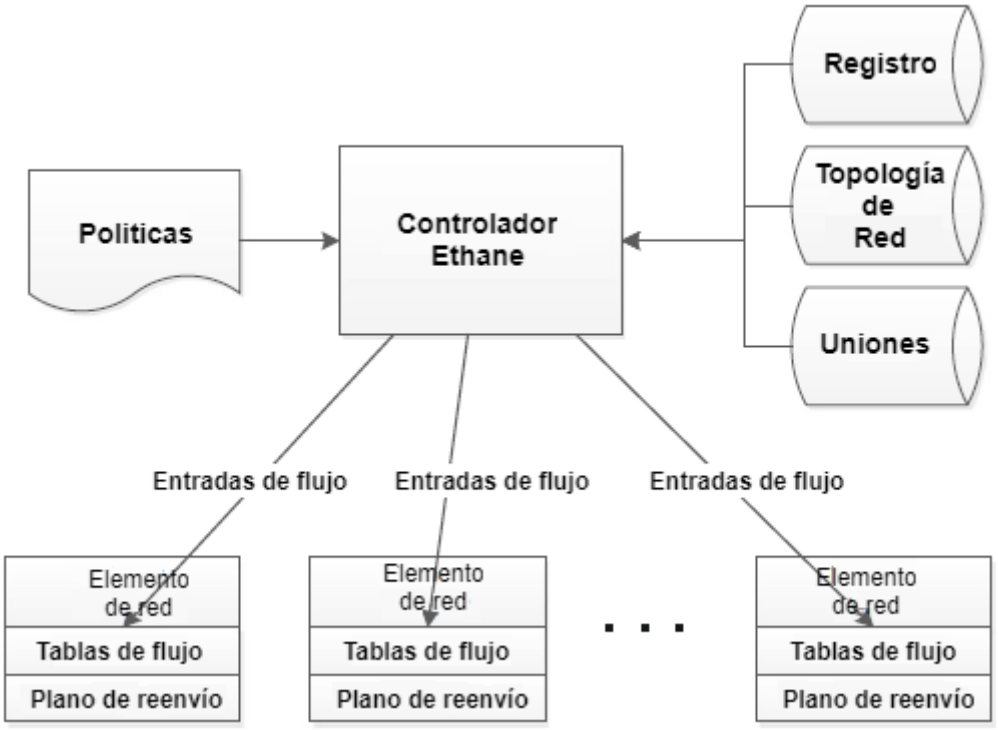


Figura 2. Arquitectura Ethane

El comportamiento de los switches Ethane es generalmente el mismo que el de los switches OpenFlow de hoy, que reenvían y filtran paquetes basados en tablas de flujo que se han configurado en el dispositivo. Si el switch no sabe qué hacer con el paquete, lo envía al controlador y espera instrucciones adicionales.

En resumen, Ethane es básicamente una tecnología de redes definidas por software, y sus componentes son antecedentes de OpenFlow, que se describen en detalle en el apartado 5.

4.3 APARICIÓN DE LAS REDES DEFINIDAS POR SOFTWARE

Se puede considerar que la aparición de OpenFlow [31], que se explicará en profundidad en el apartado 5, fue realmente el punto en el que nació SDN [17]. La realidad es que el término SDN no entró en uso hasta un año después de que OpenFlow apareciera en 2008, pero la existencia y adopción de OpenFlow por las comunidades de investigadores y las redes de vendedores marcó un cambio radical en el establecimiento de una red.

El término SDN estaba en uso en las comunidades de investigación sobre 2009, pero no conllevó un gran impacto en la industria de las redes hasta 2011.

OpenFlow fue desarrollado y diseñado para permitir a los investigadores experimentar e innovar con nuevos protocolos en redes cotidianas. La especificación de OpenFlow animó a los proveedores a implementar y habilitar OpenFlow en sus productos

de conmutación para la implementación en redes universitarias. La implementación de OpenFlow delinea tanto el protocolo que se utilizará en el controlador como el comportamiento esperado en el switch. La Figura 3 ilustra la arquitectura simple de una solución OpenFlow.

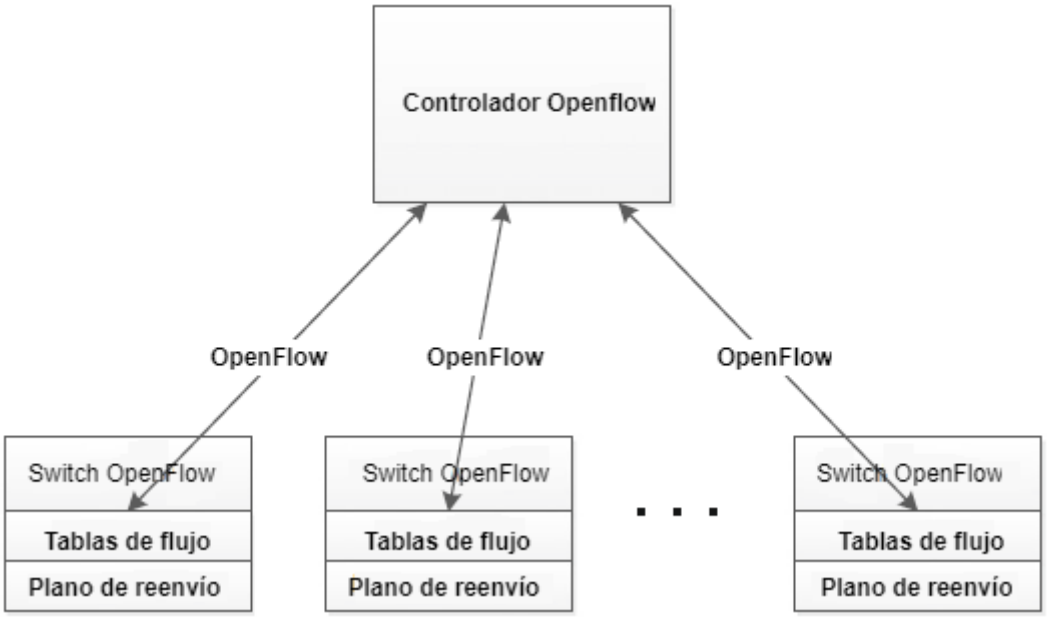


Figura 3. Diseño general de OpenFlow

El funcionamiento básico de OpenFlow se basa en:

- El controlador, que rellena el switch con las entradas de la tabla de flujo.
- El switch evalúa los paquetes entrantes y encuentra un flujo coincidente, luego realiza la acción asociada.
- Si no encuentra ninguna coincidencia, el switch envía el paquete al controlador para obtener instrucciones de cómo tratar el paquete.
- El controlador normalmente actualizará el switch con nuevas entradas de flujo. También es posible que el controlador programe las reglas que dirigirán muchos flujos a la vez.

OpenFlow comenzó con la publicación de su propuesta original en 2008. Para 2011 OpenFlow tenía el suficiente impulso para que la responsabilidad de la norma se trasladara a la *Open Networking Foundation* (ONF). La ONF es la responsable del estándar OpenFlow, que consta de varios grupos de trabajo. Es importante asegurar que OpenFlow permanezca independiente y desvinculado de cualquier institución u organización específica. De esta manera se asegura que las implementaciones de los diferentes estándares en OpenFlow se

adhieran a las normas tal y como se definen, se aclare y amplíe el estándar en donde se encuentran incompletos y en general garantizar la interoperabilidad entre las implementaciones de OpenFlow. Este objetivo se puede lograr de varios modos:

- **Plugfests.** Son entornos, conferencias y congresos donde los proveedores pueden llevar sus dispositivos y software para probar la interoperabilidad con dispositivos y software de otros proveedores. Son oportunidades que pueden aprovechar para determinar dónde puede faltar una implementación o cuando el estándar necesita ser más preciso o específico.
- **Interoperabilidad de los laboratorios.** Hay laboratorios de pruebas que se dedican a la interoperabilidad de equipos de diversos proveedores y organizaciones.
- **Programas de certificación.** Sirven para certificar al comprador que está recibiendo un dispositivo que admite una o varias versiones de OpenFlow.
- **Educación y consultoría.** SDN no penetrará fácilmente en una gran industria sin la existencia de una infraestructura para entrenar y asesorar al personal de red sobre la migración.

4.3.1 EL IMPULSO DEL SOFTWARE LIBRE

Uno de los fundamentos básicos para SDN ha sido la innovación y el desarrollo. Desde la creación de estándares como OpenFlow, se ha avanzado mucho más rápido, gracias al carácter abierto de estas implementaciones, esto debería alentar a los investigadores a eliminar viejos métodos. El progreso tecnológico a veces surge de los esfuerzos de las empresas y las principales organizaciones, que muy a menudo son los únicos que pueden hacer contribuciones.

En el mundo del software, ocasionalmente hay pequeños desarrolladores que ponen su contribución a disposición del público general consiguiendo un gran éxito, convirtiéndose en referencias mundiales. Algunos ejemplos son Linux en cuanto a sistemas operativos de código abierto, SQLite es un ejemplo de motor de base de datos de código abierto, el servidor web Apache, OpenSSL en cuanto a seguridad o BitTorrent un servicio que se apoya en un protocolo de código abierto para compartir archivos.

Sin embargo, el software de código abierto debe someterse a pruebas por un número inmenso de individuos. Esto se debe al hecho de que miles de individuos tienen acceso al código y puede probar esas contribuciones. Para cualquier nueva característica o problema que se esté resolviendo, la comunidad de código abierto puede ofrecer una serie de enfoques competidores para llevar a cabo la tarea. Muchas iniciativas de código abierto están sujetas a ciclos de lanzamiento, y hay personas claves involucradas en la decisión de

qué código será la próxima versión. Este factor minimiza el riesgo de amenazas a la seguridad.

En conclusión, con la investigación y las comunidades de código abierto que quieren un entorno abierto para la expansión, investigación y experimentación, así como la urgente necesidad de que los centros de datos tengan más agilidad y se virtualicen, los proveedores de redes se han visto obligados a entrar en el mundo de SDN.

4.4. ESTUDIO DE LOS SIMULADORES

En este punto se va a realizar un estudio de los simuladores y proyectos de código abierto que están relacionados con OpenFlow y permiten la posible implementación del Software Defined Networking. Durante el avance del capítulo se verá una clasificación en la que se podrá encontrar:

- Software de conmutación y pilas OpenFlow autónomas.
- Plataformas del controlador.
- Controladores de uso especial.
- Miscelánea.

4.4.1 SOFTWARE DE CONMUTACIÓN Y PILAS OPENFLOW AUTÓNOMAS

En este ámbito son muchas las opciones que se pueden encontrar y por tanto, a continuación se va a proceder a ver cuáles son estas opciones.

a) OPEN VSWITCH

Open vSwitch es un conmutador virtual multicapa de producción bajo licencia de Open Source Apache 2.0. Por otro lado, es una pila OpenFlow que se utiliza como conmutador virtual en entornos virtualizados y que ha sido migrada a múltiples plataformas de hardware. Ahora es parte del kernel de Linux. Está diseñado para permitir la automatización masiva de la red a través de la extensión programada, al tiempo que sigue soportando interfaces y protocolos estándar de administración (por ejemplo, NetFlow, sFlow, IPFIX, RSPAN, CLI, LACP, 802.1ag). Además, está diseñado para soportar la distribución a través de múltiples servidores físicos similares a la vnetwork distribuida por VMware vswitch o Nexus 1000v de Cisco.

Las funciones que admite Open vSwitch son las siguientes:

- Visibilidad en la comunicación entre VM a través de NetFlow, sFlow(R), IPFIX, SPAN, RSPAN.
- LACP (IEEE 802.1AX-2008)
- Estándar 802.1Q modelo VLAN con canal

- Multidifusión snooping
- IETF Auto-attach SPBM y soporte rudimentario LLDP
- BFD y monitorización de enlaces 802.1ag
- STP y RTSP
- Control de QoS
- Soporte para HFSC qdisc
- Interfaz de control de tráfico para VM
- Enlace NIC con equilibrio de carga de origen-MAC, respaldo activo y hash L4
- Soporte de protocolo OpenFlow (incluyendo extensiones para la virtualización)
- Compatibilidad con IPv6
- Múltiples protocolos de tunelización (GRE, VXLAN, STT y Geneve, con soporte IPsec)
- Protocolo de configuración remota con C y Phypton
- Opciones del motor del núcleo de reenvío y del espacio de usuario.
- Reenvío de capas de abstracción para facilitar la migración a nuevas plataformas de software y hardware.

b) Pica8

Pica8 es una plataforma de software de conmutación abierta para chips de conmutación de hardware que incluye soporte para OpenFlow. Además, esta plataforma está rompiendo barreras consiguiendo poder personalizar el rendimiento de la aplicación a través de redes abiertas. Con su sistema operativo de red PicOS basado en Linux, Pica8 permite la ingeniería de tráfico personalizada y permite que los switches genéricos se integren fácilmente con las redes existentes de capa 2 y capa 3 y ofrezcan escalabilidad SDN ilimitada a través de OpenFlow.

Desde 2009, Pica8 ha sido pionera en nuevas tecnologías de redes abiertas, tales como redes basadas en Linux, redes CrossFlow, vASIC y patrones de tipos de tablas. A través de la innovación en curso, Pica8 desbloquea el potencial de redes hechas a medida, ofreciendo una alternativa principal a los sistemas propietarios heredados.

c) INDIGO

Indigo es una implementación OpenFlow de hardware de conmutación basada en la implementación de referencia de Stanford. Además, es una implementación OpenFlow de código abierto, que se ejecuta en switches físicos y utiliza las características del hardware de los switch Ethernet de ASICs para ejecutar OpenFlow a velocidades de línea. Actualmente implementa todas las características necesarias del estándar OpenFlow 1.0.

d) Ofilib-node

Ofilib-node es una biblioteca de protocolo OpenFlow para Node. Convierte los mensajes del protocolo de cable OpenFlow y objetos JavaScript. En la actualidad admite el desempaquetado de los mensajes OpenFlow 1.0 y 1.1.

f) ORTIGA

Ortiga es una biblioteca de OpenFlow escrita en el lenguaje de programación Haskell. Por otro lado, esta biblioteca está en desarrollo activo y proporciona:

- Tipos de datos que modelan los mensajes del protocolo OpenFlow
- Funciones que implementan serialización y deserialización entre estos tipos de datos
- Funciones que implementan sus representaciones binarias en el protocolo
- Un servidor OpenFlow eficiente

4.4.2 PLATAFORMAS DEL CONTROLADOR

En este ámbito son muchas y diferentes los proyectos que están en desarrollo y que se pueden encontrar y, por tanto, a continuación, se va a proceder a ver cuáles son estos proyectos.

a) IRIS

OpenIRIS es una versión de IRIS de código abierto. IRIS es un controlador SDN basado en Openflow y diseñado para resolver problemas de escalabilidad y disponibilidad de SDN. Por otro lado, las características principales se enumeran a continuación:

- Soporta Openflow desde la versión 1.0.1 hasta la versión 1.3.2
- La API Openflow está basada en Loxigen
- La implementación pura está basada en Java
- Soporta cerca de 500 conexiones simultáneas a switches con Commodity HW
- Sigue la misma política de licencias que Floodlight (licencia Apache-2.0)
- Proporciona aprendizaje MAC, descubrimiento de vínculos, administración de topología, reenvío, administrador de dispositivos, cortafuegos, switch de falla de red, empujador estático de flujo y módulos de administrador de estado
- Incluye tres implementaciones de controlador de ejemplo, que son totalmente funcionales.

b) OPEN MUL

OpenMUL Controller proporciona una plataforma de control básico para todo SDN/Openflow. Es un controlador SDN/Openflow ligero escrito casi en su totalidad en C (a partir de cero) y proporciona un rendimiento superior en términos de gestión de flujo (velocidad de descarga y latencia), así como una plataforma de desarrollo de aplicaciones muy estable.

Los siguientes son los principales componentes del controlador OpenMUL:

1. Núcleo Mul
    a. Componente principal de Mul
    b. Maneja todas las conexiones de los switches de nivel bajo y realiza el procesamiento de Openflow
    c. Proporciona la interfaz de programación de aplicaciones en forma de MLAPIS (APIs de nivel medio)
    d. Proporciona acceso agnóstico Openflow (o cualquier protocolo conectado al sur) a los dispositivos
    e. Soporta ganchos para la infraestructura de procesamiento de aprendizaje de baja latencia de alta velocidad (reactiva)
    f. Asegura que todos los flujos, grupos, medidores y otras entidades específicas del conmutador se mantengan sincronizados a través del conmutador, los reinicios / fallos del controlador.
2. Servicios de infraestructura Mul
    a. Estos proporcionan servicios de infraestructura básicos construidos en la parte superior del núcleo de Mul
    b. Actualmente disponible:
        i. **Servicio de detección de topologías**. Este servicio utiliza paquetes LLDP para descubrir la topología de conmutadores de red
        ii. **Servicio de búsqueda de rutas**. Este servicio utiliza el algoritmo Flloyd-Warshall (todos los pares de ruta más corta) para calcular la ruta más corta entre dos nodos de red.
        iii. **Path Connector Service**. Este servicio proporciona una interfaz flexible para que la aplicación instale flujos a través de una ruta. Otra característica importante de este servicio es que separa un dominio SDN en núcleo y borde. Los flujos solicitados por la aplicación sólo se mantienen en los switches de borde. Esto mantiene el flujo (un recurso costoso en el mundo de Openflow) en los conmutadores de núcleo al mínimo.
3. Aplicaciones del sistema Mul

a. Las aplicaciones del sistema se crean utilizando una API común proporcionada por el núcleo Mul y los servicios Mul
b. Estas son apenas conscientes de Openflow y, por tanto, están diseñadas para trabajar a través de diferentes versiones de OpenFlow siempre que los switches sean compatibles.
c. Actualmente disponible:
    i. **L2switch**: Una aplicación que proporciona una forma rápida de verificar la conectividad de red, en una pequeña configuración de red.
    ii. **Aplicación CLI:** proporciona una herramienta de aprovisionamiento basada en CLI común para todos los componentes Mul.
    iii. **Servidor web NBAPI:** Esto proporciona una API RESTful para los controladores de Mul. Webserver escrito en Python

c) TREMA

Trema es un marco de programación de controladores OpenFlow que proporciona todo lo necesario para crear controladores OpenFlow en Ruby. Proporciona una biblioteca OpenFlow de alto nivel y también un emulador de red que puede crear redes basadas en OpenFlow para probar en el PC. Este entorno autónomo ayuda a agilizar todo el proceso de desarrollo y pruebas.

d) BEACON

Beacon es un controlador OpenFlow basado en Java, rápido, multiplataforma y que soporta tanto operaciones basadas en eventos como en subprocesos. Además, cuenta con una serie de características las cuales, las principales son:

- **Estable**: Beacon ha estado en desarrollo desde principios de 2010, y se ha utilizado en varios proyectos de investigación, clases de redes y despliegues de prueba. Beacon actualmente alimenta un switch virtual de 100 switches, a 20 switches físicos y se ha ejecutado durante meses sin tiempo de inactividad.
- **Plataforma cruzada**: Beacon está escrito en Java y funciona en muchas plataformas, desde servidores Linux multi-core de gama alta hasta en teléfonos Android.
- **Código abierto**: Beacon está bajo una combinación de licencias, la licencia GPL v2 y la licencia de la universidad de Stanford v1.0
- **Dinámico**: Los paquetes de código en Beacon se pueden iniciar, detener, actualizar o instalar en tiempo de ejecución, sin interrumpir otros paquetes no dependientes (es decir, reemplazar la aplicación de aprendizaje del Switch sin necesidad de desconectar los switches.

- **Rápido desarrollo**. Beacon es fácil de poner en marcha. Java y eclipse simplifican el desarrollo y la depuración de sus aplicaciones.
- **Veloz**: Beacon es multiproceso.
- **Web UI**: Beacon incorpora opcionalmente el servidor web empresarial Jetty y un marco de interfaz de usuario extensible y personalizado.
- **Frameworks**. Beacon se basa en plataformas de Java como es Spring y Equinox.

e) FLOODLIGHT

Floodlight es un controlador SDN abierto. El controlador Floodlight Open SDN es un controlador OpenFlow basado en Java con licencia Apache. Es apoyado por una comunidad de desarrolladores, incluyendo un gran número de ingenieros de Big Switch Networks. Este controlador tiene algunas características que cabe destacar:

- Ofrece un sistema de carga de módulos que lo hace fácil de ampliar y mejorar
- Es fácil de configurar con dependencias mínimas
- Soporta una gran gama de switches OpenFlow virtuales y físicos
- Puede gestionar redes mixtas OpenFlow y no OpenFlow, además puede gestionar múltiples “islas” de switches de hardware OpenFlow
- Soporte para plataforma de orquestación OpenStack

f) MAESTRO

Maestro es un “sistema operativo” para orquestar aplicaciones de control de red. Además, proporciona interfaces para implementar aplicaciones de control de red modular para acceder y modificar el estado de la red y coordinar sus interacciones. Maestro es una plataforma para lograr funciones de control de red automáticas utilizando estas aplicaciones modulares. Aunque este proyecto se centra en la construcción de un controlador OpenFlow con Maestro, Maestro no sólo se limita a las redes OpenFlow. El marco de programación de Maestro proporciona interfaces para:

- Introducción de nuevas funciones de control personalizadas mediante la adición de componentes de control modulares.
- Mantener el estado de la red en nombre de los componentes de control.
- Composición de componentes de control especificando la secuencia de ejecución y el estado de red compartida de los componentes.

Además, Maestro trata de explotar el paralelismo dentro de una sola máquina para mejorar el rendimiento del rendimiento del sistema. La característica fundamental de una red OpenFlow es que el controlador es responsable del establecimiento inicial de cada flujo

poniéndose en contacto conmutadores relacionados. Así, el rendimiento del controlador podría ser el cuello de botella. Al diseñar Maestro se trató de requerir el menor esfuerzo posible de los programadores para gestionar el paralelismo. En su lugar, Maestro maneja la mayor parte del trabajo tedioso y complicado de gestionar la distribución de la carga de trabajo y la programación de los hilos de trabajo. Por diseño, Maestro es portable y escalable:

- Desarrollado en Java (tanto la plataforma como los componentes) - Muy portable para varios sistemas operativos y arquitecturas.
- Multi-hebra - Aprovecha al máximo los procesadores multi-core.

Maestro actualmente proporciona los componentes de control para realizar una red de conmutación de aprendizaje o una red encaminada que utiliza conmutadores OpenFlow. Algunos componentes, como la consola de la línea de comandos, etc., todavía no están completos.

g) NODEFLOW

NodeFlow es un controlador OpenFlow escrito en JavaScript para Node.JS. Node.JS proporciona una biblioteca asíncrona sobre JavaScript para la programación del lado del servidor que es perfecta para escribir aplicaciones basadas en red (aquellas que no requieren una cantidad excesiva de CPU).

NodeFlow es un programa muy simple que depende en gran medida de un intérprete de protocolo llamado OFLIB-NODE. A pesar de que el Open Networking Forum ha ratificado la especificación del protocolo 1.2, todavía se necesita un diseño de referencia que permita a los desarrolladores experimentar.

Aprovechando OpenvSwitch y herramientas como Mininet, se facilita la creación de un entorno de red simulado dentro de una máquina local.

h) OESS

OESS es una aplicación para configurar y controlar los conmutadores habilitados para OpenFlow. A través de una interfaz de usuario muy sencilla y fácil de usar. OESS proporciona un segundo circuito de aprovisionamiento, conmutación de errores automáticos, permisos de interfaz y estadísticas automáticas por VLAN.

OESS requiere varios paquetes para trabajar, el núcleo está basado en Apache, y MySQL, sin embargo, hay otros paquetes necesarios.

La interfaz de usuario de administración proporciona la capacidad de controlar casi todos los aspectos del software de OESS. Incluyendo usuarios, grupos de trabajo, puertos, nodos y enlaces.

4.4.3 CONTROLADORES DE USO ESPECIAL

En este ámbito son muchas y diferentes los proyectos que están en desarrollo y que se pueden encontrar y, por tanto, a continuación, se va a proceder a ver cuáles son estos proyectos.

a) ROUTEFLOW

RouteFlow, es un proyecto de código abierto para proporcionar servicios de enrutamiento IP virtuales sobre hardware habilitado para OpenFlow.

RouteFlow está compuesto por una aplicación OpenFlow Controller, un servidor RouteFlow independiente y un entorno de red virtual que reproduce la conectividad de una infraestructura física y ejecuta motores de enrutamiento IP (por ejemplo, Quagga). Los motores de enrutamiento generan la base de información de reenvío (FIB) en las tablas IP de Linux según los protocolos de enrutamiento configurados (por ejemplo, OSPF, BGP). A su vez, las tablas IP y ARP se recopilan mediante procesos RouteFlow Slave y luego se traducen en tuplas OpenFlow que finalmente se instalan en los dispositivos asociados a OpenFlow en el plano de reenvío.

RouteFlow es la evolución de QuagFlow - Partnering Quagga con OpenFlow y funciona de forma transparente para el motor de enrutamiento específico (por ejemplo, Quagga, XORP, BIRD) siempre y cuando se base en la pila de redes Linux.

Los principales componentes de la solución RouteFlow son:

- RouteFlow Client (RF-Client), anteriormente RF-Slave (RF-S)
- Servidor RouteFlow (servidor RF)
- RouteFlow Proxy (RF-Proxy), aplicación de controlador RF-C

La interacción entre los componentes se define mediante un conjunto de mensajes de protocolo RouteFlow.

Los componentes de terceros adicionales requeridos o soportados son:

- Controlador OpenFlow: NOX, POX, Floodlight, Ryu
- Motor de enrutamiento Quagga, XORP
- Open vSwitch (OVS)
- Switches habilitados para OpenFlow: Basados en software (por ejemplo, Mininet, OVS) o basados en hardware (por ejemplo, NetFPGA)

b) FLOWVISOR

FlowVisor es un controlador OpenFlow de propósito especial que actúa como un proxy transparente entre los switches OpenFlow y varios controladores OpenFlow.

FlowVisor crea buenos recursos de red y delega el control de cada segmento a un controlador diferente. Los segmentos se pueden definir por cualquier combinación de puertos de conmutación (capa 1), dirección origen o destino Ethernet (capa 2), dirección de origen o destino IP (capa 3) y puerto de origen o destino TCP/UDP o código ICMP (capa 4).

FlowVisor impone el aislamiento entre cada segmento, es decir, una porción no puede controlar el tráfico de otro.

c) RESONANCE

Resonance es una plataforma de control de SDN que aboga por el control de red impulsado por eventos. Resonance facilita la gestión de las redes al automatizar muchas tareas que se realizan actualmente modificando manualmente varios archivos de configuración de dispositivos distribuidos frente a varios eventos de red, donde los archivos de configuración se expresan en comandos CLI de bajo nivel y específicos del proveedor.

d) Kinetic

Kinetic es un lenguaje específico de dominio (DSL) y un sistema de control de red SDN que permite a los operadores expresar políticas de red dinámicas de forma concisa e intuitiva. Usando Kinetic, los programadores pueden implementar un programa de control de toda la red que puede cambiar dinámicamente el comportamiento de la red basado en varios tipos de eventos de red.

Kinetic se basa en la parte superior de Pyretic, un lenguaje de programación SDN incrustado en Python. Además, es miembro de la familia Frenetic de lenguas SDN.

4.4.4 MISCELANEA

a) IRONFLOW

IronFlow es una integración altamente experimental de OpenFlow System. La integración tiene como objetivo compartir información relacionada con la seguridad dada por OpenFlow con otros componentes de red a través de IF-MAP.

IronFlow consiste en dos elementos:

- Una parte - el "editor" - simplemente obtiene las últimas informaciones proporcionadas por un controlador OpenFlow y las convierte en metadatos IF-MAP que finalmente se publicarán en un servidor MAP.

IronFlow actualizará las informaciones de metadatos en un intervalo que se puede configurar en el archivo de propiedades de IronFlow. En otras palabras, esto significa que el flujo de hierro siempre intenta reflejar el conocimiento actual o más reciente de un controlador OpenFlow en un servidor MAP mediante sondeo.

Los metadatos publicados por IronFlow se rellenan con los siguientes valores del controlador OpenFlow:

- Las direcciones MAC de los hosts
- Las direcciones IP de controladores, hosts e interruptores
- La descripción de los interruptores conectados
- El descubridor (los switches) de IPs y MACs de hosts

- La segunda, más experimental, - el "suscriptor" - va al revés. Se suscribirá para solicitar la investigación de un PDP en el MAPS. Si el PDP publica esos metadatos en una dirección IP, IronFlow programa una tarea de entrada de firewall para esa dirección IP en el controlador OpenFlow. Si el PDP elimina la petición para investigación de la dirección IP, IronFlow también elimina la entrada del firewall de OpenFlow.

b) FLOWSCALE

FlowScale es un proyecto para dividir y distribuir tráfico a través de múltiples puertos de conmutación física. FlowScale replica la funcionalidad en el equilibrio de carga, pero utilizando un switch *Top of Rack* (ToR) para distribuir el tráfico. El uso de software para manejar la especificación del plano de control, pero el cambio de hardware para realizar el reenvío proporciona una gran flexibilidad y permite implementaciones de bajo costo y alto rendimiento.

Características principales

- **Interfaz de usuario web**: Se utiliza un webUI para administrar los conmutadores ToR conectados a FlowScale y para administrar cómo se divide el tráfico. Y además proporciona una visión de las estadísticas y el estado
- **Configuración de políticas**: el controlador OpenFlow envía reglas basadas en la configuración de políticas basadas en una dirección IP, un tipo de Ethernet o un puerto TCP / UDP
- **OpenRlow capaz ToR switch**: La capacidad de reemplazar equipos costosos con un ToR switch.

- **Multi-Platform**: FlowScale está basado en Java y puede ejecutarse en múltiples plataformas. Se ha probado en diferentes distribuciones de Linux, así como Mac OS X.
- **HotSwapping**: Capacidad de FlowScale para descargar dinámicamente el tráfico pesado desde los puertos de salida de alta carga a los de menor carga.
- **Failover**: El tráfico saliente de un conmutador se distribuirá entre otros puertos en el switch si ese puerto particular se desconecta.
- **Traffic Mirroring**: FlowScale permite que un ToR refleje el tráfico duplicado dirigido a los puertos de salida.
- **API REST:** Una interfaz API REST está disponible para permitir a los usuarios administrar directamente FlowScale

c) MININET

Mininet es un emulador de red, o quizás más precisamente un sistema de orquestación de emulación de red. Ejecuta una colección de hosts finales, switches, enrutadores y enlaces en un solo núcleo de Linux. Utiliza una virtualización ligera para hacer que un único sistema se vea como una red completa, ejecutando el mismo núcleo, sistema y código de usuario. Un host de Mininet se comporta como una máquina real; que puede ejecutar SSH en él y ejecutar programas arbitrarios (incluyendo cualquier cosa que se instala en el sistema Linux subyacente.) Los programas que ejecuta pueden enviar paquetes a través de lo que parece una verdadera interfaz Ethernet, con una velocidad de enlace dada y un retardo. Los paquetes se procesan por lo que parece un switch Ethernet real, enrutador o *middlebox*[2]. Cuando dos programas, como un cliente y un servidor iperf, se comunican a través de Mininet, el rendimiento medido debe coincidir con el de dos máquinas (más lentas).

d) NICE-OF

NICE es una herramienta para probar la aplicación del controlador OpenFlow para la plataforma del controlador NOX a través de una combinación de comprobación de modelos y ejecución simbólica.

La aparición de switches compatibles con OpenFlow permite una nueva funcionalidad de red, con el riesgo de errores de programación que hacen que la comunicación sea menos confiable. El modelo de programación centralizado, en el que un único programa controlador gestiona la red, parece reducir la probabilidad de errores. Sin embargo, el sistema está inherentemente distribuido y es asíncrono, con eventos que

[2] Middelbox: dispositivo de red que trasforma, inspecciona, filtra o manipula el tráfico para propósitos que no sean el reenvío.

ocurren en diferentes conmutadores y hosts finales, y retrasos inevitables que afectan la comunicación con el controlador. En este trabajo se presentan técnicas eficientes y sistemáticas para probar programas de control no modificados.

La herramienta NICE aplica la comprobación de modelos para explorar el espacio de estado de todo el sistema: el controlador, los conmutadores y los hosts. La escalabilidad es el principal desafío, dada la diversidad de paquetes de datos, el estado del gran sistema y los muchos posibles pedidos de eventos. Para abordar esto, se propone una forma novedosa de aumentar la comprobación de modelos con la ejecución simbólica de manejadores de eventos (para identificar paquetes representativos que ejercen rutas de código en el controlador). El prototipo prueba las aplicaciones Python en la popular plataforma NOX.

e) MIRAGE OS

MirageOS es un sistema operativo de biblioteca que construye núcleos únicos para aplicaciones de red seguras y de alto rendimiento en una gran variedad de plataformas móviles y de computación en nube. El código se puede desarrollar en un sistema operativo normal como Linux o MacOS X y, a continuación, se compila en un núcleo único completamente independiente, que se ejecuta bajo un hipervisor Xen o KVM.

Esto permite que sus servicios funcionen de manera más eficiente, segura y con un control más preciso que con una pila de software convencional completa.

MirageOS utiliza el lenguaje OCaml, con bibliotecas que proporcionan soporte de red, almacenamiento y concurrencia que funcionan bajo Unix durante el desarrollo, pero que se convierten en controladores del sistema operativo cuando se compilan para el despliegue de producción. El framework está totalmente dirigido a eventos, sin soporte para el enrutamiento preventivo.

f) WAKAME

Wakame-vdc es un centro de datos virtual, lo que hace es virtualizar un centro de datos completo. Por otro lado, Wakame-vdc es un marco de computación en la nube IaaS (Infrastructure as a Service). Básicamente, puede configurarlo y ofrecer infraestructura de servidores "en la nube". Los usuarios pueden controlar Wakame-vdc a través de un navegador web. Esto significa que, por ejemplo, puede configurar Wakame-vdc y permitir a los usuarios alquilar servidores virtuales. Algunas de las empresas que actualmente utilizan Wakame-vdc hacen precisamente eso.

Wakame-vdc es un software gratuito de código abierto. La fuente está disponible gratuitamente en Github y utiliza el lenguaje Ruby para su desarrollo.

5. CÓMO TRABAJA SDN

En esta sección se ofrece una visión general de cómo SDN trabaja, que se puede ver en la Figura 4 que se encuentra más abajo, además de los componentes básicos que componen SDN y cómo interactúan entre sí. Primero nos centraremos en los métodos utilizados por Open SDN. Luego cómo funcionan, las formas alternativas de SDN y después cómo SDN se ha ganado a algunos vendedores de redes, que han respondido con alternativas de SDN. Además, se pueden agrupar las implementaciones alternativas de SDN en dos categorías, SDN a través de API existentes y SDN a través de un híper-visor (o *hypervisor*), que es una plataforma que permite aplicar diversas técnicas de control de virtualización para utilizar, al mismo tiempo, diferentes sistemas operativos en una misma computadora basada en redes superpuestas.

5.1 LAS CARACTERÍSTICAS FUNDAMENTALES DE SDN

Como se ha comentado en el estado del arte, SDN evolucionó a partir de propuestas, normas e implementaciones tales como ForCES, 4D y Ethane y que caracterizan a SDN con cinco factores fundamentales: Separación de los planos, dispositivos simplificados, control centralizado, automatización de la red y la virtualización y apertura.

5.1.1 SEPARACIÓN DE PLANOS

La primera característica fundamental de SDN es la separación de los planos de reenvío y control. La funcionalidad de reenvío, incluye la lógica y las tablas para elegir cómo tratar a los paquetes entrantes basados en características como la MAC, dirección IP e ID de la VLAN que reside en el plano de reenvío.

Las acciones fundamentales realizadas por el plano de reenvío se pueden describir por la forma en que reparte los paquetes que llegan. Puede enviar, descartar, usar o replicar un paquete entrante. Para un reenvío básico el dispositivo determina el puerto de salida correcto realizando una búsqueda en la tabla de direcciones en el hardware de circuitos integrados de aplicaciones específicas ASIC (del inglés *Application-specific integrated circuit*). Un paquete se puede perder debido al desbordamiento de un búfer o a un filtro que es el resultado de una función de limitación de la tasa de QoS, por ejemplo. Los paquetes especiales que requieren un procesamiento por los planos de control o de gestión se pasan al plano apropiado. Por último, un caso especial de reenvío pertenece a la multidifusión, donde un paquete entrante debe ser replicado antes de reenviar las diferentes copias a los puertos de salida.

Los protocolos, la lógica y los algoritmos que se han usado para programar el plano de reenvío residen en el plano de control. Muchos de estos protocolos y algoritmos requieren un conocimiento global de la red. El plano de control determina cómo se deben programar o configurar las tablas de reenvío y la lógica en el plano de datos. Dado que en una red tradicional cada dispositivo tiene su propio plano de control, la tarea principal del plano de control es ejecutar protocolos de enrutamiento o conmutación, para que todas las tablas de reenvío distribuidas en los dispositivos a lo largo de la red permanezcan sincronizadas para la prevención de bucles, por ejemplo. En SDN el plano de control es desplazado del dispositivo hacia un controlador externo.

5.1.2 DISPOSITIVO SIMPLE Y CONTROL CENTRALIZADO

Sobre la base de la separación de los planos de reenvío y control, la siguiente característica es la simplificación de los dispositivos, que luego son controlados por un sistema centralizado donde corre un software de gestión y control. En lugar de cientos de miles de líneas de un complicado software del plano de control corriendo en el dispositivo y permitiendo que se comporte de forma autónoma, este software se elimina del equipo y en su lugar se pone un controlador centralizado. Este controlador basado en software gestiona la red utilizando políticas de nivel superior. El controlador proporciona instrucciones primitivas a los dispositivos simplificados cuando se les pueda permitir tomar decisiones rápidas sobre cómo manejar los paquetes entrantes.

5.1.3 LA AUTOMATIZACIÓN Y VIRTUALIZACIÓN DE LAS REDES

SDN puede ser derivado precisamente de las abstracciones del estado distribuido, el reenvío y la configuración. Estos han sido derivados de la descomposición, en abstracciones simples de un problema complejo del control de la red. La abstracción del estado distribuido proporciona al programador de red una vista global de la red actual, que en realidad se compone de muchas máquinas, cada una con su propio estado, y que están colaborando para resolver problemas de toda la red.

La abstracción de reenvío permite al programador especificar los comportamientos de reenvío necesarios sin ningún conocimiento del hardware específico del proveedor. Esto implica que cualquier lenguaje o los lenguajes que surjan de la abstracción necesitan representar un pequeño denominador común de las capacidades de reenvío del hardware de la red. Finalmente, la abstracción de configuración debe ser capaz de expresar los objetivos deseados de la red global sin perderse en los detalles de cómo está implementada la red física. Para los desarrolladores trabajar con la red a través de esta configuración, viene a ser realmente la virtualización de la red en su nivel más básico. En este tipo de virtualización está el corazón de cómo se define Open SDN [24].

El controlador centralizado basado en software SDN proporciona una interfaz abierta en el controlador para permitir el control automatizado de la red. En el contexto de Open SDN, los términos norte se usan para describir las aplicaciones y sur para los dispositivos.

La API sur es la interfaz de OpenFlow, que el controlador utiliza para programar los dispositivos de red. El controlador ofrece una API que permite que las aplicaciones de software se conecten al controlador permitiendo de ese modo que el software nos proporcione los algoritmos y protocolos que pueden hacer funcionar a la red eficientemente. Estas aplicaciones pueden hacer rápida y dinámicamente cambios en la red según lo necesite.

5.1.4 CARÁCTER ABIERTO

Una característica de Open SDN es que sus interfaces deben permanecer como un estándar, bien documentadas y sin propietario. Las API que se definen deben proporcionar al software suficiente control para experimentar y para controlar las opciones del plano de control. La premisa es mantener abiertas las interfaces del SDN para permitir la investigación de nuevos e innovadores métodos de funcionamiento de la red. Tanto las instituciones de investigación como los empresarios pueden aprovechar esta capacidad para experimentar con facilidad y probar nuevas ideas. De ahí la velocidad con la que la tecnología de red se ha desarrollado y desplegado en gran medida gracias a un gran número de individuos y organizaciones que son capaces de resolver los problemas, dejando como resultado mejores y más rápidos avances en la estructura y el funcionamiento de las redes.

La presencia de estas interfaces abiertas alienta a promover proyectos de código abierto relacionados con SDN. El poder del código abierto debe acelerar en gran medida la innovación en SDN [20]. Además de facilitar la investigación y la experimentación, las interfaces abiertas permiten que diferentes proveedores puedan interoperar. Esto normalmente produce un entorno competitivo con bajos costes para los consumidores de equipos de red. Esta reducción ha sido parte de SDN desde su creación.

5.2 COMPORTAMIENTO DE SDN

A nivel conceptual, el comportamiento y funcionamiento de una red definida por software es sencillo. En la Figura 4 se proporciona una representación gráfica de las operaciones de los componentes básicos de SDN: los dispositivos SDN, el controlador y las aplicaciones. La forma más fácil de entender cómo funcionan es mirar de abajo a arriba, comenzando con el dispositivo SDN. Como se muestra en la Figura 4, los dispositivos SDN

contienen la funcionalidad de reenvío para decidir qué hacer con cada paquete entrante. Los dispositivos también contienen los datos que impulsan esas decisiones de reenvío. Los datos en sí que se representan en realidad son los flujos definidos por el controlador, como se representa en la parte superior izquierda de cada dispositivo.

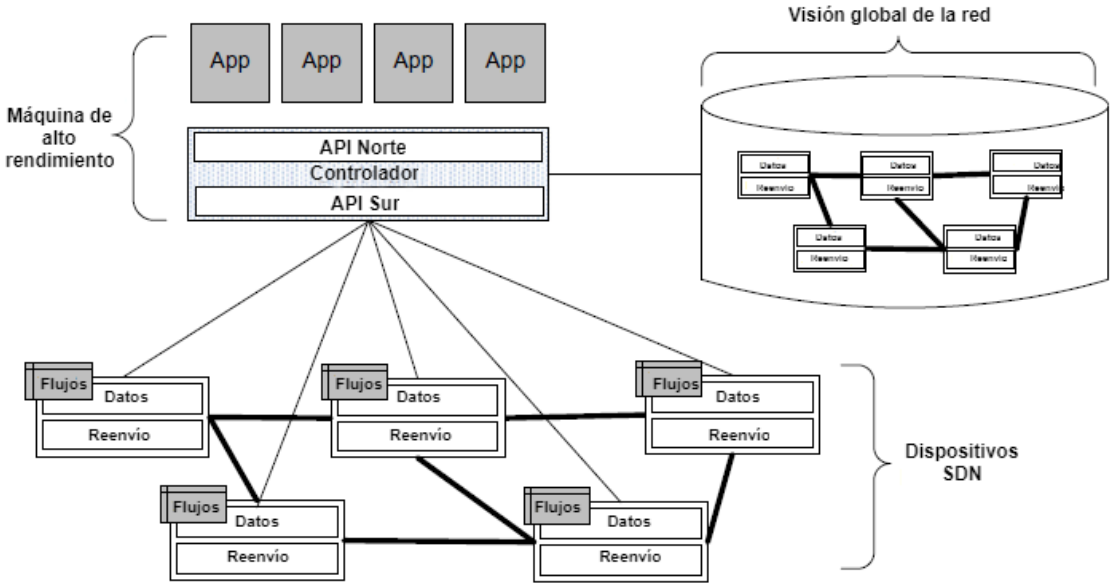


Figura 4. Vista general de las operaciones SDN

Un flujo describe un conjunto de paquetes transferidos desde un extremo de la red a otro o conjunto de puntos finales. Los puntos finales pueden definirse como direcciones IP y pares de puertos TCP/UDP, puntos finales de VLAN, puntos finales de túneles de la capa tres y puertos de entrada entre otras cosas. Un flujo unidireccional es aquel en el cual, los paquetes fluyen entre los mismos puntos finales en direcciones opuestas, pudiendo constituir un flujo separado. Los flujos se representan en un dispositivo como una entrada de flujo.

Una tabla de flujo reside en un dispositivo de red y consiste en una serie de entradas de flujo y acciones para realizar. Cuando un dispositivo SDN recibe un paquete, este consulta sus tablas de flujo en busca de una coincidencia. Estas tablas de flujo han sido construidas previamente cuando el controlador descarga unas reglas de flujo apropiadas para el dispositivo. Si un dispositivo SDN encuentra una coincidencia, coge la acción de configuración apropiada, que normalmente implica el reenvío del paquete. Si no encuentra una coincidencia, el switch puede o borrar el paquete o pasarlo al controlador, dependiendo de la versión de OpenFlow y la configuración del switch.

La definición de un flujo es una expresión de programación relativamente simple que puede ser un cálculo del plano de control muy complejo previamente realizado por el controlador. En la arquitectura de un switch tradicional, es importante comprender que la

complejidad de la conmutación es tal, que simplemente no se puede realizar a grandes velocidades de línea y en su lugar debe ser asimilado por el plano de control y reducido a reglas simples que se puedan procesar a esa velocidad. En Open SDN la forma de asumir esto es con las entradas de flujo.

El controlador SDN, es responsable de la abstracción de la red de dispositivos SDN y controla y presenta una abstracción de estos recursos de red para las aplicaciones SDN que se ejecutan. El controlador permite a la aplicación SDN definir los flujos en los dispositivos y ayudar a la aplicación a responder a paquetes que son enviados al controlador por los dispositivos SDN. En la Figura 4 se puede ver en el lado derecho del controlador que mantiene una visión completa de la red que está controlando. Esto permite calcular una solución óptima de reenvío de manera determinista y predecible. Dado que un controlador puede controlar un gran número de dispositivos de red, estos cálculos se realizan normalmente en un alto nivel de rendimiento de la máquina con una ventaja de rendimiento en la CPU y con una capacidad de memoria que no es típica de estos dispositivos de red. Por ejemplo, un controlador podría ser implementado en una CPU de 8 núcleos de 2 GHz frente a una CPU de 1 solo núcleo y 1 GHz que es más típica en un switch.

Las aplicaciones SDN se construyen en la parte superior del controlador. Esas aplicaciones no deben confundirse con la capa de aplicación definida en la capa siete del modelo OSI. La aplicación SDN se interconecta con el controlador, que se usa para establecer flujos proactivos en los dispositivos y recibir paquetes que han sido reenviados por el controlador. Los flujos proactivos son establecidos por la aplicación, que normalmente establecerá estos flujos cuando la aplicación se inicie, y los flujos persisten hasta que la configuración cambie.

Además de los flujos proactivos definidos por la aplicación, muchos flujos son definidos en respuesta a un paquete reenviado al controlador. Al recibir los paquetes entrantes que han sido reenviados al controlador, la aplicación SDN instruirá al controlador sobre cómo debe responder al paquete y, si es apropiado, establecerá nuevos flujos en los dispositivos para que permitan que el dispositivo pueda responder localmente las siguientes veces que el paquete pertenezca a ese flujo. Estos flujos se llaman flujos relativos, de esta manera, es posible escribir aplicaciones de software que implementen reenvío, enrutamiento, multitrayecto y funciones de control de acceso entre otras.

También hay flujos reactivos que se definen o modifican como resultado de un estímulo de las fuentes como los paquetes del controlador. Por ejemplo, el controlador puede insertar flujos relativos en respuesta de fuentes de datos como los sistemas de detección de intrusiones (IDS) [25] o el analizador de tráfico NetFlow [26]. La Figura 5 representa el protocolo OpenFlow como medio de comunicación entre el controlador y el

dispositivo. Aunque OpenFlow es el estándar definido para la comunicación en Open SDN, hay soluciones SDN alternativas.

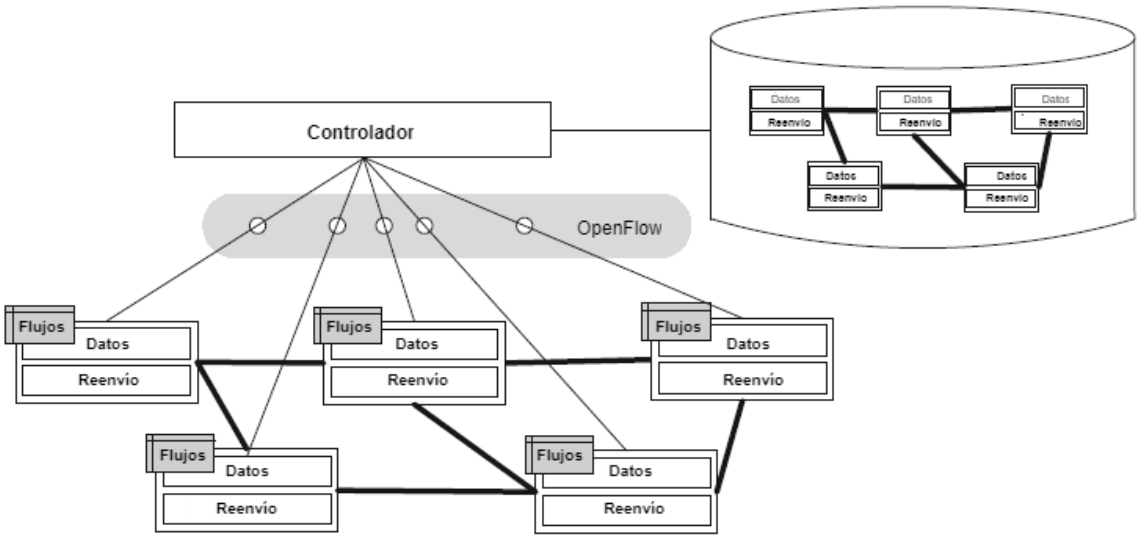


Figura 5. Comunicación de controlador a dispositivo

5.3 DISPOSITIVOS SDN

Un dispositivo SDN está compuesto por una API para la comunicación con el controlador, una capa de abstracción y una función de procesamiento de paquetes. En el caso de un switch virtual, esa función de procesamiento de paquetes es un software de procesamiento de paquetes, como se ve en la Figura 6. En el caso de un switch físico, la función de procesamiento de paquetes está incorporada en la lógica de procesamiento de paquetes, como se muestra en la Figura 7.

La capa de abstracción incorpora una o más tablas de flujo. La lógica de procesamiento de paquetes consiste en mecanismos que toman decisiones basadas en los resultados de evaluar los paquetes entrantes y encontrar la coincidencia de máxima prioridad. Cuando se encuentra una coincidencia, los paquetes entrantes son procesados localmente a menos que tenga que ser reenviado al controlador. Cuando no hay coincidencia, el paquete puede ser copiado en el controlador para procesarlo después. A este proceso también se le conoce como el controlador que consume paquetes.

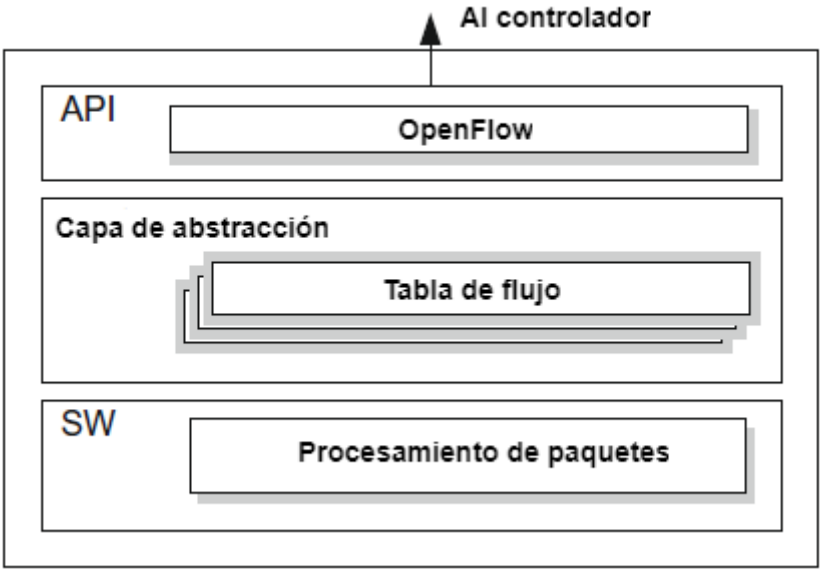


Figura 6. Anatomía del software SDN del switch

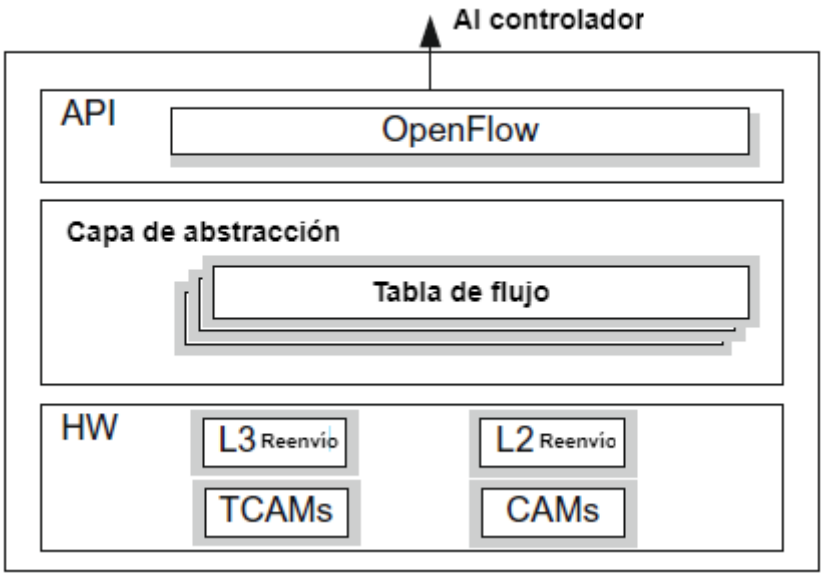


Figura 7. Anatomía del hardware SDN del switch

5.3.1 TABLAS DE FLUJO

Las tablas de flujo son las estructuras fundamentales de datos en un dispositivo SDN. Estas tablas de flujo permiten al dispositivo evaluar los paquetes entrantes y tomar la decisión adecuada en base al contenido del paquete que se acaba de recibir. Los paquetes han sido tradicionalmente recibidos por dispositivos de red y evaluados en base a ciertos campos. Dependiendo de la evaluación, se toman medidas que incluyen el reenvío de un paquete a un puerto específico, su eliminación o la inundación de paquetes por todos los puertos entre otros. Un dispositivo SDN no es fundamentalmente diferente excepto que la

operación básica ha sido hecha más genérica y más programable a través de las tablas de flujo y su lógica asociada.

Las tablas de flujo consisten en un número de entradas de flujo priorizadas, las cuales constan de dos componentes: los campos de coincidencia y las acciones. Los campos de coincidencia se utilizan para comparar de nuevo los paquetes entrantes. En un paquete entrante se comparan los campos de coincidencia por orden de prioridad y la primera coincidencia completa hace que se seleccione. Las acciones son instrucciones que el dispositivo de red debe realizar si un paquete entrante coincide con los campos de coincidencia especificados por esa entrada de flujo.

Los campos de coincidencia pueden tener campos irrelevantes en una coincidencia en particular. Por ejemplo, cuando se emparejan paquetes basados solo en la dirección IP o subred, todos los demás campos serían irrelevantes. Dependiendo de los que la aplicación necesite, todos los campos pueden ser importantes y en ese caso no habría campos irrelevantes. La tabla de flujo y las entradas de flujo construyen que la aplicación SDN desarrolle una amplia gama de posibilidades para emparejar paquetes y tomar las decisiones adecuadas.

5.3.2 SWITCHES DE SOFTWARE SDN

La implementación de dispositivos SDN en software es el medio más sencillo de crear un dispositivo SDN, porque las tablas de flujo, sus entradas y los campos de emparejamiento implicados se asignan fácilmente a las estructuras de datos de software generales, tales como matrices ordenadas y tablas de hash. En consecuencia, es más probable que dos dispositivos de software SDN desarrollados por diferentes equipos de desarrolladores, se comporten de manera más consistente que dos implementados en hardware. Por el contrario, las implementaciones en software probablemente serán más lentas y menos eficientes que las implementadas en hardware, ya que no se benefician de ninguna aceleración del hardware. Por consiguiente, para dispositivos de red que deben funcionar a velocidades altas como 10, 40 y 100 Gbps, solo son factibles en implementaciones hardware.

Debido al uso de campos irrelevantes en el emparejamiento, supone un problema para las típicas tablas de hash, ya que el procesamiento de paquetes utiliza una sofisticada lógica de software para implementar eficientes emparejamientos en las búsquedas de los campos. Por tanto, en los primeros días de SDN, hubo una amplia variación en el rendimiento de las diferentes implementaciones de software, que estaban basadas en la eficiencia con la que se realizaban estas búsquedas. Afortunadamente, las implementaciones de dispositivos SDN de software han madurado, ya que hay métodos

eficientes de realizar estas búsquedas, que además han dado mayor uniformidad en el rendimiento del software del dispositivo SDN.

Las implementaciones de los dispositivos de software también sufren menos limitaciones de recursos, ya que las consideraciones tales como la potencia de procesamiento y el tamaño de la memoria no son un problema en este tipo de implementaciones. Así, la implementación de un dispositivo hardware soportará solo un número limitado de entradas de flujo. Como las implementaciones de dispositivos software tiene más flexibilidad para implementar acciones más complejas, es de esperar un conjunto de acciones disponibles más grande en los dispositivos software que hardware.

5.3.3 DISPOSITIVOS HARDWARE SDN

Las implementaciones de dispositivos hardware tienen la promesa de operar mucho más rápido que los implementados en software, y por tanto, son más aplicables a entornos de rendimiento sensible, como pueden ser centros de datos o núcleos de red. Actualmente, los dispositivos de red utilizan hardware especializado diseñado para facilitar la inspección de los paquetes entrantes o para las decisiones que debe seguir basadas en el emparejamiento de paquetes. En la Figura 7 se ve que la lógica de procesamiento ha sido reemplazada en la Figura 6 por hardware especializado. Este hardware incluye las tablas de reenvío de las capas dos y tres, que son normalmente implementadas usando memorias de contenido direccionable (CAMs) [27] y memorias ternarias de contenido direccionable (TCAMs) [28].

La tabla de reenvío de la capa tres se usa para tomar decisiones de enrutamiento a nivel IP. Esta es la acción fundamental de un router, hace coincidir la dirección IP de destino con las entradas de la tabla y en base a la coincidencia, toma la opción de encaminamiento más adecuada. La tabla de reenvío de la capa dos se usa para tomar decisiones de reenvío a nivel MAC. Esta es la operación fundamental de un switch, hace coincidir la MAC de destino con las entradas de la tabla y toma la opción de reenvío adecuada en base a la coincidencia.

La tabla de reenvío de la capa dos se implementa normalmente usando una CAM regular o hardware basado en hashing. Este tipo de memorias asociativas son usadas cuando existen índices precisos. Las TCAMs, sin embargo, son asociativas con funciones de emparejamiento más complejas. Las TCAMs son utilizadas en el hardware para comprobar no solo una coincidencia exacta, sino también para el tercer estado, que usa una máscara para tratar ciertas partes de los campos irrelevantes de los paquetes. Las TCAM son esenciales para las funciones como el enrutamiento basado en políticas.

5.4 EL CONTROLADOR SDN

En la Figura 8 se puede ver los módulos que proporcionan la funcionalidad principal al controlador, tanto la API del norte como la del sur y unas pocas aplicaciones de ejemplo que podría utilizar el controlador. La API del sur se utiliza para la interfaz con los dispositivos SDN. Hay que destacar que, en algunos productos, OpenFlow y otras alternativas coexisten en el mismo controlador. OpenFlow es el mejor ejemplo de estabilidad, pero los estándares como CLI de Cisco y SNMP también representan la interfaz del sur. El protocolo complementario de OpenFlow, *OF-Config* [21] y *Open vSwitch Database Management Protocol* (OVSDB) [22] son protocolos abiertos para la interfaz del sur, aunque se limitan a funciones de configuración.

Se considera una deficiencia de SDN que en la actualidad no haya un estándar para la interfaz aplicación-controlador y para subsanar este problema algunos organismos están elaborando propuestas para normalizarla. A pesar de la ausencia de una norma, las interfaces del norte se han implementado de diferentes formas. La API del norte representa una increíble oportunidad para la innovación y la colaboración con un montón de proveedores y la comunidad del código abierto.

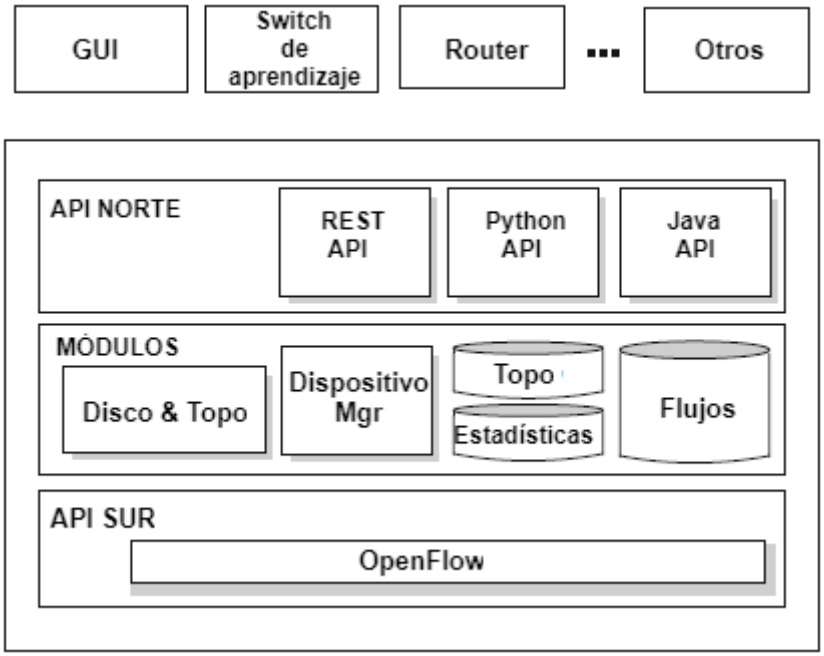


Figura 8. Anatomía del controlador SDN

5.4.1 MÓDULOS DEL NÚCLEO DEL CONTROLADOR SDN

El controlador abstrae los detalles del protocolo que controla al dispositivo SDN para que las aplicaciones sean capaces de comunicarse con los dispositivos SDN. La Figura 8 muestra la API debajo del controlador, que es OpenFlow en Open SDN, y la interfaz

proporcionada para las aplicaciones. Muchos de los controladores proveen funcionalidad de núcleo entre estas interfaces sin procesar. En el controlador las funciones del núcleo incluyen:

- **Descubrimiento de dispositivos finales.**
- **Detección de dispositivos de red como pueden ser routers, switches…**
- **Gestión de la topología del dispositivo de red**, manteniendo información sobre los detalles de la interconexión de los dispositivos.
- **Gestión del flujo**, manteniendo una base de datos de los flujos gestionados por el controlador y realizando la coordinación necesaria para asegurar la sincronización de las entradas de flujo del dispositivo con esa base de datos.

Las funciones principales del controlador son el descubrimiento de la topología de los dispositivos y su seguimiento, la gestión de flujo, la gestión de dispositivos y el seguimiento de estadísticas. Todo esto es implementado por un conjunto de módulos internos en el controlador. Como se muestra en la Figura 8, estos módulos necesitan mantener bases de datos locales que contengan la topología actual y las estadísticas. El controlador busca en la topología y aprende de la existencia de dispositivos SDN, de usuario final y el seguimiento de conectividad entre ellos. Mantiene una caché de flujo que refleja las tablas de flujo en los distintos switches que controla. El controlador mantiene localmente estadísticas de flujo que ha recopilado de sus switches. El controlador puede diseñar que las funciones sean implementadas a través de módulos, de tal manera que el conjunto de características del controlador puede ser adaptado a los requisitos de una red en particular.

5.4.2 INTERFACES DE CONTROL SDN

Para las aplicaciones SDN, una función clave proporcionada por el controlador SDN es la API para acceder a la red. En algunos casos, esta API norte es una interfaz de bajo nivel que proporciona acceso a la red de dispositivos de manera común y consistente.

La Figura 9 examina de cerca cómo el controlador interactúa con las aplicaciones. El controlador informa a la aplicación de los eventos que ocurren en la red. Los eventos pueden referirse a un paquete individual que ha sido recibido por el controlador o algún estado que ha cambiado en la topología de la red. Las aplicaciones utilizan diferentes métodos que pueden afectar al funcionamiento de la red. Tales métodos pueden ser invocados en respuesta a un evento recibido y puede también puede ser resultado de un paquete recibido que ha sido borrado, modificado o reenviado o también la eliminación, adición o modificación de un flujo.

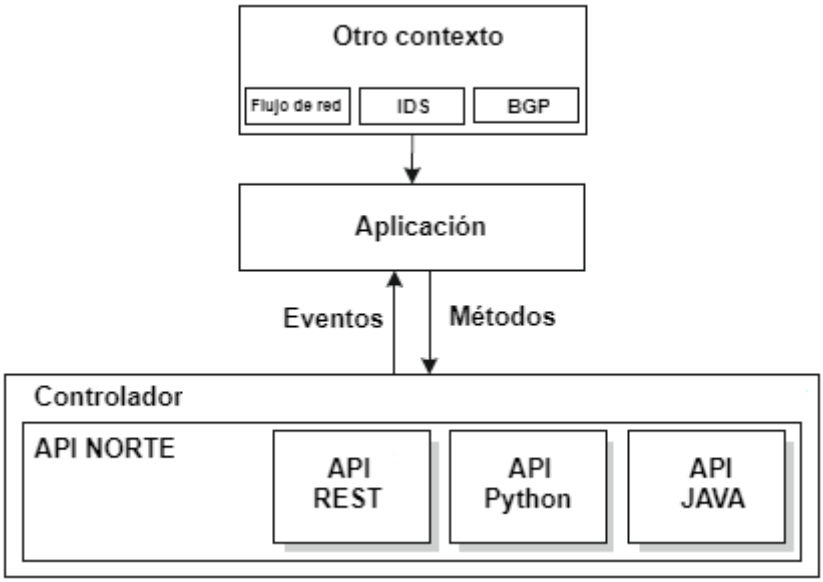


Figura 9. API norte del controlador SDN

Puede haber más de una aplicación SDN ejecutándose en un solo controlador. Cuando este es el caso, las cuestiones relacionadas con la priorización de las aplicaciones y el manejo del flujo son importantes. La falta de un estándar emergente para la API norte está obstaculizando los esfuerzos para desarrollar aplicaciones que se podrán reutilizar en una amplia gama de controladores.

Los flujos de un dispositivo SDN se procesan en orden de prioridad. El primer flujo que coincide con el paquete entrante se procesa. Dentro de una aplicación SDN, es muy importante que los flujos en el dispositivo SDN sean priorizados correctamente. Sin embargo, cuando hay múltiples aplicaciones SDN, la priorización de entrada de flujo se vuelve más difícil de gestionar.

5.5 LAS APLICACIONES SDN

Las aplicaciones SDN se ejecutan por encima del controlador SDN, conectándose a la red a través de la API del controlador norte. Las aplicaciones SDN son responsables en último lugar de la gestión de las entradas de flujo que se programan en los dispositivos de red que utilizan la API del controlador para gestionar los flujos. A través de esta API, las aplicaciones son capaces de configurar los flujos para encaminar paquetes a través de la mejor ruta entre dos puntos finales, equilibrar las cargas de tráfico a través de múltiples rutas o destinadas a un conjunto de puntos finales, reaccionar ante cambios en la topología de la red como fallos de enlaces y la adición de nuevos dispositivos y rutas y por último redirigir el tráfico para fines como inspección, autenticación, segregación y tareas similares

relacionadas con la seguridad. De esta manera, protocolos como *Border Gateway Protocol* (BGP) quedarían sustituidos por estas aplicaciones SDN.

La Figura 8 incluye algunas aplicaciones estándar, como una interfaz gráfica de usuario (GUI) para manejar el controlador, un switch de aprendizaje y una aplicación de enrutamiento. Hay que entender que una simple funcionalidad de la capa 2 en switches de aprendizaje no se obtiene simplemente emparejando un dispositivo SDN con un controlador SDN. La lógica adicional es necesaria para reaccionar ante nuevas direcciones MAC y actualizar las tablas de reenvío en los dispositivos SDN que se controlan de tal manera que proporcionan conectividad a las nuevas direcciones MAC, evitando así bucles de conmutación. Una de las fortalezas de la arquitectura SDN es el hecho de que las decisiones de conmutación pueden ser controladas por una familia, cada vez más rica, de aplicaciones que controlan el controlador. De esta manera, la potencia de la arquitectura SDN es altamente expansible.

5.6 SDN A TRAVÉS DE LAS API EXISTENTES

Si un concepto básico de SDN es mover la funcionalidad de control de los dispositivos a un controlador centralizado, esto se puede lograr de otras maneras, como con el enfoque centralizado en OpenFlow acoplado a Open SDN. En particular, un método consiste en proporcionar un conjunto más rico de puntos de control en los dispositivos, de modo que un software centralizado pueda manipular esos dispositivos y proporcionar el comportamiento inteligente y predecible, que se espera en una red controlada por SDN.

En la Figura 10 se describe SDN a través de las APIs. El diagrama muestra un controlador que se comunica con dispositivos a través de una API propietaria. A menudo los proveedores a través de las soluciones API existentes, pueden proporcionar un nivel mejorado de la API en sus dispositivos, en lugar de sólo la tradicional CLI y SNMP. En la Figura 10, hay un conjunto de aplicaciones que utilizan un controlador centralizado que afecta en el reenvío en la red física.

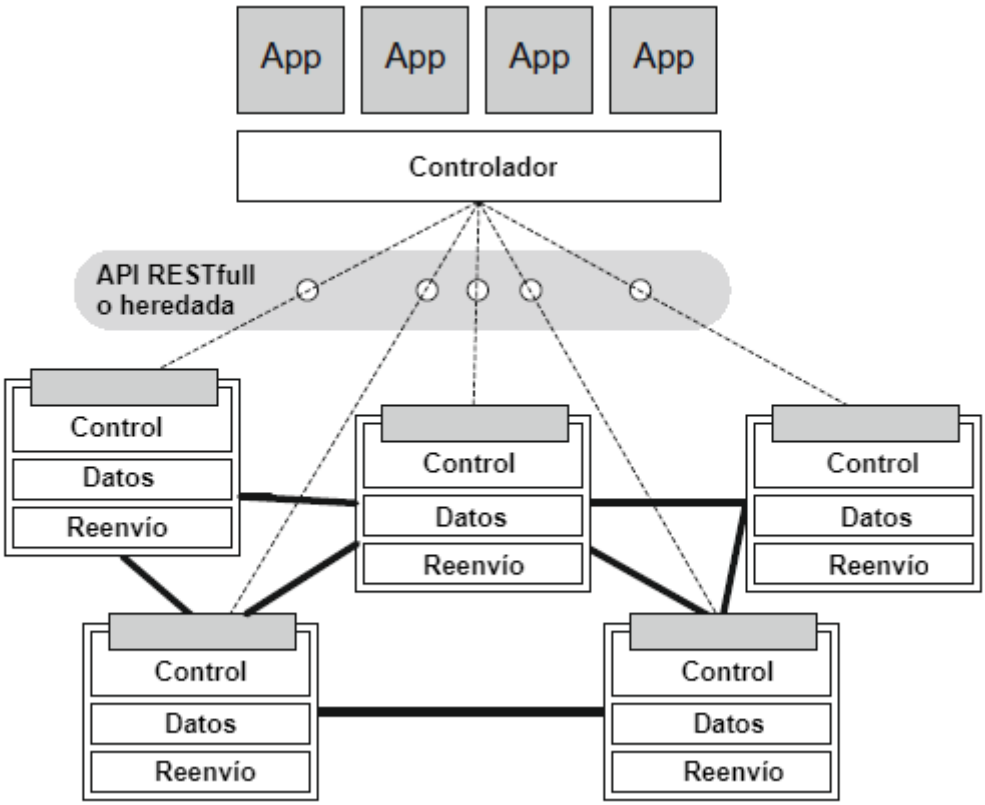


Figura 10. SDN existiendo a través de las APIs

Se han desarrollado nuevos métodos para proporcionar los medios necesarios para hacer los cambios de configuración que se han desarrollado estos últimos años, como puede ser la API RESTful. El método REST se ha convertido en el método dominante de hacer llamadas API a través de las redes. Utiliza *HyperText Transfer Protocol* (HTTP), que es el protocolo comúnmente utilizado para transferir tráfico web.

Una ventaja clara de este enfoque es que permite mejoras en agilidad y automatización. Además, también es más fácil escribir software de herramientas de orquestación que pueden responder rápida y automáticamente a los cambios en la red. Otra ventaja es que estas APIs permiten un cierto control centralizado de los dispositivos en la red. Por lo tanto, es posible crear una solución SDN utilizando las APIs que proporcionan los dispositivos de red distribuidos.

Por supuesto, los métodos SDN basados en API también tienen sus limitaciones. En primer lugar, en la mayoría de los casos no hay un controlador. El programador de red necesita interactuar directamente con cada switch. Segundo, incluso cuando hay un controlador, no se proporciona una abstracción al programador. En su lugar el programador necesita pensar en términos de switches individuales. Y, por último, dado que todavía existe un plano de control operando en cada switch, el controlador y, lo que es más importante, el programador de desarrollo de las aplicaciones de la parte superior de ese controlador debe sincronizarse con lo que se está haciendo el plano de control distribuido.

El precepto de mover el control fuera del switch a un controlador común estaba en parte destinada a crear switches más simples y menos costosos. El enfoque SDN a través de las API existentes se basa en reutilizar los switches costosos y complicados de siempre.

Por último, aunque el enfoque SDN a través de las API existentes permite cierto control sobre el reenvío, en particular con las VLAN y las VPN, no se permite el mismo grado de control de flujo que el ofrecido por OpenFlow.

5.7 SDN A TRAVÉS DE REDES DE CAPAS SOBREPUESTAS BASADAS EN HYPERVISOR

Otro método alternativo más innovador es lo que se refiere como redes de superposición basadas en hypervisor [29]. Bajo este concepto, la red física actual se deja como está, con dispositivos de red y permaneciendo sus configuraciones inalteradas. Por encima de esa red, sin embargo, el hypervisor basado en redes virtuales, se construye. Los sistemas en los bordes de la red interactúan con estas redes virtuales, que esconden los detalles de la red física para los dispositivos que se conectan a las capas sobrepuestas. Esto se puede ver en la Figura 11, donde se puede observar las redes virtualizadas que se superponen a la infraestructura de la red física. Las aplicaciones SDN que hacen uso de estos recursos tienen acceso a redes y puertos virtualizados, que son de naturaleza abstracta y no necesariamente se relacionan directamente con sus partes físicas.

Como se muestra en la Figura 11, conceptualmente el tráfico de la red virtual se ejecuta por encima de la infraestructura de la red física. Los hypervisors inyectan tráfico en la red virtual y reciben tráfico de ella. El tráfico de las redes virtuales se pasa a través de los dispositivos físicos, pero los puntos finales no son conscientes de los detalles de la topología física. Ya que estas redes virtuales existen por encima de la infraestructura física, pueden ser controladas por los dispositivos del borde de la red. En un centro de datos, éstos serían normalmente los hypervisors de las máquinas virtuales (VM) que se ejecutan en cada servidor.

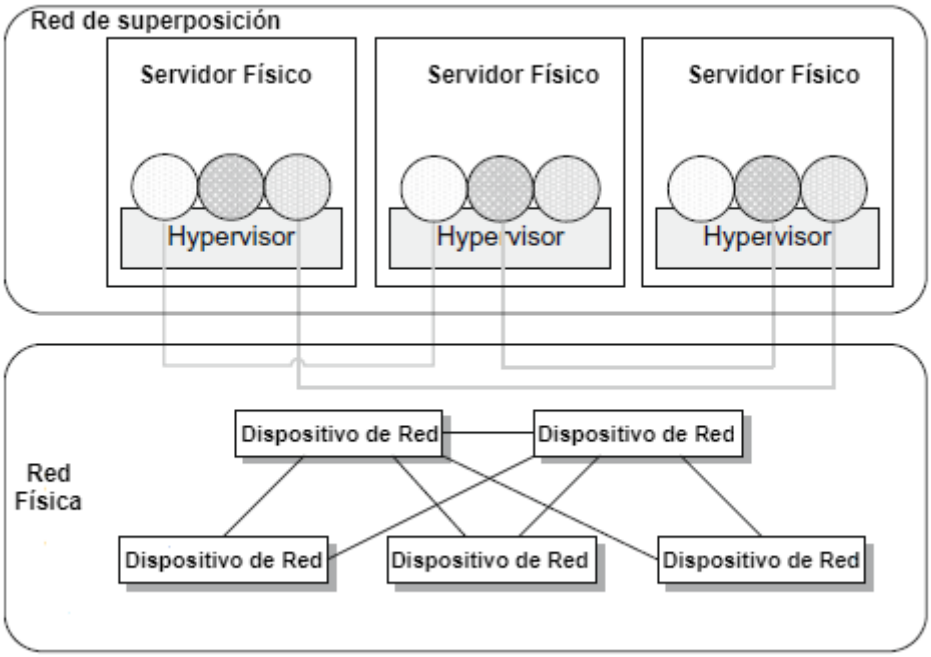


Figura 11. Redes virtualizadas

El mecanismo que hace todo esto posible es el túnel, que se utiliza en la encapsulación. Cuando un paquete entra en el borde de la red virtual de la fuente, el dispositivo de red (normalmente el hypervisor) cogerá el paquete en su totalidad y lo encapsulará dentro de otro *frame* (trama), como se puede ver en la Figura 12. Al borde de la red virtual se le llama punto final de túnel virtual o VTEP (del inglés *Virtual Tunnel Endpoint*).

El *hypervisor* entonces coge el paquete encapsulado, y basado en la información programada por el controlador, lo envía al VTEP [30] del destino. Este VTEP desencapsula el paquete y lo envía al host de destino. Normalmente, en la virtualización de la red, los VTEP están asociados con los *hypervisors*. Este mecanismo de túnel se conoce como túnel MAC en IP porque todo el *frame*(trama), desde la dirección MAC, está encapsulado dentro del frame IP unicast, como se muestra en la Figura 12. Este enfoque exige que un controlador centralizado se encargue de asegurarse que siempre hay un camino desde el actual host de destino al *hypervisor* de destino que sirve a este host.

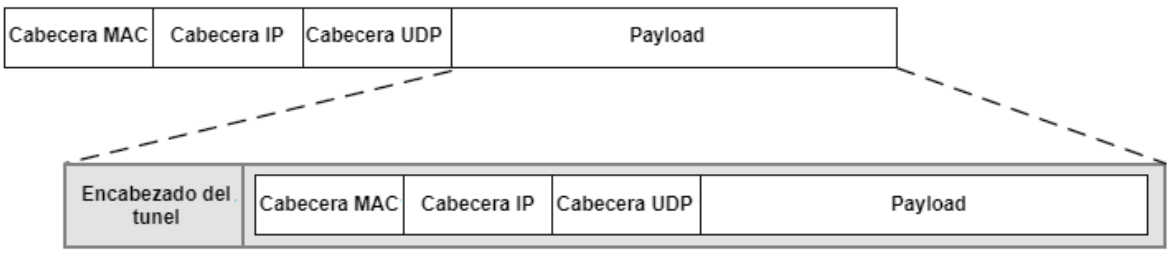


Figura 12. Tramas encapsuladas

Los roles de estos VTEPs sirven a los dispositivos host de origen y de destino. La capacidad de la red virtual suele agregarse a un *hypervisor* que la extiende con un switch virtual.

La red virtual tiene una topología virtual que consiste en switches virtuales interconectados por enlaces virtuales punto a punto. Estos se representan como los VTEPs y los enlaces virtuales son los túneles que los interconectan. Todo el tráfico en cada red virtual está encapsulado como hemos explicado anteriormente y se envía de VTEP a VTEP.

En resumen, uno de los beneficios de estas redes es, que se ocupan de las limitaciones de VLAN, ya que todo el tráfico es tunelizado y las VLANs no requieren apoyo del aislamiento de múltiples inquilinos. Además, abordan las necesidades de agilidad y automatización porque es implementado en software, y estas redes virtuales se pueden construir y desmontar en menos de una fracción del tiempo del que se requiere para cambiar la estructura de la red física.

Sin embargo, estas redes de capas sobrepuestas no resuelven todos los problemas que pueden ser abordados por un Open SDN. En particular, no se ocupan de cuestiones existentes dentro de la infraestructura física, que todavía requiere una configuración manual y mantenimiento. Tampoco abordan la priorización del tráfico ni la eficiencia de la infraestructura física, por lo que se enfrentan a enlaces bloqueados de STP y la configuración de QoS sigue siendo un desafío.

En conclusión, es importante darse cuenta de que no existe incompatibilidad en cuanto al enfoque de la red de sobre capas basado en hypervisor para SDN y Open SDN. De hecho, algunas implementaciones utilizan OpenFlow para crear y utilizar los túneles necesarios en este tipo de virtualización de red. No es irrazonable pensar que estas redes de sobrecapa son un escalón más hacia una solución SDN más completa que incluya SDN y OpenFlow para abordar las necesidades virtuales, así como las físicas de la red.

6. LA ESPECIFICACIONES DE OPENFLOW

OpenFlow y SDN no son la misma cosa. OpenFlow (OF) es considerado uno de los primeros estándares de SDN. Se definió originalmente el protocolo de comunicación en entornos SDN, que permite al controlador SDN interactuar directamente con el plano de reenvío de dispositivos de red, como conmutadores y encaminadores, tanto físicos como virtuales. Además, es un conjunto de tecnologías sustancialmente distintas incluidas bajo la misma capa de SDN. Es una herramienta importante y una fuente de innovación, además OpenFlow define tanto los protocolos de comunicación entre el plano de datos SDN y el plano de control y además la parte del comportamiento del plano de datos. Este no describe el comportamiento del controlador en sí mismo. Hay otras aproximaciones a SDN, pero hoy en día OpenFlow es el único protocolo de propósito general sin propietario para programar el siguiente plano de switches SDN. Ahora solo nos centraremos en el protocolo y comportamiento dictado por el OpenFlow, ya que no hay competencia alternativa directa.

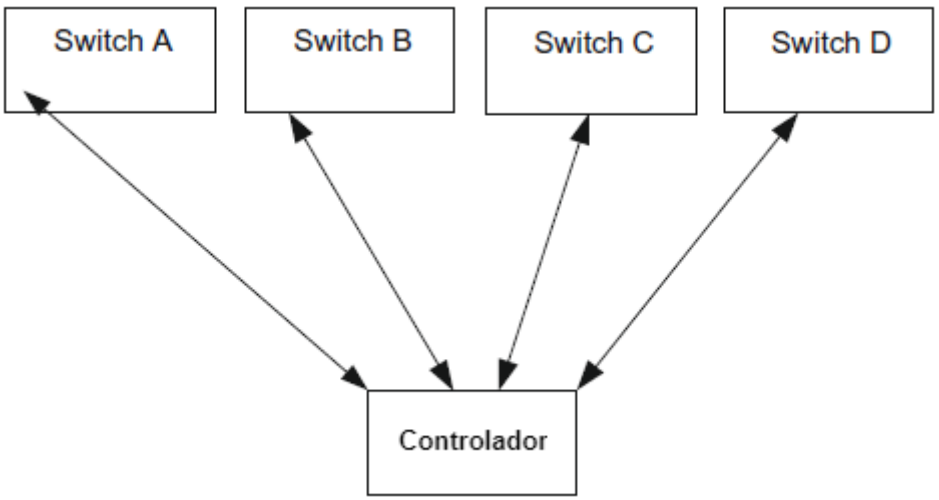


Figura 13. Componentes OpenFlow

Se pueden ver los componentes básicos de un sistema OpenFlow en la Figura 13. Hay siempre un controlador de OpenFlow que se comunica con uno o más Switches de OpenFlow. El protocolo de OpenFlow define mensajes específicos y el cambio de los formatos en los mensajes entre controladores (plano de control) y dispositivos (plano de datos). El comportamiento de OpenFlow específica como el dispositivo debe reaccionar en varias situaciones y cómo debe responder a comandos que provienen del controlador.

6.1 TERMINOLOGÍA ESPECÍFICA

Estas son las nuevas operaciones:

- Hacer pop a un objeto es quitarlo de una lista ordenada de objetos del ultimo en entrar es el primero en salir (LIFO).
- Hacer push a un objeto es añadirlo a una lista LIFO ordenada de objetos.

Estos términos se usan frecuentemente en conjunto con el término pila, que es un término de ciencias computacionales para una lista ordenada LIFO. Cuando añadimos algo a la pila hacemos “push” para ponerlo encima de la pila y luego “pop” para recuperarlo. La especificación OpenFlow incluye varias definiciones que usan este concepto. En este contexto, “push” implica añadiré un nuevo elemento de encabezado a la cabecera del paquete existente, como puede ser la etiqueta MPLS. Ya que muchas de estas etiquetas pueden ser activadas antes de que se borren, y ya que se eliminan en el orden LIFO, la analogía de la pila es apta.

6.2 LA PERSPECTIVA DE OPENFLOW

OpenFlow es un intento de permitir a los programadores, de forma genérica, desarrollar implementaciones para encaminadores que conformen un nuevo paradigma. OpenFlow intenta aprovechar los diseños impulsados por tablas existentes en muchas de las soluciones actuales. Con gran número de comerciantes que se ha consolidado, debería haber una gran posibilidad de concordancia en el futuro de las versiones de OpenFlow.

Aunque las implementaciones de puro software SDN existen, estas no pueden enrutar paquetes a una tasa lo suficientemente alta como para mantenerse como interfaces de alta velocidad. Lo que quiere decir que la palabra software SDN, entonces, es que los dispositivos SDN son totalmente programables, no que todo está hecho usando software ejecutado en un CPU tradicional.

6.2.1 EL SWITCH OPENFLOW

En la Figura 14 se describe las funciones básicas de un Switch OpenFlow V.1.0 y su relación con un controlador. Como se esperaría en un switch conmutador de paquetes, vemos que la función principal es coger los paquetes que llegan a un puerto (camino X en el puerto2 en la figura) y enviarlo a través de otro puerto (puerto N en la figura), haciendo cualquier modificación en el camino. Un aspecto del switch OpenFlow es expresado en la función de empaquetar paquetes “*packet-matching function*” mostrado en la Figura 14. La tabla adyacente es una tabla de flujo. La doble flecha gris gruesa en la Figura 14 empieza

en la decisión lógica, mostrando una coincidencia con una entrada particular en esa tabla, y direcciona la nueva igualdad del paquete a la caja de acción de la derecha. Esta caja de acción tiene tres opciones fundamentales para la disposición del paquete entrante.

A. Enviar el paquete a un puerto local, modificando posiblemente ciertos campos de cabecera primero.

B. Eliminar el paquete.

C. Pasar el paquete al controlador.

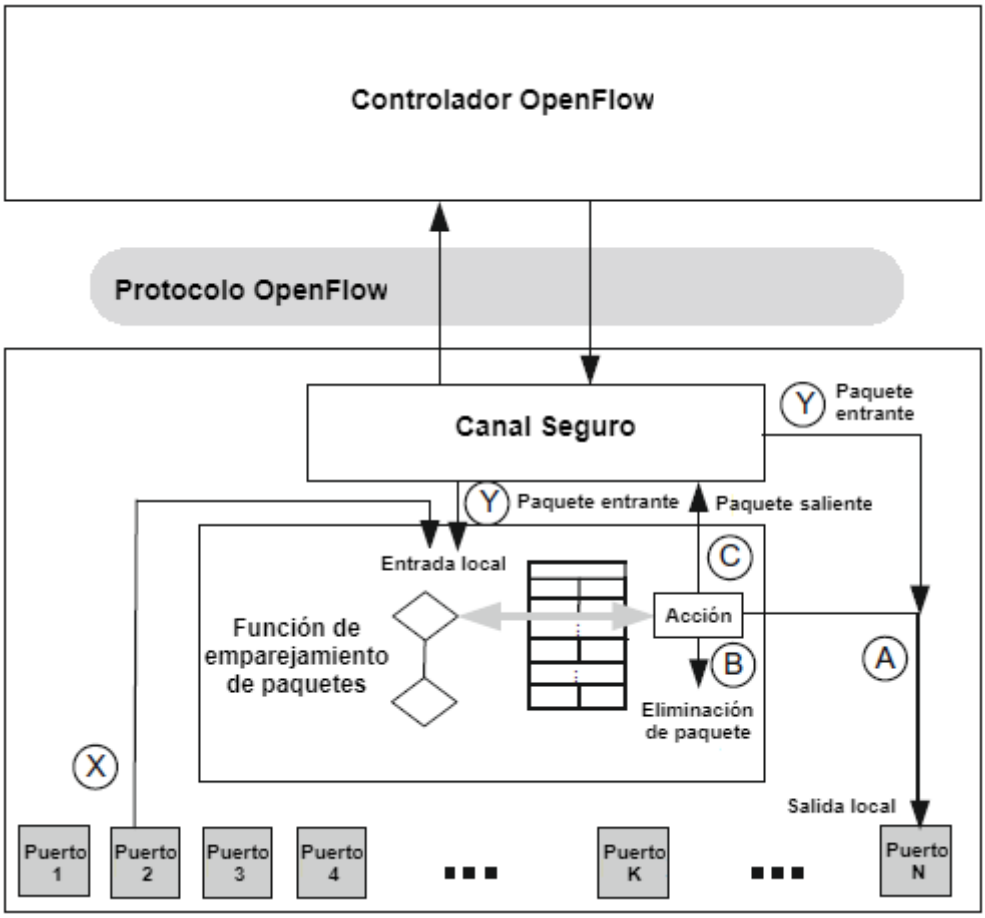


Figura 14. Switch OpenFlow V.1.0

Estos tres caminos fundamentales del paquete son ilustrados en la Figura 14. En el caso del camino C, el paquete es pasado al controlador por el canal seguro, mostrado en la figura. Si el controlador tiene o un mensaje de control o un paquete de datos que dar al switch, el controlador usa este mismo canal seguro en la dirección opuesta. Cuando el controlador tiene un paquete de datos que pasar a través del switch, este usa el mensaje de OpenFlow PACKET_OUT. Podemos ver en la Figura 14 que este paquete de datos viene del controlador puede tomar dos caminos diferentes a través de la lógica de OpenFlow, ambos denotados como Y. En el caso extremo derecho, el controlador especifica directamente el puerto de salida y que el paquete pasa por ese puerto N en el ejemplo. En

el camino Y del extremo izquierdo, el controlador indica que este quiere aplazar la siguiente decisión para la lógica de emparejamiento de paquetes.

Una implementación de switch OpenFlow dada es o "*OpenFlow-only*" o "*OpenFlow-hybrid*". Un switch "*OpenFlow-only*" es aquel que reenvía los paquetes solo de acuerdo con la lógica OpenFlow ya expresada anteriormente. Un "*OpenFlow-hybrid*" es un switch que puede también enrutar paquetes en su forma tradicional como Switch Ethernet o un router IP. Uno puede ver en el caso híbrido, como un switch OpenFlow que se encuentra próximo a un switch tradicional es totalmente independiente. Tal switch híbrido requiere un mecanismo de clasificación de preprocesamiento que direccione el paquete directamente hacia un procesamiento de OpenFlow o el procesado de paquetes tradicional. Es probable que los switches híbridos sean la norma durante la migración hacia implementaciones de OpenFlow.

6.2.2 EL CONTROLADOR DE OPENFLOW

Hay que decir que a partir de ahora estamos usando el término switch OpenFlow en vez del término dispositivo OpenFlow que usamos habitualmente. Esto es porque el término Switch es el usado en la especificación OpenFlow. En general, sin embargo, optamos por usar el término dispositivo, ya que ya hay dispositivos no enrutadores siendo controlados por controladores OpenFlow, como puntos de acceso inalámbricos.

Los switches de internet modernos toman millones de decisiones por segundo sobre si enviar o no un paquete entrante como, por ejemplo, que conjunto de puertos de salida debería ser reenviado, que campos de cabecera pueda necesitar el paquete modificar, añadir, o eliminar. Esto es una tarea muy complicada. El hecho de que esto pueda ser llevado a cabo con una tasa de medios multigigabit es una maravilla tecnológica. La industria de conmutación sabe que no todas las funciones de la ruta de conmutación se pueden llevar a cabo con esos índices, y durante mucho tiempo ha estado la idea de separar el plano de datos del plano de control.

El plano de datos empareja cabeceras, modifica paquetes y los envía basándose en un conjunto de tablas de reenvío y lógica asociada, y lo hace muy rápido. El índice de decisiones que se toman como paquetes de flujo hacia un switch a través de un interfaz de 100 Gbps es asombrosamente alto.

El plano de control ejecuta protocolos de enrutamiento de conmutación y la lógica en el plano de datos. Este proceso es muy complejo y no se puede hacer a las tasas de línea en la que los paquetes se están procesando, y es por esta razón por la que el plano de control está separado del plano de datos, incluso en los switches de redes tradicionales.

El plano de control de OpenFlow difiere del plano de control tradicional de tres formas.

1. Puede programar diferentes elementos del plano de datos con un lenguaje común y estandarizado, OpenFlow.
2. Existe un dispositivo hardware separado al plano de reenvío, al contrario que en los switches tradicionales, donde el plano de control y el plano de datos son instanciados en la misma caja física. Esta separación es posible porque el controlador puede programar los elementos del plano de datos remotamente a través de internet.
3. El controlador puede programar múltiples elementos del plano de datos desde una única instancia del plano de control.

El controlador OpenFlow es responsable de la programación de todo el emparejamiento de paquetes y las reglas de reenvío en el switch. Mientras que un router tradicional ejecutaría algoritmos de enrutamiento para determinar cómo programar la tabla de reenvío, esta función o un reemplazo equivalente sería llevada a cabo por el controlador. Cualquier cambio que resulte de recalcular rutas será programado en el switch por el controlador.

6.2.3 EL PROTOCOLO OPENFLOW

Como se muestra en Figura 14 el protocolo de OpenFlow define la comunicación entre un controlador OpenFlow y un switch OpenFlow. Este protocolo es el que únicamente identifica una tecnología OpenFlow. En esencia, el protocolo consiste en un conjunto de mensajes que son enviados del controlador al switch y el correspondiente conjunto de mensajes son enviados en la dirección contraria. Conjuntamente los mensajes permiten al controlador programar el switch permitiendo así un detallado control sobre el enrutamiento del tráfico del usuario. La programación más básica define, modifica y borra flujos.

Anteriormente definimos a un flujo como un conjunto de paquetes transferidos de un punto final de una red (o conjunto de punto finales) a otro punto final (o conjunto de puntos finales). Los puntos finales pueden definirse como puertos pares de dirección IP TCP/UDP, puntos finales VLAN, puntos finales de la capa de tres, o puertos de entrada, entre otras cosas. Un conjunto de reglas describe las siguientes acciones que el dispositivo debe tomar de todos los paquetes pertenecientes a ese flujo. Cuando el controlador define un flujo, este provee al switch con la información necesaria para saber cómo tratar los paquetes entrantes que se correspondan con ese flujo. Las posibilidades para tratarlo se han vuelto más complejas a medida que el protocolo OpenFlow ha evolucionado, pero la prescripción más básica para el tratamiento de un paquete se denota por los caminos A, B y C en la Figura 14. Estas tres opciones son para enviar el paquete por uno o más puertos de salida, eliminar el paquete o pasarlo al controlador para la gestión de excepciones.

El protocolo de OpenFlow ha evolucionado significativamente con cada versión de OpenFlow, así que podemos cubrir los detallados mensajes del protocolo en las secciones específicas de cada versión que sigue. Esta especificación ha evolucionado su lanzamiento. Los numerosos puntos que se han lanzado en los años intermedios han traído problemas con versiones anteriores y han añadido funcionalidad gradualmente. OpenFlow se había visto como una plataforma experimental en sus primeros años. Por esta razón, había poca preocupación por parte de la comunidad de desarrollo en avanzar en este estándar para proveer de una interoperabilidad entre versiones lanzadas. Como OpenFlow empezó a verse ampliamente usado comercialmente, la compatibilidad hacia attrás se volvió un asunto más importante. Muchas características, sin embargo, fueron introducidas en versiones anteriores de OpenFlow que no continúan en la versión actual. Se verán los componentes clave de cada actualización que llegan a ser la base para los avances en las posteriores actualizaciones.

6.2.4 EL CONTROLADOR DE CONMUTACIÓN DE CANAL SEGURO

El canal seguro es el camino utilizado para las comunicaciones entre el controlador OpenFlow y el dispositivo OpenFlow. Generalmente, esta comunicación está asegurada por un cifrado asimétrico basado en TLS, aunque se permiten conexiones TCP no cifradas. La Figura 15 representa estas dos variantes del canal seguro. En el ejemplo fuera de banda, vemos en la figura que la conexión segura del canal entra en el conmutador a través del puerto Z, que no se conmuta por el plano de datos OpenFlow. Algunas pilas de red heredadas entregarán los mensajes de OpenFlow a través del canal al proceso de canal seguro en el switch, donde se analizan y manejan todos los mensajes OpenFlow. Por lo tanto, el canal seguro fuera de banda es relevante sólo en el caso de un conmutador híbrido OpenFlow.

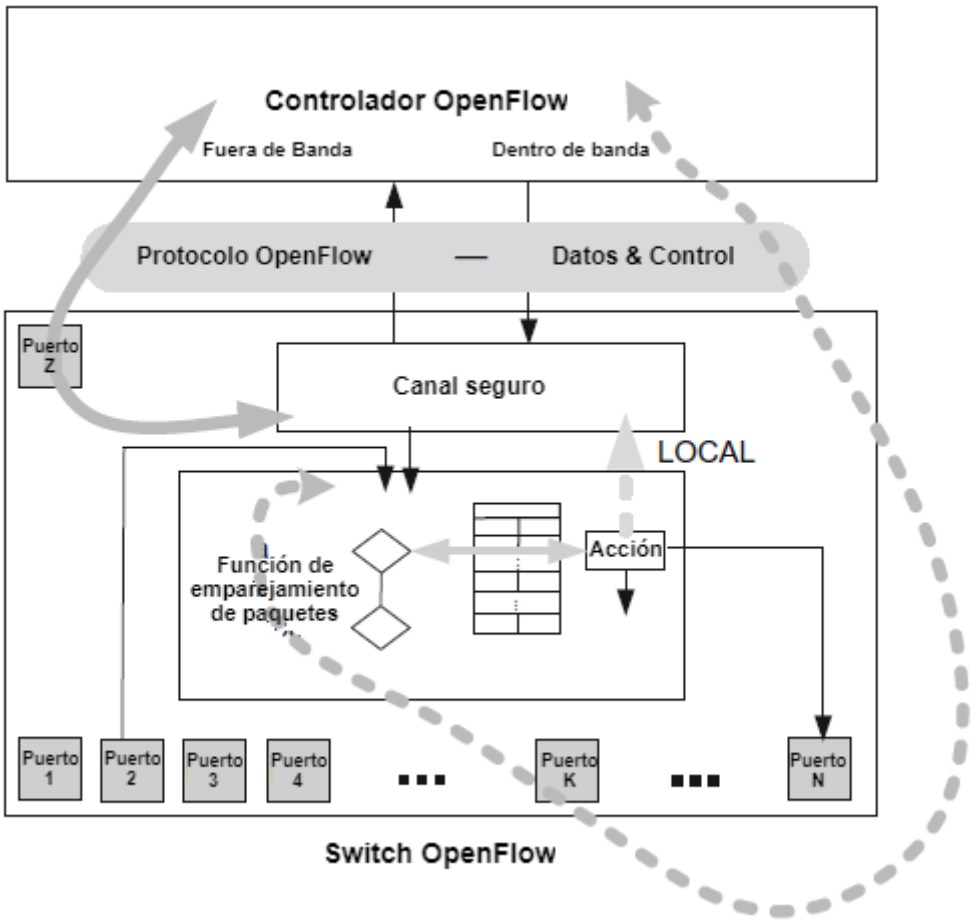


Figura 15. Canal seguro del switch-controlador OpenFlow

En el ejemplo, vemos los mensajes OpenFlow del controlador que llegan a través del puerto K, que forma parte del plano de datos OpenFlow. En este caso, estos paquetes serán manejados por el paquete OpenFlow que coincida con la lógica mostrada en la figura. Las tablas de flujo se habrán construido para que el tráfico OpenFlow se reenvíe al puerto virtual local, lo que da lugar a que se pasen los mensajes al proceso de canal seguro.

Hay que tener en cuenta que cuando el controlador y todos los switches que controlan se encuentran completamente dentro de un ambiente controlado como un centro de datos, puede ser prudente considerar no utilizar cifrado basado en TLS para asegurar el canal. Esto se debe a una sobrecarga de rendimiento al usar este tipo de seguridad, y si no es necesario, es preferible no pagar esta penalización de rendimiento.

6.3 PRINCIPIOS BÁSICOS DE OPENFLOW 1.0 Y OPENFLOW

En esta sección describimos detalladamente los componentes básicos de esta implementación inicial de OpenFlow.

6.3.1 PUERTOS Y COLAS DE PUERTOS

La especificación OpenFlow define el concepto de un puerto OpenFlow. Un puerto OpenFlow V.1.0 corresponde a un puerto físico. Este concepto se amplía en versiones posteriores de OpenFlow. Durante muchos años, los switches sofisticados han soportado varias colas por puerto físico. Estas colas generalmente se sirven de algoritmos de programación que permiten el aprovisionamiento de diferentes niveles de calidad de servicio (QoS) para diferentes tipos de paquetes. OpenFlow abarca este concepto y permite que un flujo sea mapeado a una cola ya definida en un puerto de salida. Así, si volvemos a la Figura 14, la salida de un paquete en el puerto N puede especificar en qué cola en el puerto N debe colocarse el paquete. Así, si nos acercamos a la opción A de la Figura 14, se revela lo que vemos en la Figura 16. En nuestra figura ampliada se puede ver que la caja de acciones específicamente pone en cola al paquete que se procesa en la cola 1 en el puerto N. Hay que tener en cuenta que el soporte para QoS es muy básico en V.1.0. El soporte de QoS en OpenFlow se ha expandido considerablemente en las versiones posteriores.

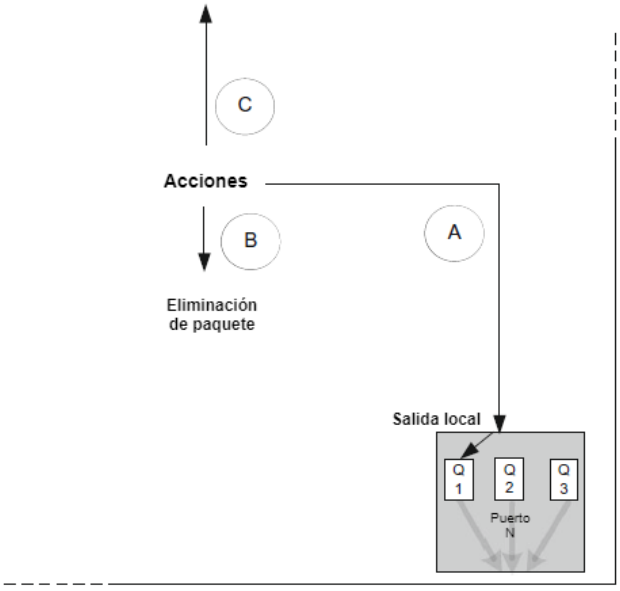


Figura 16. Compatibilidad con OpenFlow para múltiples colas por puerto

6.3.2 TABLA DE FLUJO

La tabla de flujo se encuentra en la base para la definición de un switch OpenFlow. Describimos una tabla de flujo genérico en Figura 17. Una tabla de flujo consiste en entradas de flujo, una de las cuales se muestra en la Figura 18. Un flujo de entrada consiste en campos de encabezado, contadores y acciones asociadas con esa entrada. Los campos de cabecera se utilizan como coincidencia para determinar si un paquete entrante coincide

con esta entrada. Si existe una coincidencia, el paquete pertenece a este flujo. Los contadores se usan para rastrear estadísticas relativas a este flujo, tales como cuántos paquetes tienen han sido enviados o eliminados para este flujo. Los campos de acciones prescriben lo que el switch debe hacer con un paquete que coincida con esta entrada.

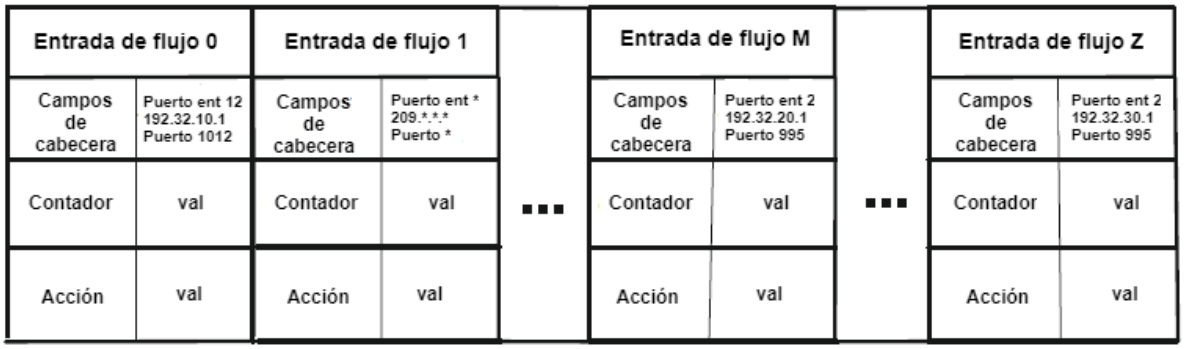


Figura 17. Tabla de flujo OpenFlow V.1.0

Campos de cabecera	Valor de campo
Contadores	Valor de campo
Acciones	Valor de campo

Figura 18. Entrada de flujo básica

6.3.3 EMPAREJAMIENTO DE PAQUETES

Cuando un paquete llega al switch de OpenFlow desde un puerto de entrada (o, en algunos casos, desde el controlador) se hace coincidir con la tabla de flujo para determinar si hay una entrada de flujo coincidente. Los siguientes campos de coincidencia asociados con el paquete entrante pueden utilizarse para coincidir con entradas de flujo:
Puerto de entrada de conmutador, VLAN ID, prioridad VLAN, dirección de origen Ethernet, Dirección de destino Ethernet, Tipo de trama Ethernet, Dirección de origen IP, Dirección de destino IP, protocolo IP, bits *IPType of Service* (ToS), puerto de origen TCP / UDP, puerto de destino TCP / UDP.

Estos 12 campos de coincidencia se denominan colectivamente las 12 filas básicas de los campos de coincidencia. El flujo de campos entrantes de coincidencia pueden ser un comodín utilizando una máscara de bit, lo que significa que cualquier valor que coincida con los bits no enmascarados en los campos de coincidencia del paquete entrante serán coincidentes. Las entradas de flujo se procesan en orden y una vez que se encuentra una coincidencia, no se realizan más intentos de coincidencia con esa tabla de flujo. (Lo veremos en las versiones posteriores de OpenFlow que puede haber tablas de flujo

adicionales en contra de qué paquete coincidente puede continuar). Por esta razón, es posible que haya múltiples entradas de flujo coincidentes para un paquete, que debe estar presente en una tabla de flujo. Sólo la primera entrada de flujo que coincida es significativa; las otras no se encontrarán, porque la coincidencia de paquetes se detiene en la primera coincidencia.

La especificación V.1.0 no dice nada sobre cuáles de estos 12 campos de coincidencia son necesarios en comparación con aquellos que son opcionales. La ONF ha aclarado esta confusión definiendo tres tipos diferentes de conformidad en su programa de prueba de conformidad. Los tres niveles son de conformidad completa, es decir, los 12 campos de coincidencia son compatibles; la conformidad de la capa dos, cuando sólo se apoya la coincidencia de campo de cabecera de la capa dos; y, finalmente, conformidad de la capa tres, cuando sólo se apoya la coincidencia de campo de encabezado de capa tres.
Si se llega al final de la tabla de flujo sin encontrar una coincidencia, se denomina error de tabla. En el caso de un error de tabla en V.1.0, el paquete se reenvía al controlador (esto ya no es estrictamente verdad en versiones posteriores). Si se encuentra una entrada de flujo coincidente, las acciones asociadas con esa entrada de flujo determinan cómo se maneja el paquete.

La acción más básica prescrita por una entrada de conmutador OpenFlow es cómo enviar este paquete. Es importante señalar que esta función de emparejamiento de paquetes V.1.0 fue diseñada como una abstracción de la forma en que el hardware de conmutación real funciona hoy. A medida que se agrega más funcionalidad en versiones posteriores del protocolo OpenFlow, se puede ver que la especificación supera a la realidad del hardware actual. En ese punto, ya no proporciona una abstracción limpia para las implementaciones actuales, pero especifica el comportamiento para el hardware de conmutación que todavía no se ha construido.

6.3.4 ACCIONES Y REENVÍO DE PAQUETES

Las acciones necesarias que deben ser soportadas por una entrada de flujo son a la salida, reenviadas o eliminadas del emparejamiento. El caso más común es que la acción de salida especifica un puerto físico por el que el paquete debe ser reenviado. Hay, sin embargo, cinco puertos virtuales especiales definidos en V.1.0 que tienen una importancia especial para la acción de salida. Son LOCAL, ALL, CONTROLLER, IN_PORT y TABLE. Describimos las transferencias de paquetes resultantes de estas designaciones de puertos virtuales en la Figura 19. En la figura, las anotaciones entre corchetes junto a las flechas anchas y sombreadas representan uno de los tipos de puertos de virtuales descritos en esta sección.

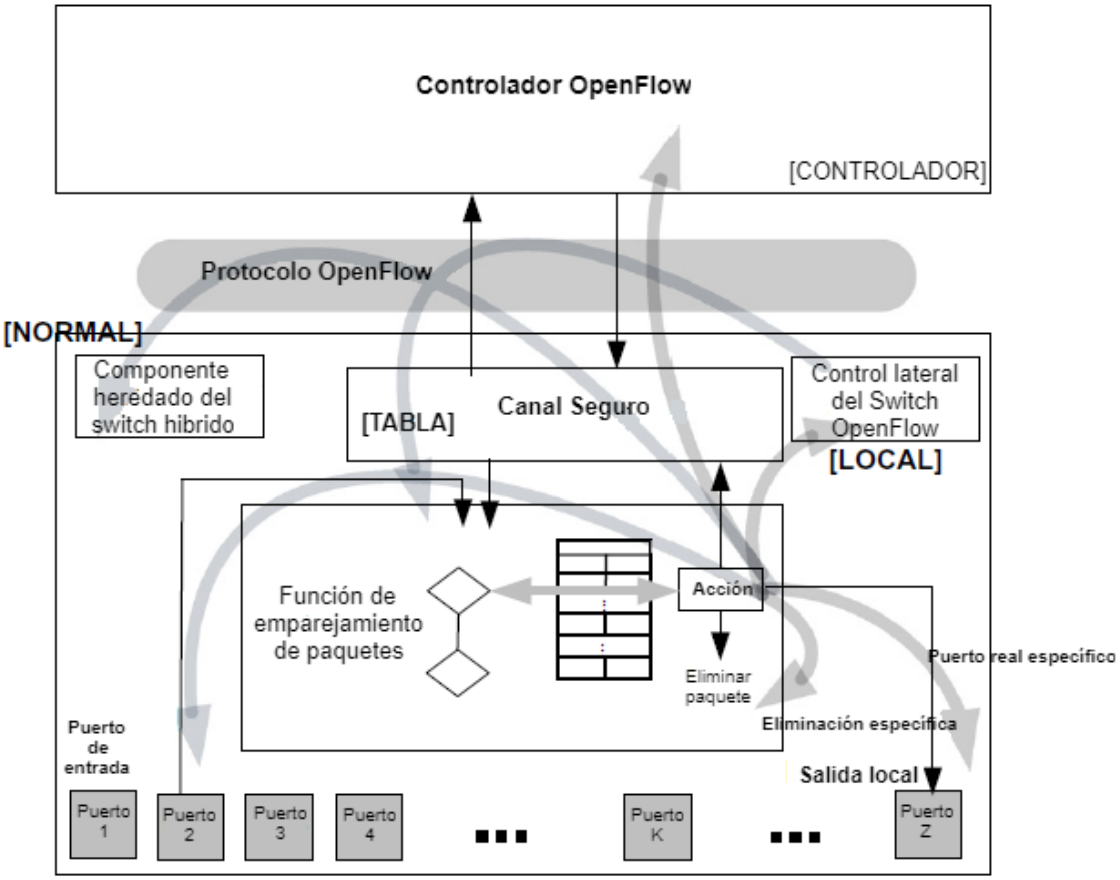


Figura 19. Rutas de paquetes correspondientes a puertos virtuales

LOCAL indica que el paquete debe ser reenviado al software de control local del switch OpenFlow, evitando el procesamiento adicional de la tubería OpenFlow. LOCAL se utiliza cuando los mensajes OpenFlow del controlador se reciben en un puerto que está recibiendo paquetes conmutados por el plano de datos OpenFlow.

- **LOCAL** indica que el paquete debe ser procesado por el software de control OpenFlow local.
- **ALL** se utiliza para inundar un paquete en todos los puertos del conmutador excepto el puerto de entrada. Esto proporciona una rudimentaria capacidad de difusión al conmutador OpenFlow.
- **CONTROLLER** indica que el conmutador debe reenviar este paquete al controlador OpenFlow.
- **IN_PORT** indica al conmutador que reenvíe el paquete por el puerto que llegó. Efectivamente, IN_PORT normalmente crea una situación de bucle invertido, que podría ser útil para ciertos escenarios. Un escenario en el que esto es útil es el caso de un puerto inalámbrico 802.11. En este caso, es bastante normal recibir un paquete de ese puerto desde un host y para reenviarlo al host receptor a través del

mismo puerto. Esto debe hacerse con mucho cuidado para no crear situaciones de bucle inesperado. Por lo tanto, el protocolo requiere la estipulación explícita de esta intención a través de este puerto virtual especial.

- **El puerto virtual TABLA**, que sólo se aplica a los paquetes que el controlador envía al switch. Estos paquetes llegan como parte del mensaje PACKET_OUT del controlador, que incluye una lista de acciones. Esta lista de acciones generalmente contendrá una acción de salida, que especificará un número de puerto.

Hay dos puertos virtuales adicionales, pero el soporte para estos es opcional en V.1.0. El primero es el puerto virtual NORMAL. Cuando la acción de salida reenvía un paquete al puerto virtual NORMAL, este envía el paquete a la lógica de reenvío heredada del switch. El uso de NORMAL tiene sentido sólo en el caso de un switch híbrido. El puerto virtual restante es FLOOD. En este caso, el conmutador envía una copia del paquete a todos los puertos excepto al puerto de ingreso.

Hay dos acciones opcionales en V.1.0: poner en cola y modificar el campo. La acción en cola selecciona una cola específica perteneciente a un puerto determinado. Esto se utilizaría junto con la acción de salida y se utiliza para alcanzar los niveles de QoS deseados mediante el uso de múltiples colas de prioridad en un puerto. Finalmente, la acción de modificar campo informa al switch cómo modificar ciertos campos de encabezado. La especificación contiene una extensa lista de campos que pueden ser modificados a través de esta acción. En particular, los encabezados VLAN, Ethernet, origen y dirección de destino, origen IPv4 y dirección de destino, y el campo TTL pueden modificarse.

El campo Modificar es esencial para las funciones de enrutamiento más básicas. Para enrutar paquetes de capa tres, un enrutador debe disminuir el campo TTL antes de reenviar el paquete hacia fuera el puerto de salida designado.

Cuando hay varias acciones asociadas con una entrada de flujo, aparecen en una lista de acciones, al igual que el mensaje PACKET_OUT descrito anteriormente. El switch debe ejecutar acciones en el orden en el que aparecen en la lista de acciones.

7. SIMULADOR

Mininet es el emulador de red que se analizará con detalla en el presente trabajo ya que tiene una gran cantidad de características que hacen que el simulador sea ideal para la experimentación y el desarrollo de entornos SDN, las cuales son:

1. **Es rápido**: iniciar una red sencilla tarda unos segundos. Esto significa que su ciclo run-edit-debug puede ser muy rápido.
2. **Puede crear topologías personalizadas:** un único conmutador, la columna vertebral de Stanford, un centro de datos o cualquier otra cosa.
3. **Puede ejecutar programas reales**: todo lo que funciona en Linux está disponible para que se ejecute, desde servidores web a las herramientas de supervisión de ventanas TCP a Wireshark.
4. **Puede personalizar el reenvío de paquetes**: Los switches de Mininet son programables mediante el protocolo OpenFlow. Los diseños de red personalizados definidos por software que se ejecutan en Mininet se pueden transferir fácilmente a los conmutadores OpenFlow de hardware para el reenvío de paquetes de velocidad de línea.
5. **Puede compartir y replicar resultados**: cualquier persona que tenga un ordenador puede ejecutar su código una vez que lo haya empaquetado.
6. **De fácil uso**: puede crear y ejecutar experimentos Mininet escribiendo scripts Python simples.
7. **Mininet es un proyecto de código abierto**, por lo que se le anima a examinar su código fuente, modificarlo, corregir errores, problemas de archivo
8. **Se pueden cambiar las características de los enlaces**: como por ejemplo el retraso o la tasa de baudios. Incluso se pueden cambiar fallos de red conectando y desconectando enlaces.

En Mininet no todo son ventajas, también es necesario conocer cuáles pueden ser sus desventajas o limitaciones, entre las que cabe destacar:

1. Mininet impone límites de recursos: ya que los recursos serán compartidos entres hosts virtuales y switches.
2. Usa un solo kernel de Linux para todos los hosts virtuales, es decir, que no puede ejecutar software que dependa de BSD, Windows u otros kernels del sistema operativo

3. No se puede hacer uso de NAT hacia al exterior ya que está aislada de su LAN e Internet. Pero se puede usar --nat para conectar la red Mininet a su LAN a través de la traducción de direcciones de red.
4. Todos los hosts de Mininet comparten el sistema de archivos, es decir, que hay que tener cuidado de no matar los procesos erróneos por error.
5. Mininet no tiene noción de tiempo virtual, significando esto que las mediciones de tiempo se basarán en tiempo real.

Para poder comenzar a trabajar con este software, en mi caso necesito tener una máquina virtual emulando el sistema operativo Linux. Para ello se utiliza VirtualBox de Oracle. Para que funcione bien la máquina virtual, se ha de configurar con suficiente memoria RAM. En este caso se configurará con 2Gb para poder ejecutar topologías diferentes.

Además, hay que configurar la máquina virtual con dos interfaces de red, una de ellas para poder acceder a internet y la otra para poder comunicarse con la máquina host. Hay que tener en cuenta, que cuando se establecen sesiones SSH, es con la IP de la última interfaz descrita con la que se establecerá la comunicación.

Además de la imagen de la VM necesitaremos un servidor X y un terminal que soporte SSH. En este caso que se va a utilizar el sistema operativo Windows 10 utilizaremos como servidor X el software Xming y como terminal utilizaremos PuTTY.

7.1 VIRTUALBOX

VirtualBox [32] es un potente software de virtualización x86 y AMD64/Intel64 para empresas y uso doméstico. Es la única solución empresarial disponible gratuitamente como software de código abierto bajo los términos de la versión 2 de la GPL.

Este software, se ejecuta en Linux, Windows, Macintosh y Solaris y soporta un gran número de sistemas operativos invitados. Además, se está desarrollando activamente y tiene una lista de características, sistemas operativos invitados y plataformas en las que se ejecuta cada vez mayor.

En cuanto a sus funcionalidades, se puede destacar la ejecución de máquinas remotas por medio del *Remote Desktop protocol* (RDP), soporta iSCSI, pero estas opciones no están aún disponibles en la versión OSE.

El software se puede conseguir en la dirección https://www.virtualbox.org/ y la instalación es sumamente sencilla. A la hora de crear la máquina virtual es importante configurar las interfaces como se ha mencionado anteriormente y que la versión del sistema operativo sea de 32 bits.

7.2. SECURE SHELL

Secure SHell (SSH) es un protocolo que utiliza el cifrado para asegurar la conexión entre un cliente y un servidor. Todas las autenticaciones de usuario, comandos, salida y transferencias de archivos se cifran para protegerlas contra ataques en la red. Además, es un método de inicio de sesión remoto seguro, proporcionando opciones alternativas para la autenticación fuerte. Es una alternativa segura a los protocolos de inicio de sesión no protegidos como puede ser Telnet o métodos de transferencia inseguros como puede ser FTP.

El protocolo SSH se utiliza con diversos motivos:

- Proporciona acceso seguro para usuarios y procesos automatizados.
- Se utiliza para transferencias de archivos interactivas y automatizadas.
- Emisión de comandos remotos
- Gestión de la infraestructura de red y otros componentes del sistema de misión crítica.

Con estas características el protocolo SSH puede convertir protocolos inseguros en seguros mediante la función del servidor de reenvío por puerto. Además, el protocolo SSH dispone de los siguientes tipos de protección. Tras una conexión de inicio, se puede constatar que la conexión se hace en el mismo servidor que se usó en otras ocasiones. Después se usa encriptación de 128 bits para que la información transmitida por el cliente para la autenticación sea robusta. Por otro lado, se usa el mismo tipo de encriptación para la transferencia de información de un extremo a otro. Y finalmente se pueden usar aplicaciones gráficas sobre una red usando un medio seguro usando el reenvío por X11[33].

7.3 XTERM

Xterm es un emulador de terminal para el sistema X Window. Provee terminales compatibles con DEC VT102 y Tekthonix 4014 para programas que no pueden usar el sistema de ventanas directamente. El terminal Xterm se conectará a un host en la red virtual.

7.4 WIRESHARK

Wireshark es el analizador de protocolo de red más utilizado del mundo. Permite ver lo que está sucediento en la red a nivel microscópico y es el estándar de facto en empresas

comerciales, agencias gubernamentales e instituciones educativas. Es utilizado para realizar análisis y solucionar problemas en redes de comunicaciones, para desarrollo de software y protocolos, y como una herramienta didáctica.

7.5 IPERF

Iperf es una herramienta para las mediciones activas del ancho de banda máximo alcanzable en redes IP. Soporta la afinación de varios parámetros relacionados con la temporización, búferes y protocolos. Para cada prueba informa del ancho de banda, la pérdida de paquetes y otros parámetros.

7.6 DPCTL

Dpctl es una utilidad de gestión que permite un cierto control sobre el switch OpenFlow. Con esta herramienta se pueden agregar flujos a la tabla de flujo, consultar las características del switch, etc.

7.7 CBENCH

Cbench es un *framework* para hacer pruebas, análisis y comparaciones basadas en Linux y además se usa para el definir el flujo de controladores Openflow.

7.8 CREAR TOPOLOGÍAS DE RED

Mininet dispone de diferentes formas para poder crear una red entre las cuales se encuentra:

- **Utilidad mn**
- **Aplicaciones gráficas**
- **Código usando una API Python**

7.8.1 UTILIDAD MN

En Mininet se usa el comando mn para arrancar el entorno y crear la topología por defecto. Este comando es muy útil ya que es rápido y sencillo de utilizar, pero no permite la personalización muy detallada de las topologías a crear.

A este comando se le pueden añadir los siguientes parámetros que se describen a continuación:

- **h**

  Permite mostrar la ayuda.

- **--switch --user|--ovsk**

Para la referencia del usuario y los conmutadores del kernel vSwitch Open, respectivamente.

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

  Este parámetro se suele usar con esta sintaxis -- controller remote, ip=[controller IP], port=[controller port] ya que así, se puede instalar un controlador desde cualquier lugar. Si no se especifican los parámetros IP y port toman los valores 127.0.0.1 para IP y 6633 para port, siendo estos los valores de la VM.

- **topo = linear|Minimal|Reserved|Single|Tree**

  Con este parámetro se crean topologías predefinidas que se explican más detenidamente en el Anexo I.

- **c**

  Eliminar la topología creada y salir

- **custom=CUSTOM**

  Sirve para poder leer y a continuación cargar una topología creada para Mininet en Python.

- **test= cli|build|pingall|pingpair|iperf|all|iperfudp|none**

  Se encarga de realizar un test a la recien cargada topología.

  - **pingall**: verifica la conectividad de todos los pares con el switch y los hosts.
  - **pingpair**: ejecuta un ping de prueba para todos los pares.
  - **iperf**: ejecuta un servidor iperf en un host y un cliente iperf en otro host y analiza el ancho de banda alcanzado
  - **none**: registra la hora de configuración y elimina una topología

- **i IPBASE**

  IP base para los hosts

- **Mac**

  Asignación automática de macs sencillas para las interfaces de red

- **Arp**

  Su utilidad consiste en rellenar las tablas arp de la red

- **v info|warning|critical|error|debug|output**

  - **debug**: imprime información que se oculta.
  - **output**: con este comando se imprime la salida CLI y una poca información más.
  - **warning**: se utiliza con las pruebas de regresión para ocultar la información de la función de salida no necesaria.

- **version**

  Para elegir la versión de Mininet ya que hay varias versiones.

7.8.2 APLICACIONES GRÁFICAS

Mininet incorpora un script GUI en Python llamado MiniEdit, sirve para la creación de topologías de red.

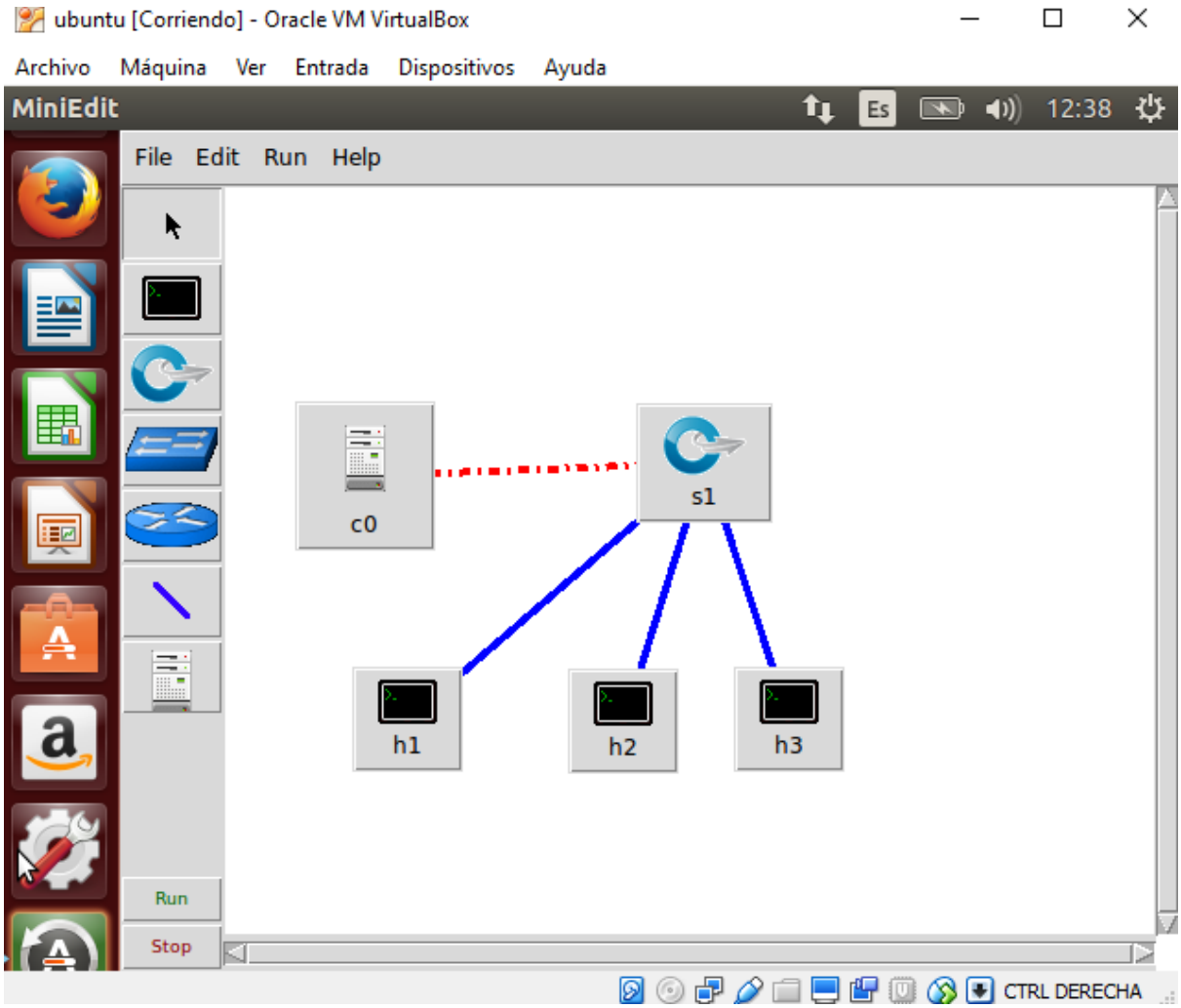


Figura 20. MiniEdit

7.8.3 APLICACIONES MININET

Mininet presenta un grupo de aplicaciones en Python para empezar a trabajar, y en este apartado se verán algunas.

- **Baresshd.py**

  Utiliza una API de nivel medio de Mininet para crear un proceso SSHD que se ejecuta en un espacio de nombres pero no utiliza OpenFlow.

- **Bind.py**

  Se usa para mostrar cómo crear directorios privados para cada nodo en una topología Mininet.

- **Cluster.py**

  Contiene el código para la edición de un cluster experimenta. Las clases remotas y MininetCluster se importan desde este ejemplo para crear una topología con nodos en máquinas remotas.

- **ClusterSanity.py**

  Ejecuta la edición del Cluster localmente para probar la funcionalidad básica.

- **Clustercli.py**

  Contiene una CLI para la edición del clúster experimental.

- **Clusterdemo.py**

  Edición del clúster en 3 servidores con una topología de árbol de profundidad 3 y fanout 3.

- **Consoles.py**

  Es una aplicación que origina ventanas, una para cada host, switch y controlador, y que permite interactuar con ellos con las demás ventanas, ya ofrece una interacción gráfica.

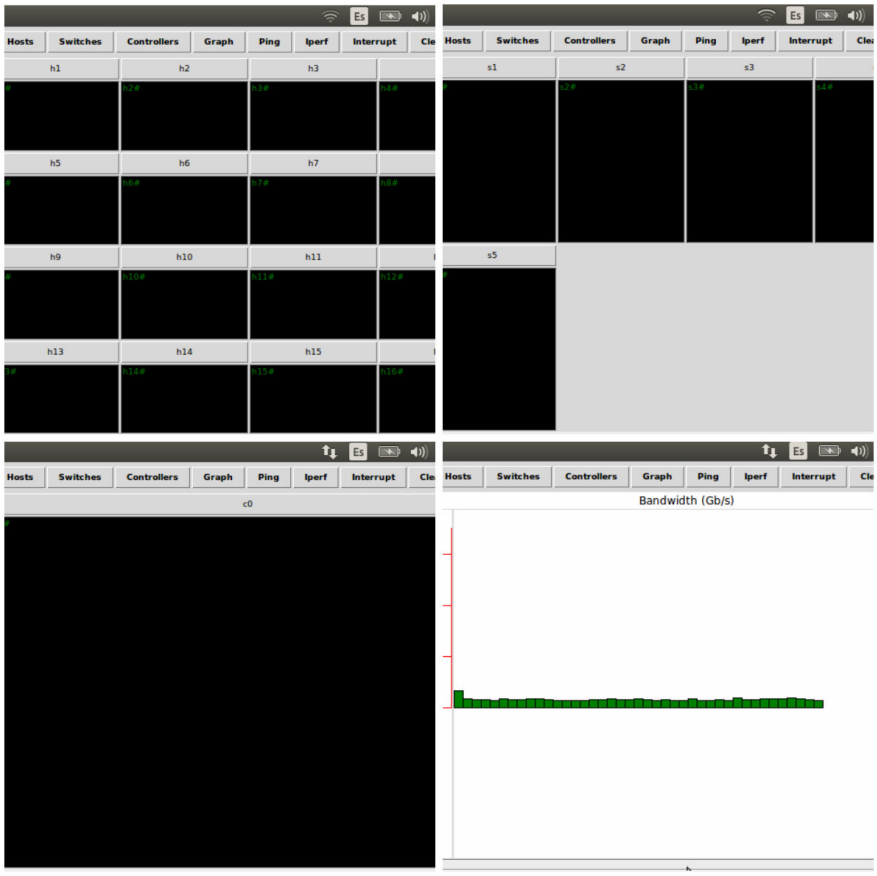


Figura 21. Consoles.py

- **Controllers.py**

  Crea una red con varios controladores, mediante un switch () que es una subclase personalizada.

- **Controllers2.py**

  Crea una red con varios controladores creando una red vacía, agregando nodos a ella y arrancando los switches manualmente.

- **Controlnet.py**

  Muestra cómo se puede modelar la red de control, así como la red de datos, creando solamente dos objetos Mininet

- **Cpu.py**

  Prueba el ancho de banda de iperf para variar los límites de la CPU

- **Emptynet.py**

  Se una para producir una red vacía como se muestra en la siguiente figura y la adición de nodos a ella.

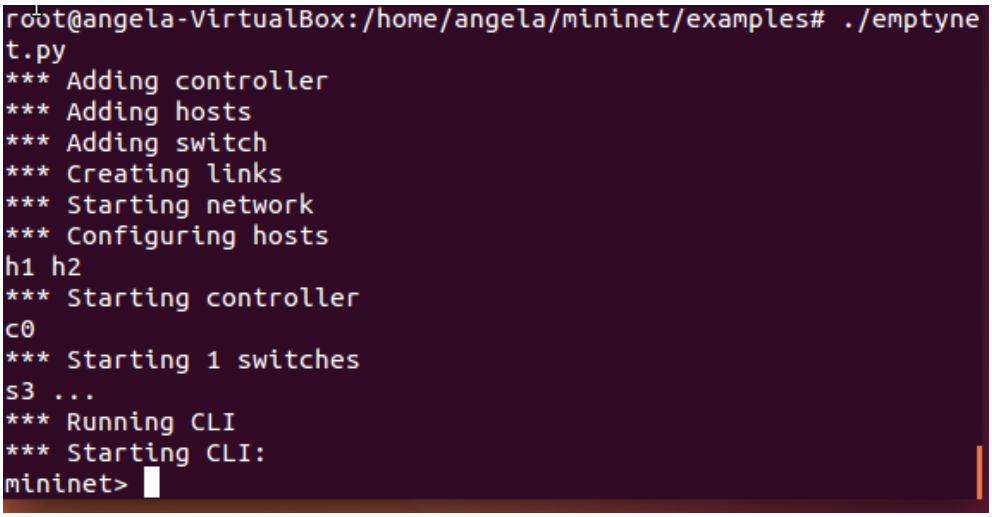


Figura 22. Emptynet.py

- **Hwintf.py**

  Muestra cómo agregar una interfaz a una red después de crear la red

- **intfoptions.py**

  Reconfigura un TCIntf durante el tiempo de ejecución con diferentes comandos de control de tráfico para analizar el ancho de banda, la pérdida y el retraso.

- **limit.py**

  Se muestra cómo usar los límites de alcance y CPU

- **Linearbandwidth.py**

  Muestra cómo crear una topología personalizada mediante la programación subclasificando Topo y además ejecuta una serie de pruebas en ella.

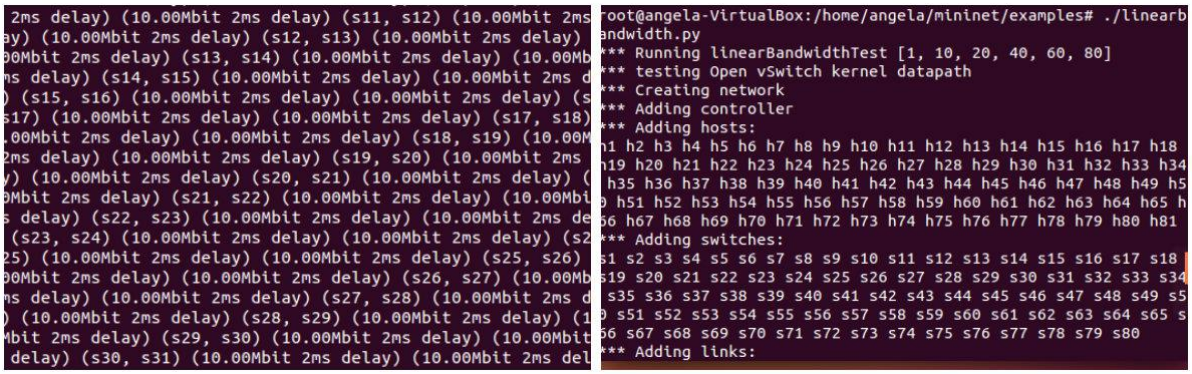


Figura 23. Linearbandwidth.py

- **linuxrouter.py**

  Muestra cómo crear y configurar un router en Mininet que utiliza el reenvío IP de Linux
- **mobility.py**

  Muestra cómo separar una interfaz de un switch y adjuntar otro como una forma básica de mover un host alrededor de una red.
- **multiLink.py**

  Muestra la creación de varios enlaces entre nodos utilizando una clase de topología personalizada.
- **multiping.py**

  Muestra un método para supervisar la salida de múltiples hosts, utilizando node.monitor()
- **multipoll.py**

  Muestra el seguimiento de archivos de salida de varios hosts.
- **multitest.py**

  Crea una red y ejecuta pruebas en ella
- **nat.py**

  Muestra cómo conectar una red Mininet a Internet mediante NAT.
- **natnet.py**

  Muestra cómo crear una red que utiliza un nodo NAT para conectar hosts a Internet.
- **numberedports.py**

  Verifica que los números de puerto en Mininet coinciden con los números de puerto OVS. También verifica que los números de puerto coinciden con los de interfaz
- **sshd.py**

  Muestra cómo ejecutar un proceso SSHD en cada host, lo que permite iniciar sesión a través de SSH. Esto requiere conectar la red de datos de Mininet a una interfaz en el espacio de nombres raíz.

8. PRUEBAS Y DISCUSIÓN DE RESULTADOS

A continuación, se harán una serie de pruebas iniciales. Para ello se utiliza un ordenador portátil que consta con un procesador Intel core i7. Para la puesta en marcha se ha necesitado instalar VirtualBox que es un software de virtualización, donde se ha procedido a instalar Ubuntu 14.04 como sistema operativo. Además, se realizan pruebas para confirmar la compatibilidad con el sistema operativo y el equipo.

Para iniciar las pruebas se propone inicialmente dos escenarios. Después se procederá a establecer un controlador OpenFlow, la topología se ha tomado en base al tutorial de https://github.com/mininet/openflow-tutorial/wiki/Learn-Development-Tools , el cúal se ha traducido y ampliado en el Apéndice II.

8.1 CREACIÓN DE LA TOPOLOGÍA

En este apartado se verá cómo desarrollar la topología anteriormente propuesta. a través de del lenguaje de programación de Python se desarrollará esta topología de control SDN.

Para el primer escenario se desarrolla una topología que consta de un controlador, dos switches OpenFlow y cinco hosts. Como se puede observar en la Figura 24 tres de los hosts irán conectados al switch 1 y los otros dos restantes irán conectados al switch 2.

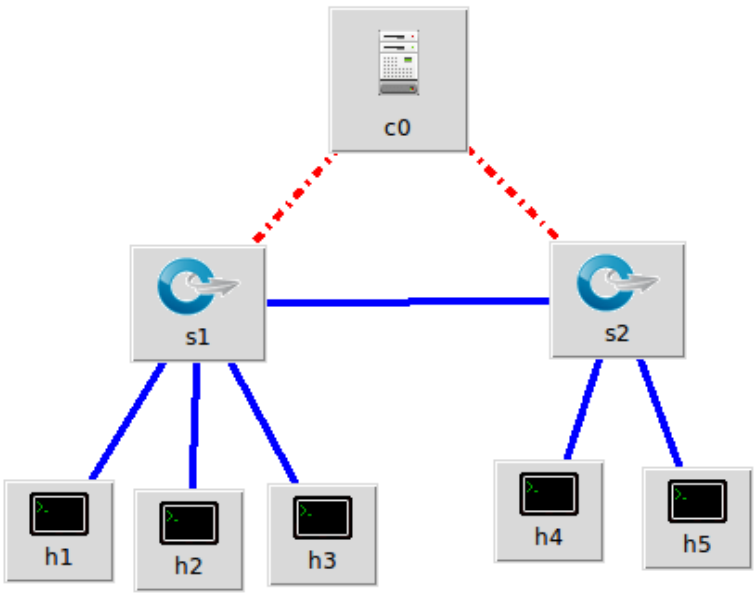


Figura 24. Topología del escenario 1

A continuación, se puede observar el código desarrollado para la creación de esta topología.

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')

    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                         controller=Controller,
                         protocol='tcp',
                         port=6633)

    info( '*** Add switches\n')
    s2 = net.addSwitch('s2', cls=OVSKernelSwitch)
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch)

    info( '*** Add hosts\n')
    h3 = net.addHost('h3', cls=Host, ip='10.0.0.3', defaultRoute=None)
    h1 = net.addHost('h1', cls=Host, ip='10.0.0.1', defaultRoute=None)
    h4 = net.addHost('h4', cls=Host, ip='10.0.0.4', defaultRoute=None)
    h5 = net.addHost('h5', cls=Host, ip='10.0.0.5', defaultRoute=None)
    h2 = net.addHost('h2', cls=Host, ip='10.0.0.2', defaultRoute=None)

    info( '*** Add links\n')
    net.addLink(s1, h1)
    net.addLink(s1, h2)
    net.addLink(s1, h3)
    net.addLink(s1, s2)
    net.addLink(s2, h5)
    net.addLink(s2, h4)

    info( '*** Starting network\n')
    net.build()
    info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()

    info( '*** Starting switches\n')
    net.get('s2').start([c0])
    net.get('s1').start([c0])

    info( '*** Post configure switches and hosts\n')

    CLI(net)
    net.stop()

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

Es importante saber cómo funciona cada función de agregar hosts, controladores, switches y links para crear tu propia topología. Por ello, ahora se verá cada una de las funciones que aparecen en el código.

```
addController( self,
               name='c0',
               controller=None,
               more params
               )
```

Parámetros usados:

- **protocol**: en nuestro caso utilizaremos TCP aunque también funciona con UDP
- **port**: 6633 ya que es el puerto por defecto de OpenFlow.

```
addHost  ( self,
           name,
           cls=None,
           more params
           )
```

Parámetros usados:

- **name**: nombre del host a añadir
- **cls**: constructor personalizado en el que pondremos que es un Host
- **ip**: se pondrá la IP que queramos para cada host.
- **defaultRourte**: para añadir la ruta por defecto

```
addSwitch  ( self,
           name,
           cls=None,
           more params
           )
```

Parámetros usados:

- **name**: nombre que le queramos dar al switch
- **cls**: OVSKernelSwitch para especificar que es un switch OpenFlow

addLink (auto,
node1,
node2,
port1=None,
port2=None,
cls=None,
more params
)

Parámetros:

- **node1**: nodo origen o nombre
- **node2**: nodo destino o nombre
- **port1**: puerto de origen (opcional)
- **port2**: puerto de destino(opcional)
- **cls**: clase de enlace

build (self)

Crea la estructura de Mininet.

Para comprobar la topología que se ha creado y que se le ha dado en este caso el nombre de escenario1.py, se hará dentro del directorio donde esté guardado y se ejecutará esta serie de comandos.

```
$ sudo python escenario1.py
```

Este comando ejecutará el código y pondrá en marcha la topología. A continuación, se procederá a hacer un ping a todos los dispositivos para rellenar las tablas de flujo y para que solo tenga que consultar la tabla. De esta manera, una vez rellena la tabla de flujo se puede ver una disminución considerable de los tiempos. En la Figura 25 se puede apreciar en la imagen de arriba que aún no se había rellenado la tabla de flujo y en la imagen de abajo si, reduciendo los tiempos considerablemente.

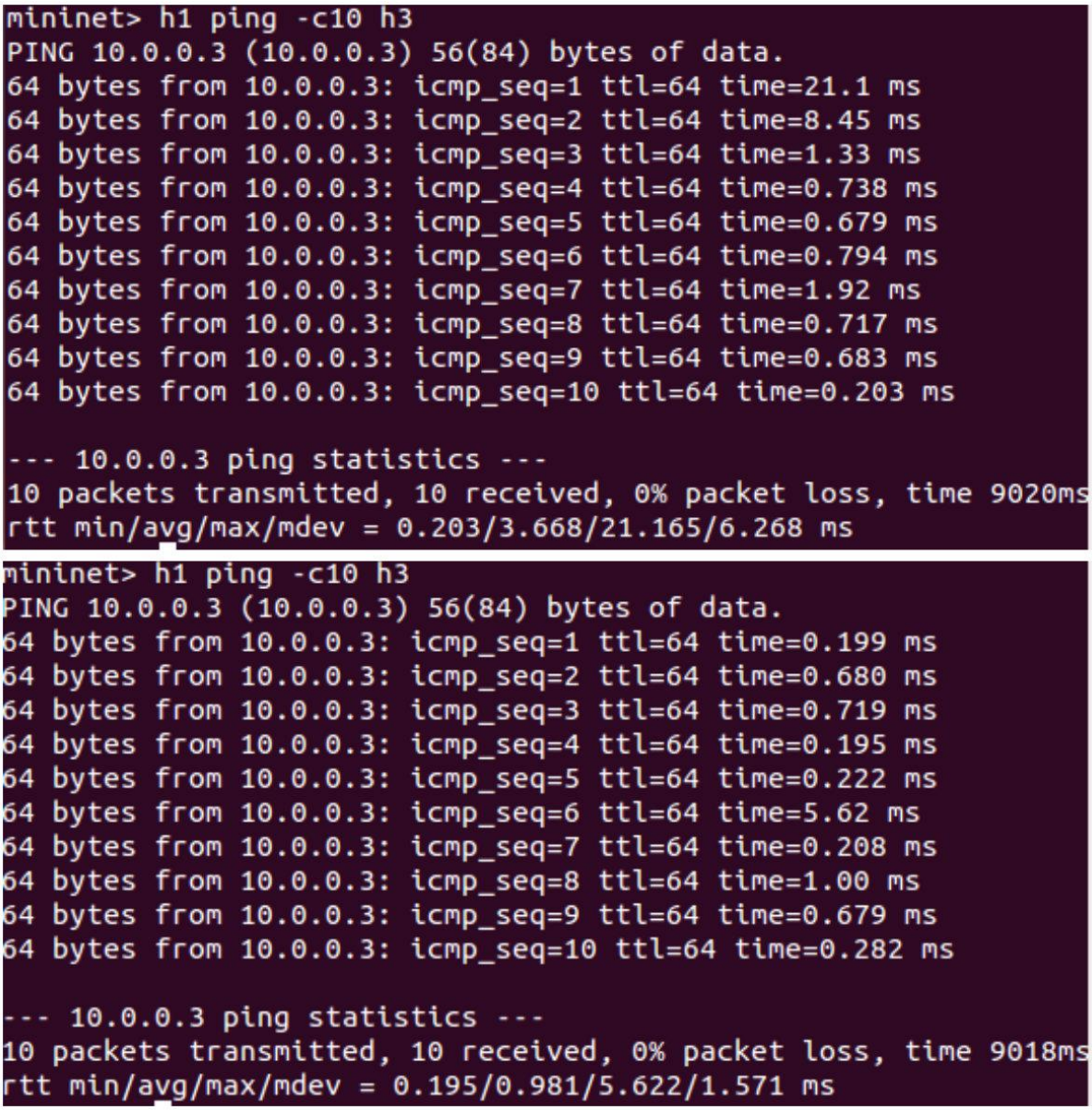


Figura 25. Reducción de tiempos

8.2 HUB.PY

A través del comando explicado en el Anexo II (dpctl dump-flows) se puede observar en este código ejemplo que la tabla contiene una entrada, en la que su acción es FLOOD y cómo se explica en el código más abajo, es para hacer una inundación de todos los puertos menos por el que llegó.

```
21
22  from pox.core import core
23  import pox.openflow.libopenflow_01 as of
24  from pox.lib.util import dpidToStr
25
26  log = core.getLogger()
27
28  def _handle_ConnectionUp (event):
29  #Genera flujos openflow.
30    msg = of.ofp_flow_mod()
31  # Hace que los flujos generados anteriormente circulen por todos los puertos
32  #salida menos por el que entró(inundación).
33    msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
34  #Emite un mensaje OpenFlow al vSwitch
35    event.connection.send(msg)
36    log.info("Hubifying %s", dpidToStr(event.dpid))
37
38  #launch() se llama automaticamente, inscribiendo los listener y donde se generan
39  #los objetos de la clase.
40  def launch ():
41  #Crea un objeto ConnectionUp y realiza la accion.
42    core.openflow.addListenerByName("ConnectionUp", _handle_ConnectionUp)
43
44    log.info("Hub running.")
```

8.3 L2_LEARNING.PY

Este componente hace que los switches OpenFlow actúen como un tipo de switch de aprendizaje L2. Éste funciona como el ejemplo "pyswitch" de NOX, aunque la implementación es bastante diferente. Mientras este componente aprende las direcciones L2, los flujos que instala son coincidencias exactas con el mayor número posible de campos. Por ejemplo, las diferentes conexiones TCP darán como resultado la instalación de diferentes flujos.

```
26  from pox.core import core
27  import pox.openflow.libopenflow_01 as of
28  from pox.lib.util import dpid_to_str
29  from pox.lib.util import str_to_bool
30  import time
31
32  log = core.getLogger()
33
34  # We don't want to flood immediately when a switch connects.
35  # Can be overriden on commandline.
36  _flood_delay = 0
37
38  class LearningSwitch (object):
39
40    def __init__ (self, connection, transparent):
41      # Se le van a agregar al switch capacidades de aprendizaje de conmutación de
42      self.connection = connection
43      self.transparent = transparent
44
45      # Nuestra tabla
46      self.macToPort = {}
47
48      # Se escuchan los mensajes entrantes y para ello escuchamos la conexión
49      connection.addListeners(self)
50
51      # Solo se usara para saber cuando registrar un mensaje útil
52      self.hold_down_expired = _flood_delay == 0
```

```
53
54    def _handle_PacketIn (self, event):
55
56      packet = event.parsed
57
58      def flood (message = None):
59        """ Inunda con el paquete """
60        msg = of.ofp_packet_out()
61        if time.time() - self.connection.connect_time >= _flood_delay:
62          # Solo habra inundación si hemos estado conectados por un tiempo
63          if self.hold_down_expired is False:
64            self.hold_down_expired = True
65            log.info("%s: Flood hold-down expired -- flooding",
66                dpid_to_str(event.dpid))
67
68          if message is not None: log.debug(message)
69          #log.debug("%i: flood %s -> %s", event.dpid,packet.src,packet.dst)
70          # OFPP_FLOOD es opcional pero en algunos switches puede que haya que
71          # cambiarlo por OFPP_ALL.
72          msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
73        else:
74          pass
75          #log.info("Holding down flood for %s", dpid_to_str(event.dpid))
76        msg.data = event.ofp
77        msg.in_port = event.port
78        self.connection.send(msg)
```

```
80      def drop (duration = None):
81        """
82        Elimina el paquete y , opcinalmene, instala un flujo para continuar dejando
83        caer paquetes similares por un tiempo.
84        """
85        if duration is not None:
86          if not isinstance(duration, tuple):
87            duration = (duration,duration)
88          msg = of.ofp_flow_mod()
89          msg.match = of.ofp_match.from_packet(packet)
90          msg.idle_timeout = duration[0]
91          msg.hard_timeout = duration[1]
92          msg.buffer_id = event.ofp.buffer_id
93          self.connection.send(msg)
94        elif event.ofp.buffer_id is not None:
95          msg = of.ofp_packet_out()
96          msg.buffer_id = event.ofp.buffer_id
97          msg.in_port = event.port
98          self.connection.send(msg)
99
100     self.macToPort[packet.src] = event.port # 1
101
102     if not self.transparent: # 2
103       if packet.type == packet.LLDP_TYPE or packet.dst.isBridgeFiltered():
104         drop() # 2a
105         return
106
107     if packet.dst.is_multicast:
108       flood() # 3a
109     else:
110       if packet.dst not in self.macToPort: # 4
111         flood("Port for %s unknown -- flooding" % (packet.dst,)) # 4a
112       else:
113         port = self.macToPort[packet.dst]
114         if port == event.port: # 5
115           # 5a
116           log.warning("Same port for packet from %s -> %s on %s.%s.  Drop."
117               % (packet.src, packet.dst, dpid_to_str(event.dpid), port))
118           drop(10)
```

```
        return
      # 6
      log.debug("installing flow for %s.%i -> %s.%i" %
                (packet.src, event.port, packet.dst, port))
      msg = of.ofp_flow_mod()
      msg.match = of.ofp_match.from_packet(packet, event.port)
      msg.idle_timeout = 10
      msg.hard_timeout = 30
      msg.actions.append(of.ofp_action_output(port = port))
      msg.data = event.ofp # 6a
      self.connection.send(msg)


class l2_learning (object):
  """
  Espera que los switches OpenFlow se conecten y los convierte en
  switches de aprendizaje.
  """
  def __init__ (self, transparent):
    core.openflow.addListeners(self)
    self.transparent = transparent

  def _handle_ConnectionUp (self, event):
    log.debug("Connection %s" % (event.connection,))
    LearningSwitch(event.connection, self.transparent)


def launch (transparent=False, hold_down=_flood_delay):
  """
  Inicia el switch de aprendizaje L2.
  """
  try:
    global _flood_delay
    _flood_delay = int(str(hold_down), 10)
    assert _flood_delay >= 0
  except:
    raise RuntimeError("Expected hold-down to be a number")

  core.registerNew(l2_learning, str_to_bool(transparent))
```

En resumen, este algoritmo se ve así:

Para cada paquete del switch (Nota: los números en verde en el código son los siguientes pasos):

1. Utilice la dirección de origen y cambie el puerto para actualizar la tabla de direcciones / puertos
2. ¿Es transparente = Falso y Ethertype es LLDP o la dirección de destino del paquete es una dirección de puente filtrado?

   Sí:

   a) Paquete suelto - no reenvíe el tráfico local de enlace (LLDP, 802.1x)

   HECHO
3. ¿Es el destino multicast?

   Sí:

   a) Inundar

HECHO

4. ¿El puerto para la dirección de destino está en nuestra tabla de direcciones/puertos?
   No:
   a) Inundar el paquete
   HECHO
5. ¿El puerto de salida es el mismo que el puerto de entrada?
   Sí:
   a) Eliminar paquetes y otros similares por un tiempo
6. Instale la entrada flujo de la tabla en el switch para que el flujo salga por el puerto apropiado
   a) Enviar el paquete hacia el puerto apropiado

8.4. CAPTURA OPENFLOW

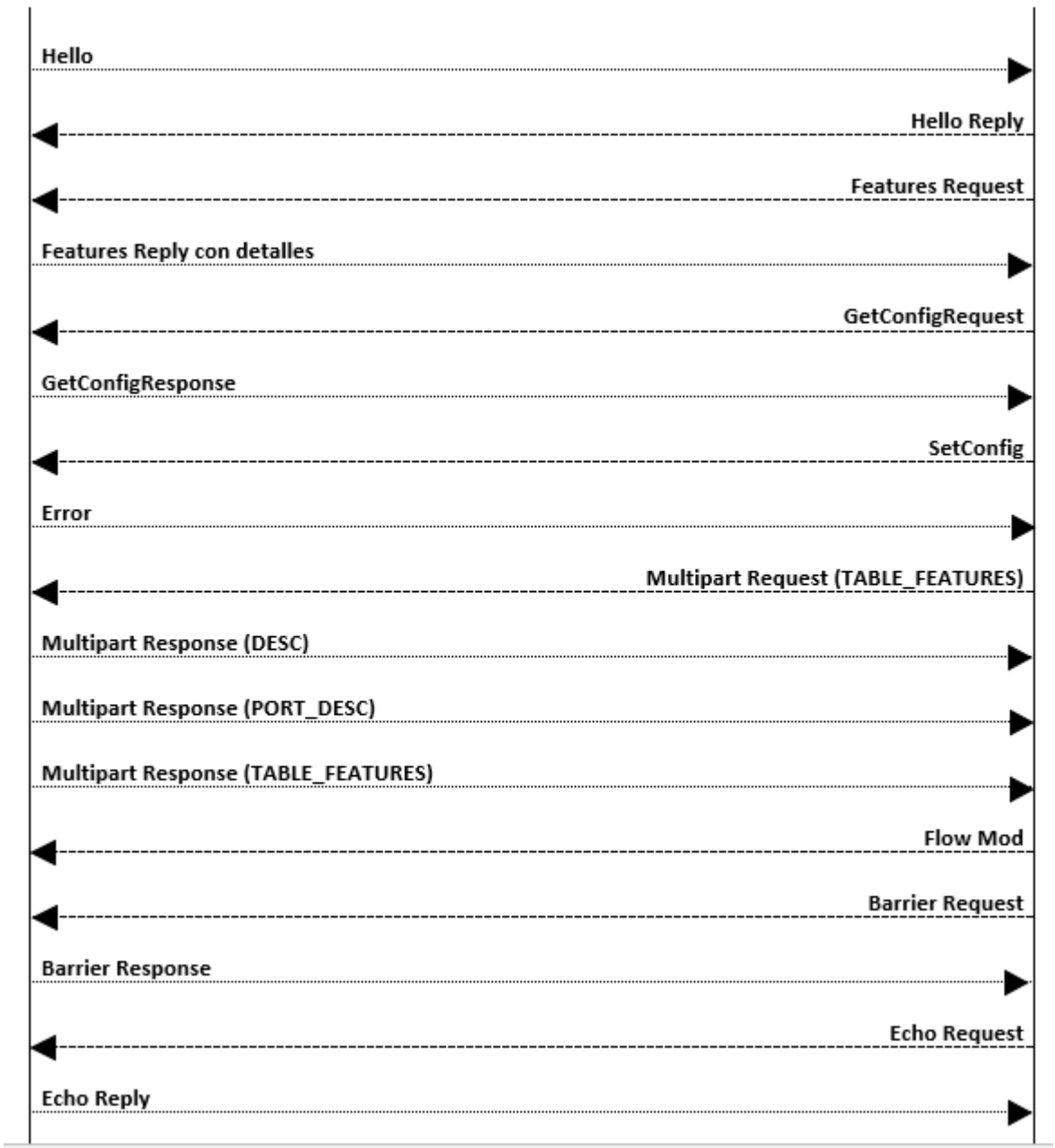
En este apartado se va a analizar los paquetes OpenFlow de una captura de tráfico. Se analizarán los tipos de paquetes enviados y se expondrá una figura que representa el flujo de estos.

La captura de paquetes OpenFlow se puede observar en la Figura 26 en la que aparece una conversación entre switch y controlador.

No.	Time	Source	Destination	Protocol	Length	Info
43	8.477456	192.168.56.246	192.168.56.31	OpenFlow	82	Type: OFPT_HELLO
46	8.483558	192.168.56.31	192.168.56.246	OpenFlow	82	Type: OFPT_HELLO
48	8.682936	192.168.56.31	192.168.56.246	OpenFlow	74	Type: OFPT_FEATURES_REQUEST
49	8.684087	192.168.56.246	192.168.56.31	OpenFlow	98	Type: OFPT_FEATURES_REPLY
50	8.684959	192.168.56.31	192.168.56.246	OpenFlow	78	Type: OFPT_SET_CONFIG
51	8.685866	192.168.56.246	192.168.56.31	OpenFlow	90	Type: OFPT_ERROR
52	8.686009	192.168.56.31	192.168.56.246	OpenFlow	114	Type: OFPT_MULTIPART_REQUEST, OFPMP_TABLE_FEATURES
53	8.687547	192.168.56.246	192.168.56.31	OpenFlow	1138	Type: OFPT_MULTIPART_REPLY, OFPMP_DESC
54	8.687551	192.168.56.246	192.168.56.31	OpenFlow	402	Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_DESC
56	8.687556	192.168.56.246	192.168.56.31	OpenFlow	362	Type: OFPT_MULTIPART_REPLY, OFPMP_TABLE_FEATURES[Malformed Packet]
71	8.844298	192.168.56.31	192.168.56.246	OpenFlow	122	Type: OFPT_FLOW_MOD
73	9.041911	192.168.56.31	192.168.56.246	OpenFlow	1138	Type: OFPT_BARRIER_REQUEST
74	9.051102	192.168.56.246	192.168.56.31	OpenFlow	74	Type: OFPT_BARRIER_REPLY
82	14.096626	192.168.56.31	192.168.56.246	OpenFlow	74	Type: OFPT_ECHO_REQUEST
83	14.097553	192.168.56.246	192.168.56.31	OpenFlow	74	Type: OFPT_ECHO_REPLY

Figura 26. Conversación entre switch y controlador

A continuación, se puede ver un diagrama de flujo de los paquetes anteriores.



Cada mensaje de OpenFlow comienza con la misma estructura de encabezado. Esta estructura fija sirve a tres roles que son independientes de la versión de OpenFlow que se utiliza. Primero, el campo de versión indica la versión de OpenFlow a la que pertenece este mensaje. Segundo, el campo de longitud indica dónde terminará este mensaje en la secuencia de bytes comenzando desde el primer byte del encabezado. En tercer lugar, el xid, o identificador de transacción, es un valor único utilizado para hacer coincidir las solicitudes con las respuestas. El campo tipo que indica qué tipo de mensaje está presente y cómo interpretar la carga, depende de la versión.

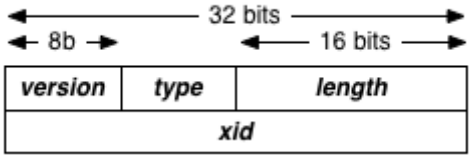


Figura 27. Encabezado OpenFlow

Hay muchos tipos de mensajes que se pueden ver en el siguiente enlace http://flowgrammable.org/sdn/openflow/message-layer/#tab_ofp_1_0

Los mensajes que se pueden ver en la captura son:

- **Hello**: Los mensajes Hello se intercambian entre el switch y el controlador al iniciar la conexión, es un elemento importante en la negociación de la versión de OpenFlow.
- **Features**: el controlador solicita funciones / capacidades del switch. Después el switch le responde con sus capacidades.
- **Get Config y Set Config**: Es una secuencia de mensajes que se usa para consultar y establecer las propiedades para la fragmentación en el procesamiento de paquetes. Además, ayuda a determinar qué cantidad de un paquete se compartirá con el controlador. GetConfig es un mensaje de confirmación y lo inicia el controlador. El controlador puede alterar el estado con el mensaje SetConfig.
- **Error**: Este mensaje puede ser enviado desde el switch o el controlador e indica que ha fallado una operación. Los errores más simples son mensajes malformados o la negociación de una versión fallida, mientras que los escenarios más complejos se reducen al fallo en el cambio de estado en el switch.
- **Multipart Request**: Se usa para solicitar información sobre los flujos individuales. Además se añaden nuevas características para agregar al cuerpo de este mensaje como por ejemplo TableFeatures, Meter, MeterConfig y MeterFeatures.
- **Multipart Reply**: es la respuesta a Multipart Request y tiene también muchas características.
- **FlowMod**: este mensaje le permite al controlador modificar el estado de un switch OpenFlow. Todos los mensajes FlowMod comienzan con el encabezado estándar OpenFlow, que contiene la versión y los valores de tipo apropiados, seguidos de la estructura FlowMod.

- **Barrier Request/Response**: Los mensajes de modificación de estado del controlador pueden ejecutarse en orden arbitrario por el switch. El controlador puede usar una solicitud para establecer un punto de sincronización, asegurándose que todos los mensajes de estado anteriores se completen antes de que la respuesta se devuelva al controlador.
- **Echo Request/Response**: se usa para intercambiar información sobre la latencia, ancho de banda y la capacidad de avanzar. El tiempo de espera de echo indica la desconexión.

9. CONCLUSIONES Y LÍNEAS DE TRABAJO FUTURAS

Como se ha visto a lo largo de todo el proyecto, las Redes Definidas por Software son una tecnología aún en desarrollo que pretenden ser la solución a los problemas de las redes actuales debido a la gran demanda de recursos por la aparición de nuevas tecnologías, servicios y usuarios.

También es importante hacer referencia a que cada vez la movilidad del usuario o la virtualización de los servidores hacen de SDN una posible solución al nuevo paradigma para la programación de la red y que será dentro de poco muy importante en el desarrollo de gran cantidad de redes.

En el proyecto se ha definido qué es SDN, cuáles son sus ventajas y qué papel juega en la virtualización de sistemas. Además, se ha explicado qué es y cómo funciona el protocolo OpenFlow, que es el protocolo principal que usan las redes SDN. Además, en el transcurso de la explicación de SDN y OpenFlow se ha visto qué es un controlador y cómo trabaja con OpenFlow. Por otro lado, se han explicado varios controladores de OpenFlow y proyectos que trabajan con este protocolo.

Además, en el estudio de los proyectos OpenFlow, se ha explicado el software Mininet, que es un simulador que otorga un escenario ideal para la creación de un número infinito de diferentes topologías de red, proporcionando así un entorno excelente para facilitar las pruebas de investigación.

Gracias a los ejemplos propuestos y a los anexos se ha podido comprender como usar las utilidades que nos ofrece OpenFlow y SDN. Esto sirve para que se puedan desarrollar nuevas aplicaciones SDN para campos como las redes móviles o las redes ópticas.

Con todo lo anterior, se concluye finalmente que se han conseguido los objetivos planteados para el presente trabajo fin de grado, y que fueron detallados en el punto 3 del presente documento.

9.1 LÍNEAS FUTURAS

Este estándar abre la puerta a gran cantidad de aplicaciones innovadoras, de hecho, hay diferentes grupos de investigación que tienen planteados diferentes proyectos de futuro entre los que se encuentran:

- El Switch OpenFlow será la conexión central entre las redes de varios niveles y será un enlace transparente para switches permitiendo así arquitecturas flexibles.
- La implementación de MPLS a través de OpenFlow, para ello se necesita la implementación de OpenFlow a través de redes de ISPs, para poder garantizar servicios de calidad.

Como se ha podido ver a lo largo del presente trabajo, tanto las Redes Definidas por Software como el propio protocolo OpenFlow ofrecen muchas posibilidades para continuar con su estudio, en los puntos resaltados anteriormente o en otros, dado que ambos se encuentran en un punto de desarrollo en el cual se le pueden plantear muchas alternativas de investigación para que, dentro de un tiempo, se pueda masificar y promover su uso.

10 ANEXOS

ANEXO I. CONFIGURACIÓN DE MININET

Una vez que ya la máquina virtual funciona es recomendable ver si las dos interfaces activadas tienen asignadas sus correspondientes IPs. Para ello se ejecutará la aplicación del terminal y ejecutaremos el código de la Figura 28. Una vez hecho esto, aparecerán las interfaces y deben estar configuradas.

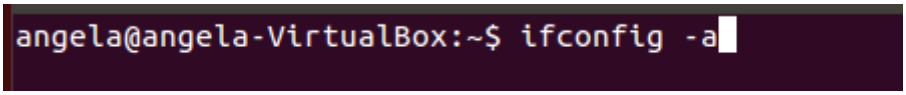


Figura 28. Mostrar interfaces

Si es así se procede con la instalación de Mininet. En su página web te dan varios métodos para que lo instales, en este caso se ha usado el caso dos, ya que la instalación por medio del terminal aporta muchas más funcionalidades. Como se muestra en la Figura 29. lo primero que hay que hacer es obtener el programa git y lo haremos mediante el comando que aparece en la figura. Esta aplicación nos permitirá la extracción del código fuente desde github.

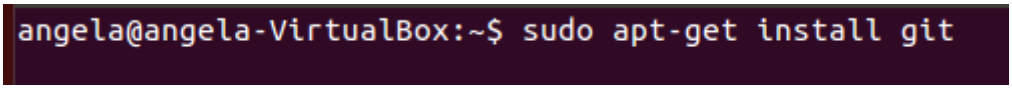


Figura 29. Instalación de git

Una vez hecho esto, se puede proceder a obtener el código fuente, la sentencia a utilizar es la que aparece en la Figura 30.

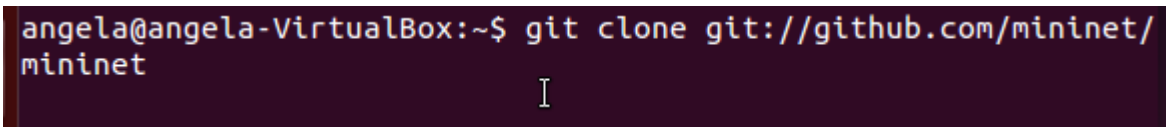


Figura 30. Obtención del código fuente de Mininet

En este caso se está usando la versión 2.2.2 de Mininet, para la instalación se ejecutará el comando de la Figura 31. Se elige utilizar la opción -a ya que así se instala todo lo que incluye la VM Mininet, incluyendo dependencias como Open vSwitch, así como las funciones de Wireshark OpenFlow y POX.

```
angela@angela-VirtualBox:~$ mininet/util/install.sh -a
```

Figura 31. Instalación de la VM Mininet

Una vez finalizada la instalación es recomendable que se pruebe la funcionalidad básica de Mininet utilizando el comando que aparece en la Figura 32.

```
angela@angela-VirtualBox:~$ sudo mn --test pingall
```

Figura 32. Probar que funciona Mininet

Para poder acceder al editor gráfico hay que introducir la serie de comandos que aparecen en la Figura 33.

```
angela@angela-VirtualBox:~$ cd mininet
angela@angela-VirtualBox:~/mininet$ ls
bin           debian    INSTALL   mininet.egg-info  mnexec.c
build         dist      LICENSE   mn.1              README.md
CONTRIBUTORS  doc       Makefile  mnexec            setup.py
custom        examples  mininet   mnexec.1          util
angela@angela-VirtualBox:~/mininet$ cd examples
angela@angela-VirtualBox:~/mininet/examples$ ls
baresshd.py       __init__.py         numberedports.py
bind.py           intfoptions.py      popenpoll.py
clustercli.py     limit.py            popen.py
clusterdemo.py    linearbandwidth.py  README.md
cluster.py        linuxrouter.py      scratchnet.py
clusterSanity.py  miniedit.py         scratchnetuser.py
consoles.py       mobility.py         simpleperf.py
controllers2.py   multilink.py        sshd.py
controllers.py    multiping.py        test
controlnet.py     multipoll.py        tree1024.py
cpu.py            multitest.py        treeping64.py
emptynet.py       natnet.py           vlanhost.py
hwintf.py         nat.py
angela@angela-VirtualBox:~/mininet/examples$ ./miniedit.py
MiniEdit running against Mininet 2.2.2
```

Figura 33. Ejecutar Miniedit

Una vez ejecutado este comando aparecerá la aplicación gráfica de Mininet que aparece en la Figura 34.

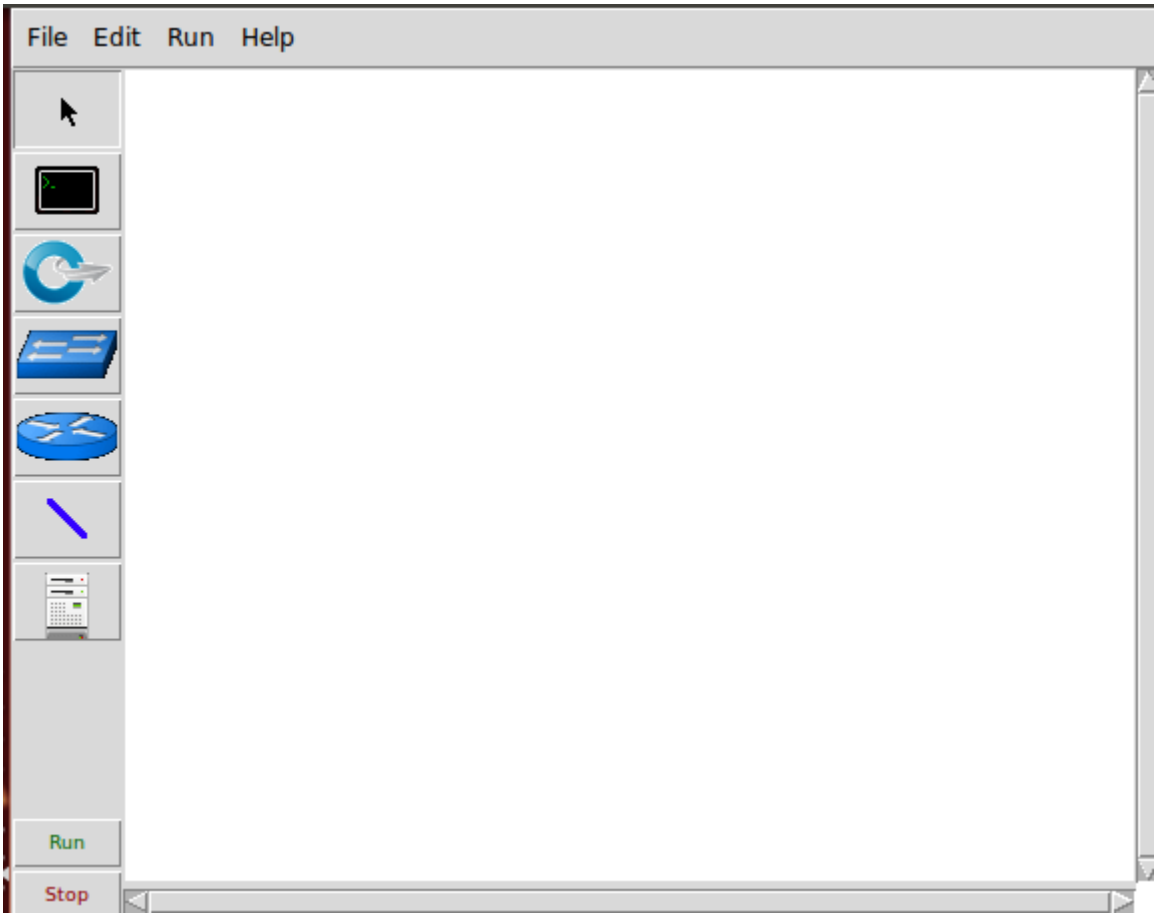


Figura 34. Miniedit

Siempre y cuando entremos al terminal de la máquina virtual y queramos acceder al entorno de Mininet el comando que se ha de ejecutar es:

$ sudo mn [parametros]

Esta sentencia tiene una serie de parámetros que se pueden utilizar con diferentes fines.

- -h → Muestra los comandos CLI de Mininet
- -c → Borra el entorno y sale.
- --topo minimal → Crea la topología por defecto que invoca el entorno Mininet. La topología consta como se muestra en la Figura 35 de un controlador, un switch OpenFlow y dos hosts.

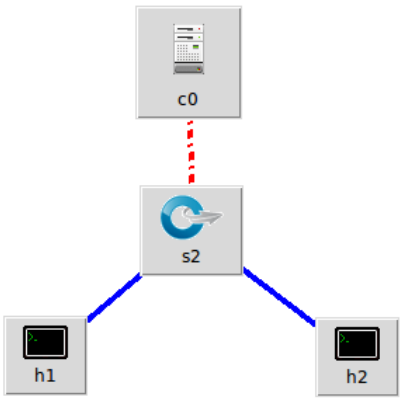


Figura 35. Topologia mínima

Según la ayuda mostrada con el comando sudo mn -h podemos definir topologías sencillas a través de llamadas a Mininet. A continuación, se describirán las topologías que se pueden invocar.

- sudo mn --topo single, [n] → Siendo n el número de hosts que están conectados al switch OpenFlow como se muestra en la Figura 36.

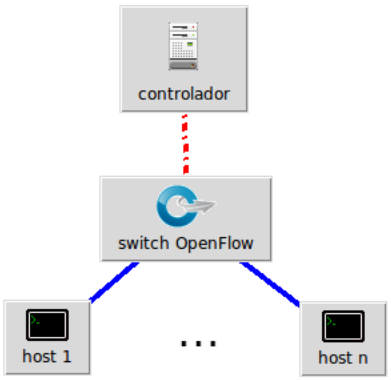


Figura 36. Topo single[n]

- sudo mn --topo linear [n] → Siendo n el números de switches OpenFlow conectados entre sí. Además esta topología consta de un host conectado a cada switch.

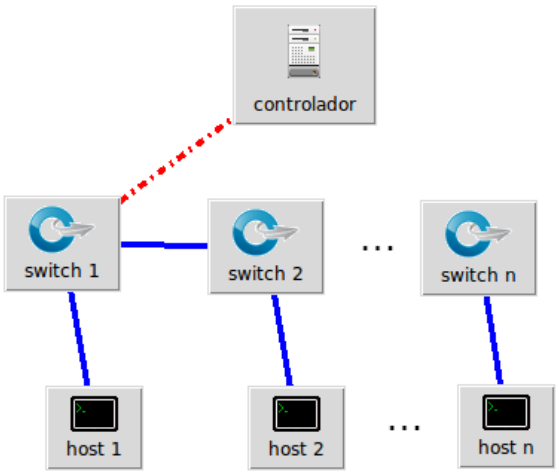


Figura 37. Topología Linear

- sudo mn --topo tree, depth=n → Implementa una topología tipo árbol, n se refiere a la profundidad, los niveles se comportan de tal manera que a cada switch del nivel superior se le conectarán dos en el nivel inferior. En la Figura 38 se muestra un ejemplo.

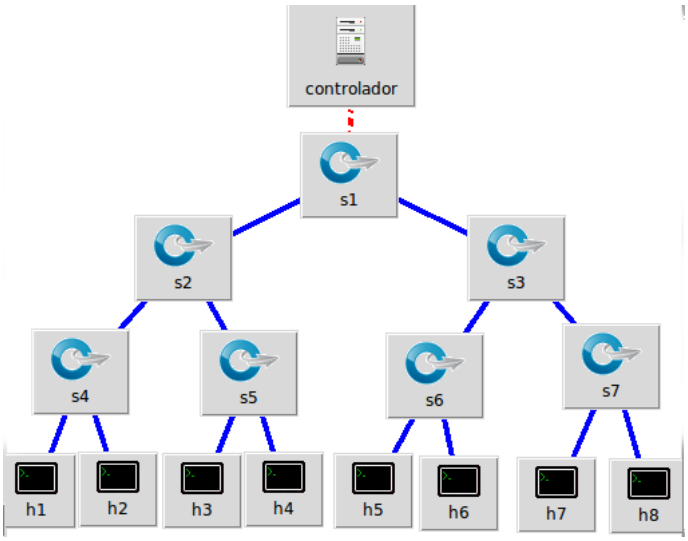


Figura 38. Topología en árbol

En caso de querer implementar una topología más complicada, también es posible implementarlas a través de scripts python.

En Mininet hay una serie de comandos que muestran información sobre la topología que hemos implementado. A continuación, se enumeran los comandos y cuál es su función:

➢ Nodes

Muestra los nodos disponibles

➢ Net

Muestra la información sobre la red, nodos y enlaces disponibles

- Dump

  Muestra información sobre las direcciones IP de cada nodo.
- h1 ifconfig -a

  Ejecuta el comando ifconfig -a desde el host h1.
- --mac

  Hace que las direcciones MAC de los hosts coincidan con sus direcciones IP.
- iperf src dest

  Establece sesiones TCP o UDP para calcular el ancho de banda entre dos hosts, si no se ponen src ni dest lo hace desde el primer al último host.
- --link

  Permite ajustar valores de los enlaces. Un ejemplo es:

  sudo mn --link tc, bw=80, delay 10 ms

  Con esta sentencia se limita la velocidad de los enlaces a 80 Mbps y además con un retardo de 10 ms.

- up/down → permite habilitar o deshabilitar un enlace. Por ejemplo;

  link s1 h1 down

  link s1 h1 up

ANEXO II.

A través de este apéndice se verán los pasos a seguir para convertir un controlador de un concentrador en un switch de aprendizaje de flujo y esto se aplicará a la topología de la Figura 20. Además, se mostrará el entorno de desarrollo en el cuál, se presentarán las herramientas que después serán útiles para convertir el hub en un switch de aprendizaje.

A continuación, se define una terminología que posteriormente se verá, los tipos de terminales son:

- **Terminal VirtualBox**: se conecta a OpenFlow. Este es crea cuando se inicia la VM.
- **Terminal SSH**: se conecta a OpenFlow. Se crea usando PuTTY en Windows o SSH en OS X/Linux.
- **Terminal Xterm**: se conecta a un host en la red virtual. Se crea cuando se inicia la red virtual.

La VM de OpenFlow incluye una serie de utilidades de red específicas y además otras específicas preinstaladas de OpenFlow.

- **Controlador OpenFlow**: Se encuentra en la parte superior de la interfaz de OpenFlow. Actúa como un conmutador de aprendizaje Ethernet en combinación con un conmutador OpenFlow. Cuando se ejecute se podrán ver los mensajes que se envían.
- **Switch OpenFlow**: Se encuentra en la parte inferior de la interfaz de OpenFlow. OpenvSwitch es otro software, pero además es un switch basado en el núcleo.
- **Ovs-ofctl**: línea de comando que se utiliza para para enviar mansajes rápidos de OpenFlow. Por otro lado, es muy útil para ver el puerto del switch y las estadísticas de flujo o insertar manualmente las entradas de flujo en la tabla de flujo.
- **Wireshark**: utilidad gráfica que se usa para ver paquetes. OpenFlow incluye un disector Wireshark en su referencia, para analizar los mensajes OpenFlow enviados al puerto por defecto de OpenFlow (6633) de manera legible.
- **Iperf**: utilidad general de la línea de comandos para probar la velocidad de una sola conexión TCP.
- **Mininet**: plataforma de emulación de red. Mininet crea una red virtual OpenFlow que puede contener un controlador, conmutadores, hosts y enlaces.
- **Cbench**: es una utilidad para probar la velocidad de ajusto de flujo de los controladores OpenFlow.

En la Figura 39 se detallan los pasos a seguir para la ejecución del tutorial OpenFlow.

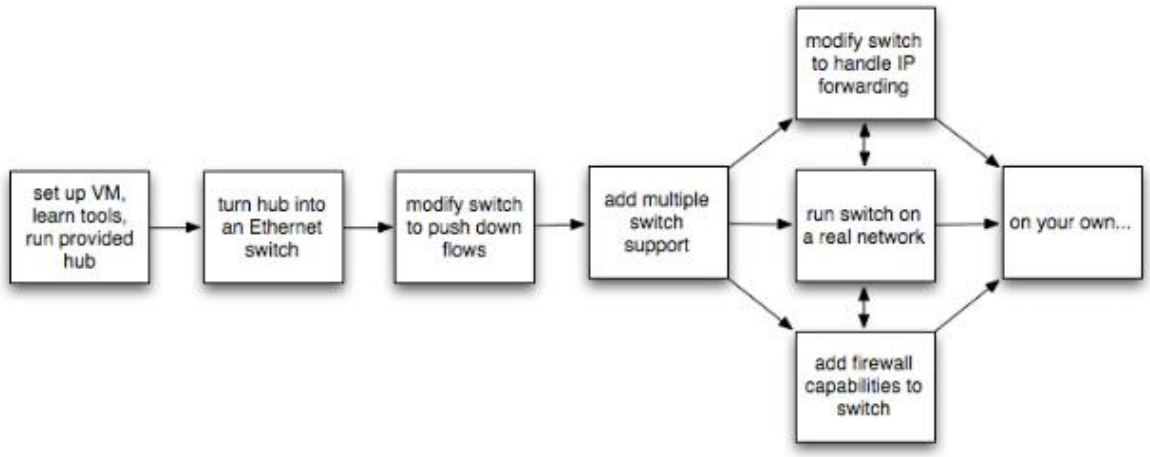


Figura 39. Tutorial OpenFlow

Como se habla en el punto 7 de este TFG en Mininet, antes de nada, para crear esa topología en particular en la VM, en un terminal SSH hay que ingresar el siguiente comando:

$ Sudo mn --topo single,3 --mac --switch ovsk --controller remote

Esta instrucción indica a Mininet que inicie la topología con 3 hosts, un switch basado en OpenvSwitch y que establezca la dirección MAC de cada host igual a su IP y apunte a un controlador remoto que tenga como valor predeterminado el local host. Para establecer la topología citada anteriormente en Mininet hay que seguir estos pasos:

- Crear con una dirección IP diferente 3 hosts virtuales
- Crear un núcleo con 3 puertos en un software OpenFlow
- A través de un cable Ethernet virtual se conectarán los hosts al switch.
- Se establecen direcciones MAC en cada host correspondiente con sus direcciones IP.
- El controlador remoto se conectará al switch a través de una configuración en el switch OpenFlow

APARTADO A

Para poder analizar la configuración de las topologías con diferentes componentes propios de OpenFlow es imprescindible familiarizarse con el entorno y las herramientas de las que se dispone y empezar a hacer pruebas para comprender su funcionamiento.

Para ver la lista de nodos disponibles, en la consola Mininet se debe ejecutar:

```
Mininet>  nodes
```

Para una lista de comandos disponibles, en la consola Mininet se debe ejecutar:

```
Mininet>  help
```

Para ver la información sobre la red, nodos y enlaces disponibles, en la consola Mininet se debe ejecutar:

```
Mininet>  net
```

Para ver la información sobre las direcciones IP de cada nodo, en la consola Mininet se debe ejecutar:

```
Mininet>  dump
```

Para ejecutar un solo comando en un nodo, se pone justo antes el nombre del nodo antes del comando a ejecutar. Por ejemplo, para comprobar la IP de un host virtual, en la consola Mininet, se debe ejecutar:

```
Mininet>  h1 ifconfig
```

La mejor alternativa para ejecutar comandos interactivos y ver la salida de depuración, es generar un xterm para uno o más hosts virtuales. En la consola Mininet, se debe ejecutar:

```
Mininet>  xterm  h1  h2
```

Para que las direcciones MAC de los hosts coincidan con sus direcciones IP se debe incluir en la línea de comando en la consola Mininet:

Mininet> --mac

Para ajustar los valores de los enlaces se debe incluir en la linea de comando –link. Un ejemplo de esto es:

Mininet> sudo mn –link tc, bw=80, delay 10 ms

Con esta sentencia se limita la velocidad de los enlaces a 80 Mbps y además con un retardo de 10 ms.

Para habilitar un enlace o deshabilitarlo se usa el comando up/down. Un ejemplo es escribir en la consola de Mininet:

Mininet> link s1 h1 down
Mininet> link s1 h1 up

Para abandonar la topología creada se puede usar el comando “exit” y para borrar cualquier estado residual o proceso se puede usar:

$ sudo mn -c

Nota: $ es para el terminal SSH (usuario normal) y # es para el terminal SSH (usuario root). Mininet tiene una gran cantidad de comandos y opciones de inicio para ayudar con la depuración.

APARTADO B. OVS-OFCTL.

Como ya se dijo antes se ejecutó en una ventana SSH **sudo mn –topo single,3 –mac –switch –controller remote**. Ahora se debe crear una segunda ventana SSH y ejecutar la siguiente sentencia:

$ ovs ofctl show s1

Al no anteponer “sudo” nos responde con permiso denegado ya que falla al abrir el socket.

```
angela@angela-VirtualBox:~$ ovs-ofctl show s1
ovs-ofctl: /var/run/openvswitch/s1.mgmt: failed to open socket
(Permission denied)
```

Figura 40. Permiso denegado

Por tanto, el comando correcto a introducir es el siguiente obteniendo la siguiente respuesta.

$ sudo ovs-ofctl show s1

```
angela@angela-VirtualBox:~$ sudo ovs-ofctl show s1
[sudo] password for angela:
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000000000000001
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP
_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC
 SET_DL_DST SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_
DST ENQUEUE
 1(s1-eth1): addr:d6:8a:1b:1a:bb:d9
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s1-eth2): addr:76:a2:e8:90:f7:1e
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 3(s1-eth3): addr:8a:81:28:39:84:39
     config:     0
     state:      0
     current:    10GB-FD COPPER
```

Figura 41. Respuesta al ejecutar como admin la sentencia

El comando show se conecta al switch y descarga su estado y capacidades de puerto.

Un comando que permite ver la tabla de flujo del switch es la que se puede ver a continuación, pero como en este momento no hemos iniciado el controlador debe estar vacía.

$ sudo ovs-ofctl dump-flows s1

Es muy importante saber que “ovs-ofctl” está hablando con una instancia local de OpenvSwitch a través de un socket de dominio Unix que está buscando por nombre. Si se quiere ejecutar un switch de referencia Stanford o un switch OpenFlow de hardware, habría que conectarse a un puerto TCP pasivo usando la siguiente línea de comando.

$ sudo ovs-ofctl dump-flows tcp: {dirección ip}: {puerto}
Ejemplo: **$ sudo ovs-ofctl dump-flows tcp: 127.0.0.1: 6634**

Hay que aclarar que {dirección IP} es la dirección IP de la interfaz de gestión el switch y {puerto} es el puerto de escucha/ gestión pasiva de OpenFlow. Si se llama a Mininet con el usuario de conmutación, se abrirá un puerto de escucha pasivo para cada conmutador en el puerto (6633+n) donde n es el número del conmutador. Esto permite que ovs-ofctl se utilice en comandos como el del ejemplo anterior.

APARTADO C. PRUEBA PING

Si desde la consola de Mininet se prueba a hacer ping desde h2 a h1 no alcanza el destino como se puede ver en la figura x. Esto ocurre porque como se vio anteriormente la tabla de flujo de conmutación está vacía. Además de eso no hay un controlador conectado al switch y por tanto el switch no sabe qué hacer con el tráfico entrante, dando lugar a un fallo de ping.

```
mininet> h1 ping -c3 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss,
 time 2012ms
pipe 3
```

Figura 42. Fallo del ping en Mininet

A continuación, en el terminal SSH se debe instalar manualmente los flujos necesarios a través del comando ovs-ofctl como se expone a continuación:

ovs-ofctl add-flow s1 in_port=1,actions=output:2
ovs-ofctl add-flow s1 in_port=2,actions=output:1

Estos comandos envían los paquetes que vienen del puerto 1 al 2 y viceversa. A continuación, se puede comprobar la tabla de flujo y se obtiene el resultado mostrado en la Figura 43.

```
root@angela-VirtualBox:/home/angela# ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=25.087s, table=0, n_packets=0, n_bytes=0,
 idle_age=25, in_port=1 actions=output:2
 cookie=0x0, duration=13.943s, table=0, n_packets=0, n_bytes=0,
 idle_age=13, in_port=2 actions=output:1
```

Figura 43. Tabla de flujo de S1

Ahora al haber configurado los flujos se puede ejecutar en la consola de Mininet el comando ping de nuevo y comprobar que ahora sí, llega a su destino como se muestra en la Figura 44

```
mininet> h1 ping -c3 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=9.42 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.453 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.460 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 0.453/3.444/9.420/4.225 ms
```

Figura 44. Ping transmitido correctamente

APARTADO D. INICIAR WIRESHARK

Wireshark por defecto no trae instalado el disector de OpenFlow y es necesario instalarlo ya que es extremadamente útil para ver mensajes de protocolo OpenFlow, así como la depuración en general.

Para la instalación del disector se han de seguir los siguientes pasos:

- **$ sudo apt-get install mercurial**
- **$ hg clone https://bitbucket.org/barnstorm/of-dissector**
- **$ cd of-dissector/src**
- **$ sudo apt-get install scons**
- **export WIRESHARK=/usr/include/wireshark/**

- **sudo scons install**

En el caso de que nos apareciera un mensaje de error como este:

```
scons: Reading SConscript files ...
### ERROR: You need to set the WIRESHARK environment
variable to the
location of your wireshark include directory.
### ERROR: (such that epan/packet.h is a valid include
path)
```

Figura 45. Error de la instalación del disector OpenFlow

Lo más seguro es abrir un nuevo terminal SSH y seguir los siguientes pasos:

- **$ sudo -s**
- **# rm -f /usr/lib/wireshark/libwireshark1/plugins/openflow.so**
- **# cd $home/of-dissector/src**
- **# export WIRESHARK=/usr/include/wireshark**
- **# scons install**

Los pasos anteriores crean una biblioteca compartida openflow.so. Este archivo se copia automáticamente en /usr/include/wireshark. El disector de OpenFlow se carga en Wireshark en modo de usuario. Si se intenta iniciar utilizando root-user, no vendrá el complemento cargado. Por lo tanto, para eso, es necesario colocar el binario en la ubicación /usr/lib/wireshark para que pueda cargarse automáticamente independientemente del usuario. Para conseguirlo se debe usar esta sentencia:

- **$ sudo cp openflow.so /usr/lib/wireshark/libwireshark3/plugins/openflow.so**

Después se debe abrir Wireshark con el comando **$ sudo wireshark &.** Después se hará click en Capturar para la interfaz ‘lo’, que es la interfaz de loopback. Para filtrar los paquetes OpenFlow es necesario escribir en el cuadro de Filtro ‘of’ y pulsar Intro.

APARTADO E. INICIAR EL CONTROLADOR Y VER LOS MENSAJES EN WIRESHARK

Con el disector de Wireshark escuchando, se inicia el controlador de referencia OpenFlow. En el terminal SSH con la sentencia:

$ controller ptcp:

Se inicia un controlador simple que actúa como un switch de aprendizaje sin instalar entradas de flujo. Después de ejecutar esta sentencia se pueden observar muchos tipos de mensajes OpenFlow. A continuación una tabla que resume algunos de ellos:

Mensaje	Tipo	Descripción
Hello	**Controlador → Switch**	**Una vez establecido el handshake de TCP, el controlador envía su número de versión al switch**
Hello	**Switch → Controlador**	**El switch contesta con su número de versión soportada**
Features Request	**Controlador → Switch**	**El controlador consulta cuales son los puertos disponibles**
Set Config	**Controlador → Switch**	**El controlador consulta al switch para enviar expiraciones de flujos**
Port Status	**Switch → Controller**	**El switch responde con una lista de puertos, velocidades de puerto y tablas y acciones soportadas.**

Tabla 2. Mensajes OpenFlow

Dado que todos los mensajes se reenvían a través de localhost cuando se utiliza Mininet, determinar quién es el remitente de un mensaje es confuso cuando hay una gran cantidad de switches emulados. Sin embargo, esto no supone un problema cuando hay un solo switch. El controlador está en el puerto OpenFlow estándar (6633), mientras que el switch está en otro puerto de nivel de usuario.

APARTADO F. MENSAJES OPENFLOW PARA PING

En este apartado es necesario habilitar los mensajes echo-request/reply. Para ver estos mensajes es necesario escribir en el filtro de Wireshark el siguiente comando:

```
of and not (of10.echo_request.type or of10.echo_reply.type)
```

En caso de tener una versión anterior de Wireshark, se deberá usar otra sentencia como la siguiente:

```
of && (of.type !=3) && (of.type !=2)
```

Después se deberá ejecutar un ping para ver los mensajes OpenFlow. Es recomendable limpiar antes la caché de ARP en los hosts, de lo contrario no se verán algunas peticiones/respuestas de ARP, ya que se utilizará la caché en su lugar. Los comandos a utilizar son:

Mininet> h1 ip -s -s neigh flush all
Mininet> h2 ip -s -s neigh flush all

Después de hacer el ping, los tipos principales de mensajes que se podrán visualizar son:

Mensaje	**Tipo**	**Descripción**
Packet-In	**Switch → Controlador**	**Se recibió un paquete que no coincide con ninguna entrada en la tabla de flujo del switch, haciendo que el paquete se envie al controlador**
Packet-Out	**Controlador → Switch**	**El controlador envía el paquete a uno o varios puertos del switch**
Flow-Mod	**Controlador → Switch**	**Le indica al switch que debe agregar un flujo particular a su tabla de flujo.**
Flow-Expired	**Switch → Controlador**	**Expiración del flujo después de un tiempo de inactividad.**

Tabla 3. Mensajes OpenFlow (2)

En las capturas de Wireshark se podrá ver una solicitud ARP perdida de la tabla de flujo, que genera un mensaje de difusión Packet-Out. Después se puede observar que llega la respuesta ARP; con ambas direcciones MAC, ahora conocidas por el controlador. De esta manera ahora es posible que el flujo vaya hacia el switch a través de un mensaje de tipo Flow-Mod. A continuación, el switch envía los paquetes ICMP para los flujos. Las solicitudes ping siguientes van directamente a través del camino y no proporcionan ningún mensaje adicional.

A continuación, para comprobar los tiempos de expiración y compararlos, a través de la consola Mininet, se debe ejecutar este comando tantas veces como sea necesario.

Mininet> h2 ping -c3 h3

APARTADO G. CONTROLADOR DE REFERENCIA CON IPERF.

Iperf es una herramienta de línea de comandos para comprobar las velocidades entre dos equipos. Para ver el resultado de usar esta herramienta en la consola Mininet se debe ejecutar:

Mininet> iperf

Este comando ejecuta un servidor TCP iperf en un host virtual, y a continuación ejecuta un cliente iperf en un segundo host virtual. Una vez conectados, intercambian los paquetes entre sí y devuelven los resultados. Para poner a prueba esta herramienta se siguen los siguientes pasos

- **Mininet> iperf**
- **Mininet> exit**
- **$ sudo mn --topo single,3 --mac --controller remote --switch user**
- **Mininet> iperf**
- **Mininet> exit**

11. SIGLAS Y ABREVIATURAS

API	Application Programmer Interface
ASIC	Application-Specific Integrated Circuit
CAM	Content-Addressable Memory
CE	Control Element
CLI	Command-Line Interface
COPS	Common Open Policy Service
CPU	Central Processing Unit
CSMA/CD	Carrier-Sense Multiple-Access/Collision Detect
DCAN	Devolved Control of ATM Networks
FE	Forwarding Element
ForCES	Forwarding and Control Element Separation
Gbps	Gigabits Per Second
GSMP	General Switch Management Protocol
HTTP	HyperText Transfer Protocol
IDS	Intrusion Detection System
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IPS	Intrusion Prevention System
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
LAN	Local Area Network
LIFO	Last-In/First-Out
MAC	Media Access Control
Mbps	Megabits Per Second
MPLS	Multiprotocol Label Switching
MSTP	Multiple Spanning Tree Protocol
NAC	Network Access Control
NEM	Network Equipment Manufacturer
ONF	Open Networking Foundation
OSI	Open Systems Interconnection
OVS	Open vSwitch
OVSDB	Open vSwitch Database Management Protocol
QoS	Quality of Service
RFC	Request for Comments
REST	Representational State Transfer

SDN	Software Defined Networking
SNMP	Simple Network Management Protocol
STP	Spanning Tree Protocol
TCAM	Ternary Content-Addressable Memory
TCP	Transmission Control Protocol
ToS	Type of Service
TTL	Time to Live
UDP	User Datagram Protocol
VLAN	Virtual Local Area Network
VM	Virtual Machine
VTEP	Virtual Tunnel Endpoint or VXLAN Tunnel Endpoint
VXLAN	Virtual Extensible Local Area Network

12.REFERENCIAS

[1] CAMPBELL, ANDREW T., ET AL. Open signaling for ATM, internet and mobile networks (OPENSIG'98). *ACM SIGCOMM Computer Communication Review*, 1999, vol. 29, no 1, p. 97-108.

[2] TENNENHOUSE, DAVID L., et al. A survey of active network research. *IEEE communications Magazine*, 1997, vol. 35, no 1, p. 80-86.

[3] NUNES, BRUNO ASTUTO A., ET AL. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials*, 2014, vol. 16, no 3, p. 1617-1634.

[4] NEWMAN, PETER; MINSHALL, GREG; LYON, THOMAS L. IP switching-atm under IP. *IEEE/ACM Transactions on networking*, 1998, vol. 6, no 2, p. 117-129.

[5] DAVIE, BRUCE S.; REKHTER, YAKOV. *MPLS: technology and applications*. Morgan Kaufmann Publishers Inc., 2000.

[6] RIGNEY C, WILLENS S, RUBENS A, SIMPSON W. Remote authentication dial-in user service (RADIUS), RFC 2865. Internet Engineering Task Force; 2000.

[7] DURHAM D, BOYLE J, COHEN R, HERZOG S, RAJAN R, SASTRY A. The COPS (Common Open Policy Service) protocol, RFC 2748. Internet Engineering Task Force; January 2000.

[8] FERRO G. Automation and orchestration. Network computing. [en línea]. Actualizada: 8 septiembre 2014. [Fecha de consulta: 22 septiembre 2017]. Disponible en: www.networkcomputing.com/private-cloud-tech-center/automation-and-orchestration/231600896 >.

[9] KUMAR, SANJAY, ET AL. VMANAGE: loosely coupled platform and virtualization management in data centers. En *Proceedings of the 6th international conference on Autonomic computing*. ACM, 2009. p. 127-136.

[10] DORIA A, HADI SALIM J, HAAS R, KHOSRAVI H,WANGW, DONG L, ET AL. Forwarding and control element separation (ForCES) protocol specification, RFC 5810. Internet Engineering Task Force; 2010.

[11] GREENBERG A, HJALMTYSSON G, MALTZ D, MYERS A, REXFORD J, XIE G, ET AL. A clean-slate 4D approach to network control and management. ACM SIGCOMM Comput Commun Rev 2005;35(3).

[12] UNIVERSITY OF CAMBRIDGE. Devolved Control of ATM Networks. [en línea]. Actualizada: 2015 [Fecha de consulta: 22 septiembre 2017]. Disponible en: www.cl.cam.ac.uk/research/srg/netos/old-projects/dcan >.

[13] WORSTER, T.; DORIA, A.; BUERKLE, J. General Switch Management Protocol (GSMP) Packet Encapsulations for Asynchronous Transfer Mode (ATM), Ethernet and Transmission Control Protocol (TCP). 2002.

[14] TENNEHOUSE D, SMITH J, SINCOSKIE W, WETHERALL D, MINDEN G. A survey of active network research. IEEE Commun Mag 1997;35(1):80–6.

[15] TENNEHOUSE D, WETHERALL D. Towards an active network architecture. In: Proceedings *of the DARPA active networks conference and exposition. IEEE; 2002. p. 2–15.*

[16] GERKEY, BRIAN P., ET AL. Most valuable player: A robot device server for distributed control. En *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*. IEEE, 2001. p. 1226-1231.

[17] MCKEOWN, NICK. Software-defined networking. *INFOCOM keynote talk*, 2009, vol. 17, no 2, p. 30-32.

[18] VMWARE VCENTER. Centralized visibility, proactive management and extensibility for VMware vSphere form a single console. [en línea]. Actualizada: 2017 [Fecha de consulta: 22 septiembre 2017]. Disponible en www.vmware.com/products/vcenter-server.html.

[19] VMWARE VMOTION. vSphere Enhanced Application Performance and Availability. [en línea]. Actualizada: 2017 [Fecha de consulta: 22 septiembre 2017]. Disponible en: www.vmware.com/products/vsphere/enhanced-app-perfomance.html

[20] OPENDAYLIGHT. Industry leaders collaborate onOpenDaylight project, donate key technologies to accelerate software-defined networking. OpenDaylight [en línea]. Actualizada: Abril 2015.[Fecha de consulta: 15 agosto 2017] www.opendaylight.org/announcements/2013/04/industry-leaders-collaborate-opendaylight-project-donate-key-technologies.

[21] DIXIT, ADVAIT, ET AL. Towards an elastic distributed SDN controller. En *ACM SIGCOMM Computer Communication Review*. ACM, 2013. p. 7-12.

[22] PFAFF, BEN; DAVIE, BRUCE. The open vSwitch database management protocol. 2013.

[23] GOLDMAN, KENNETH; PEREZ, RONALD; SAILER, REINER. Linking remote attestation to secure tunnel endpoints. En *Proceedings of the first ACM workshop on Scalable trusted computing*. ACM, 2006. p. 21-24.

[24] WALLNER, RYAN; CANNISTRA, ROBERT. An SDN approach: quality of service using big switch's floodlight open-source controller. *Proceedings of the Asia-Pacific Advanced Network*, 2013, vol. 35, p. 14-19.

[25] ROWLAND, CRAIG H. *Intrusion detection system*. U.S. Patent No 6,405,318, 11 Jun. 2002.

[26] ESTAN, CRISTIAN, ET AL. Building a better NetFlow. En *ACM SIGCOMM Computer Communication Review*. ACM, 2004. p. 245-256.

[27] SCHULTZ, KENNETH JAMES, ET AL. *Content addressable memory*. U.S. Patent No 5,859,791, 12 Ene. 1999.

[28] MATHEW, BINU K.; ISSAC, NEETHU ANNA. Ternary Content Addressable Memory.

[29] BASTA, ARSANY, ET AL. HYPERFLEX: Demonstrating control-plane isolation for virtual software-defined networks. En *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015. p. 1163-1164.

[30] BHATIA, SAPAN, ET AL. Trellis: A platform for building flexible, fast virtual networks on commodity hardware. En *Proceedings of the 2008 ACM CoNEXT Conference*. ACM, 2008. p. 72.

[31] GYIRES, TIBOR. Software Defined Networking; OpenFlow.

[32] VIRTUALBOX.Portable-VirtualBox|Download of Portable-VirtualBox 2017 [en línea]. Actualizada: 2017 [Fecha de consulta: 22 septiembre 2017]. Disponible en: http://www.vbox.me/

[33] N. MCKEOWN, T. ANDERSON, H. BALAKRISHNAN, G. PARULKAR, L. PETERSON, REXFORD, J. SHENKER SCOTT, J . TURNER . “OpenFlow: Enabling Innvotacion in Campus Networks”. Marzo, 2008

[34] MININET PYTHON API REFERENCE MANUAL: MININET.NET.MININET CLASS REFERENCE [en línea]. Actualizada: 2017 [Fecha de consulta: 22 septiembre 2017]. Disponible en: http://mininet.org/api/classmininet_1_1net_1_1Mininet.html#aecca42852bcd3f0ef885fa53c07dd2a1