Universidad de Oriente

Facultad de Ingeniería Eléctrica

Departamento de Telecomunicaciones

TRABAJO DE DIPLOMA

Conmutadores OpenFlow para Redes Definidas por Software (SDN). Funcionalidades.

Autor: Manuel José Martínez Rodríguez

Tutor: M.Sc. Lidice Romero Amondaray

Santiago de Cuba

Junio, 2015

Universidad de Oriente

Facultad de Ingeniería Eléctrica

Departamento de Telecomunicaciones

TRABAJO DE DIPLOMA

Conmutadores OpenFlow para Redes Definidas por Software (SDN). Funcionalidades.

Autor: Manuel José Martínez Rodríguez

Tutor: M.Sc. Lidice Romero Amondaray

Master en Ciencias, Departamento de Telecomunicaciones, Facultad de Ingeniería Eléctrica, e-mail: lidice@fie.uo.edu.cu

Santiago de Cuba

Junio, 2015

COMPROMISO DEL AUTOR

Hago constar que el presente trabajo de diploma es de mi autoría exclusivamente, no constituyendo copia de ningún trabajo realizado anteriormente y las fuentes usadas para la realización del trabajo se encuentran referidas en la bibliografía. Doy mi consentimiento a que el mismo sea utilizado por la Institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos, ni publicados sin autorización del Tutor o Institución.

_____

Firma del Autor

PENSAMIENTO

Yo solo sé que no se nada.

Sócrates

DEDICATORIA

A mi mamá, espero que se sienta orgullosa de su hijo…

A mí mismo, aunque sé que no me lo merezco…

AGRADECIMIENTOS

Agradecer a mi papá, a mis abues Roselva y Oralia, a mi tiamami Zory, a Yamilis y a mis tíabuelas, los que me apoyaron para culminar este reto…

A mi herma Male, por ser quien es y por ser como es, por ser súper importante para mí…

A mis profesores y compañeros de aula, a los exigentes y a los muy exigentes, a los que me vieron y a los que no me vieron, a los que me conocieron y a los que no me conocieron….Gracias…

RESUMEN

Las Redes Definidas por Software (SDN, *Software Defined Networks*), surge como una necesidad ante las dificultades que presentan las redes actuales al introducir nuevos servicios, debido a que están formadas por estructuras estáticas, con hardware de fabricantes propietarios. SDN es una tecnología que consiste en desacoplar el plano de control del plano de datos, al utilizar un controlador capaz de gestionar la información de reenvío en uno o más conmutadores. La comunicación entre controlador y conmutador se realiza a través de protocolos de control, los cuales están implementados en los conmutadores, enrutadores y demás dispositivos de red de los principales fabricantes.

En el presente trabajo se analizan los elementos fundamentales sobre las Redes Definidas por Software y el protocolo de control OpenFlow, para luego adentrarse en el análisis de uno de los elementos principales de este tipo de redes, los conmutadores SDN que soportan el protocolo OpenFlow; se analizan las particularidades de estos dispositivos, sus principios de funcionamiento y las similitudes que tienen con los conmutadores Ethernet tradicionales.

Palabras clave: SDN, OpenFlow, conmutadores.

ABSTRACT

The Software Defined Network (SDN, Redes Definidas por Software),emerges as a necessity due to the difficulties presented by the current networks in introducing new services, because they are composed of static structures, with hardware of proprietary manufacturers. SDN is a technology that consists in decouple the control path of the data path, using a controller capable to manage the forwarding information in one or more switch. The communication between controller and switch comes true through control protocols, which are implemented in the switch, router and other networks devices of the main manufacturers.

In the present work we analyzed the fundamental elements on the Software Defined Network and the control protocol OpenFlow, with the purpose of later on analyzed one of the main elements of this type of network, the SDN OpenFlow switch; this thesis focuses on the analysis of the particularities of these devices, in its principles of functioning and in the similitudes that they have with the Ethernet swith.

Keywords: *SDN, OpenFlow, switch.*

ÍNDICE

INTRODUCCIÓN

Las Redes Definidas por Software (SDN) es una nueva tecnología de red que ha surgido y se encuentra en desarrollo. Se percibe como una de las innovaciones más importantes en la creación de redes desde la introducción de MPLS (*Multiprotocol Label Switching*: por sus siglas en inglés).

Las redes de nueva generación actualmente son punto de convergencia de tecnologías que ofrecen servicios de telecomunicaciones y se propagan por el mundo. La explosión de los dispositivos móviles, la virtualización de servidores, la llegada de los servicios de la nube y al mismo tiempo, la modificación del patrón de tráfico por parte de los usuarios hacia los centros de datos, son algunas de las innovaciones que han impulsado a la industria de las TIC (Tecnologías de la Información y las Comunicaciones) a reexaminar las tecnologías de red tradicionales.

Desde hace algunos años, la gestión de redes y servicios de telecomunicaciones se ha convertido en una creciente y compleja tarea como resultado de la existencia de varios tipos de redes y servicios. A medida que las redes se han tornado más grandes y complejas, el costo de la gestión ha aumentado.

Por su parte, las arquitecturas tradicionales de redes no son óptimas a la hora de satisfacer todos los requerimientos de las empresas y los usuarios finales. Los diseños y equipos utilizados tradicionalmente en las redes de telecomunicaciones, aunque funcionan correctamente, no están alineados con los objetivos de negocio ni con la lógica de las aplicaciones, sino que forman una estructura cerrada y estática que no se puede adaptar en tiempo real a la demanda de las aplicaciones.

En ese contexto, las SDN se presentan como un nuevo paradigma para cubrir estas necesidades y promete una transformación de las arquitecturas de red y de la gestión de las redes como se conoce en la actualidad. Las SDN han surgido como un enfoque para fomentar la innovación en la red a través de una mayor flexibilidad, capacidad de programación, gestión y rentabilidad. Su objetivo es hacer la gestión de la red más inteligente y automatizada con el fin de disminuir los gastos de operaciones y capitales.

Antecedentes del problema

Varios de los grandes centros de datos del mundo, en la actualidad, están implementados con la tecnología de redes definidas por software por las ventajas que esta tecnología representa ante las redes tradicionales. En la Universidad de Oriente se inauguró un centro de datos, en el pasado mes de Mayo, que dará servicio a los usuarios de esta alta casa de estudio y al del resto de las universidades del oriente del país [1]. Una vez que esté en funcionamiento, y para prestar servicios a la altura de los centros de datos en países desarrollados, se tendrá que migrar a redes definidas por software para manejar el aumento exponencial de nuevos suscriptores y servicios y para flexibilizar el control de la red. Es importante entonces que los especialistas que trabajan en el centro de datos de la Universidad de Oriente conozcan las particularidades de esta tecnología y la forma en que pueden utilizarla. Para implementar SDN, se deben reemplazar los dispositivos de red tradicionales por dispositivos SDN que soporten un protocolo que permita la administración y la gestión desde un dispositivo controlador remoto, facilitando el trabajo de los miembros del equipo de administración del centro de datos. Entre estos dispositivos se encuentran los conmutadores SDN OpenFlow, siendo OpenFlow el protocolo de control que se utiliza para administrarlos remotamente.

Problema a resolver

¿Qué similitudes existe entre los conmutadores OpenFlow y los conmutadores tradicionales en cuanto a funcionalidades?

Objeto de estudio

Las Redes Definidas por Software (SDN).

Campo de acción

Conmutadores SDN OpenFlow.

Objetivo general

Emular un conmutador OpenFlow usando Mininet para compararlo, en cuanto a sus funcionalidades, con los tradicionales.

Objetivos específicos

- Realizar un análisis del estado del arte de la tecnología SDN.
- Caracterizar los conmutadores SDN OpenFlow, compararlos con los conmutadores tradicionales.

- Emular un conmutador SDN OpenFlow que realice funciones similares a los conmutares tradicionales.

CAPITULO 1 . REDES DEFINIDAS POR SOFTWARE

1.1 Introducción

Antes de comenzar el estudio de los conmutadores SDN OpenFlow, se hace necesario, tener un conocimiento básico acerca de la tecnología SDN y del protocolo OpenFlow.

El concepto de Redes Definidas por Software ha despertado recientemente gran interés en el mundo de las redes de telecomunicaciones debido a su potencial para proveer el control dinámico y flexible de la red, la posibilidad de programación y las funciones adecuadas de reenvío y procesamiento de tráfico. El vertiginoso desarrollo tecnológico, y en especial el advenimiento de las SDN, han significado un enorme y necesario avance en las telecomunicaciones.

En la implementación de las SDN, se utilizan protocolos de control que permiten la comunicación entre controlador y dispositivos de red gestionados. Entre los protocolos desarrollados el más popular es el protocolo OpenFlow, que externamente maneja y configura las tablas de reenvío de los dispositivos de red.

En este capítulo se hace una introducción a las condiciones que dieron lugar al desarrollo de las SDN y un análisis de esta tecnología para hacer más factible el entendimiento del tema central de esta tesis y ayudar al lector en el descubrimiento de uno de los conceptos más importantes en la rama de las redes de computadoras de los últimos años.

1.2 Necesidad de una nueva arquitectura de red

Satisfacer las necesidades actuales de servicios es prácticamente imposible con las tecnologías de redes tradicionales. Las redes tradicionales están construidas con dispositivos con propósitos especiales y específicos que corren con protocolos distribuidos que proveen la funcionalidad de descubrimiento de la red, enrutamiento, monitorización de tráfico, y control de acceso. Estos dispositivos presentan una alta integración del plano de control junto al de datos, y los operadores de red deben de configurar de manera separada cada uno de los protocolos que corren en cada uno de los dispositivos de manera individual, SDN posibilita que esto se haga de manera distinta separando el plano de control del plano de datos.

SDN es capaz de simplificar aplicaciones existentes y también servir como plataforma para el desarrollo de nuevas. Por ejemplo, para implementar el enrutamiento sobre el camino más corto, el controlador puede calcular las reglas de reenvío para cada conmutador corriendo el algoritmo Dijkstra en el gráfico de la topología de la red en vez de usar un protocolo distribuido de mucha mayor complejidad. Para conservar energía el controlador puede selectivamente apagar enlaces o inclusive conmutadores enteros luego de dirigir el tráfico hacia otras rutas. Para hacer cumplir políticas de control de acceso, el controlador puede consultar un servidor de autentificación externo e instalar caminos fijos de reenvío para cada uno de los flujos. Para balancear la carga entre servidores *back-end* de los centros de datos, el controlador puede dividir el flujo sobre otros servidores replicas y migrar flujos hacia nuevos caminos en respuesta a la congestión.

Aunque SDN hace posible programar la red, esto no significa que sea más fácil. Para soportar múltiples tareas al mismo tiempo, como enrutamiento y control de acceso, se hace extremadamente difícil, dado que las aplicaciones necesitan a la larga tener instaladas un grupo sencillo de reglas en el interior de los conmutadores. En adición, una red es un sistema distribuido, y todas las complicaciones usuales pueden llegar a darse. En general, el escribir aplicaciones para plataformas de controladores SDN de hoy en día es un ejercicio tedioso en los distribuidos bajos niveles de programación.

1.3 Características y arquitectura de las SDN

El grupo ONF (*Open Networking Foundation*, en español Fundación de Redes Abiertas) es el grupo que más relacionado está con el desarrollo y la estandarización de SDN. De acuerdo con la ONF las redes definidas por software emergen como una arquitectura dinámica, manejable, con efectividad en el costo, y adaptable, siendo ideal para grandes anchos de bandas y para la naturaleza dinámica de las aplicaciones de hoy en día. Esta arquitectura permite el desacople del control de la red y de las funciones de reenvío habilitando el control de la red a convertirse en directamente programable y que la infraestructura por debajo de este sea abstracta a las aplicaciones y a los servicios de red. De acuerdo con la ONF**,** la arquitectura SDN es [2]:

- Directamente programable: el control de la red es directamente programable porque está desacoplado de las funciones de reenvío.

- Ágil: la abstracción del control del reenvío permite a los administradores dinámicamente ajustar el ancho de los flujos de tráfico de la red para que tengan lugar los cambios pertinentes para suplir las necesidades.
- Manejo centralizado: la inteligencia de la red está (lógicamente) centralizada en el controlador SDN basado en software que mantiene una visión global de la red, que se presenta ante las aplicaciones como un único y lógico conmutador.
- Configuración programable: SDN permite a los administradores de red configurar, administrar, asegurar y optimizar los recursos de la red rápidamente de manera dinámica, mediante los programas SDN automáticos, los cuales pueden escribirse por sí mismos, porque esto programas no dependen de ningún software propietario.
- Base en estándares abiertos y vendedores neutrales: cuando la implementación es a través de estándares abiertos, SDN simplifica el diseño y la operación de las redes porque las instrucciones provienen de un controlador SDN en vez de provenir de dispositivos específicos y protocolos de múltiples vendedores propietarios.

SDN no constituye una tecnología sino más bien una arquitectura, tiene como aplicación a la virtualización de las redes, que a su vez tiene como mayor beneficio que provee soporte para la movilidad de máquinas virtuales independientemente de la red física. A continuación se ofrecen algunos conceptos claves que forman parte de la arquitectura de los sistemas SDN que se muestra en la Figura 1.1.

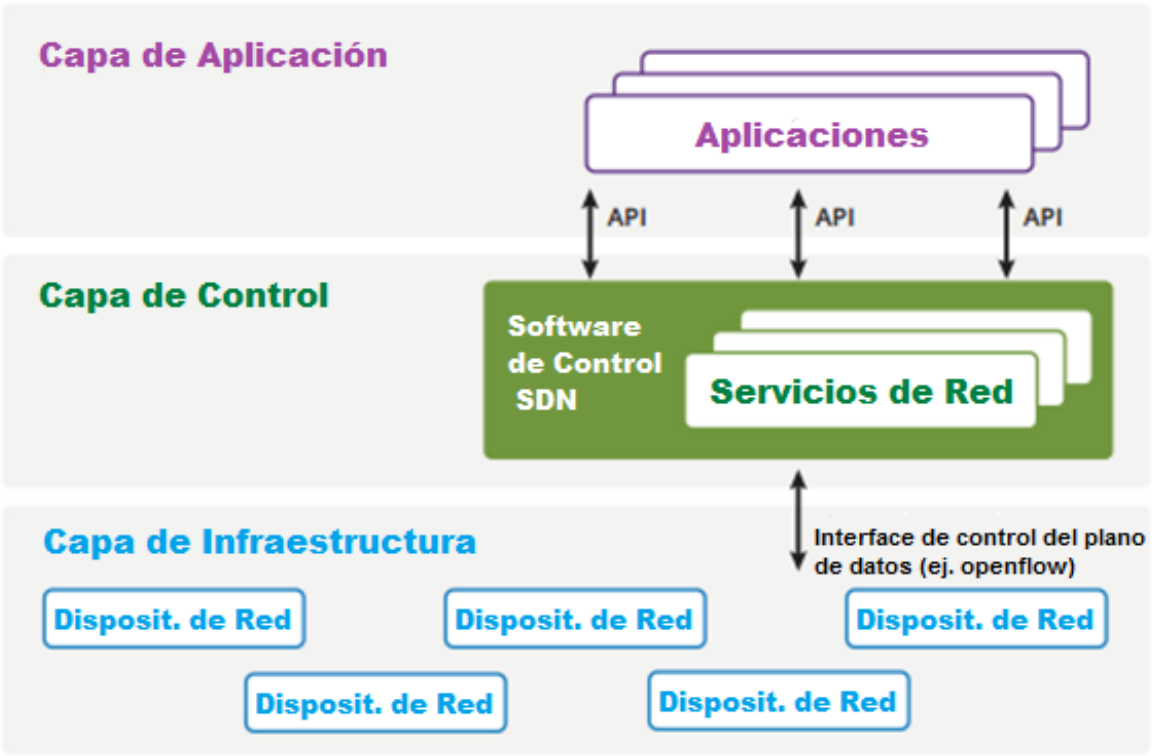

Figura 1.1 Arquitectura de SDN (Fuente:[2])

Aplicaciones

Se refiere a las aplicaciones que son directamente consumidas por los usuarios finales. Posiblemente incluya video conferencias, administración de cadenas de soporte, y administración de relación de clientes.

Servicios de red y de seguridad

Se refiere a la funcionalidad que habilita las aplicaciones para llevarse a cabo de manera eficiente y segura. Posiblemente incluya un amplio rango de funcionalidades de Capa4-Capa7 del modelo de referencia OSI, y capacidades de seguridad tales como cortafuegos.

Conmutadores SDN puros

En los conmutadores puramente SDN, todas las funciones de control de los conmutadores tradicionales (ej. protocolos de enrutamiento que se usan para crear bases de información de reenvío) están corriendo en el controlador central. La funcionalidad de estos conmutadores está restringida totalmente al plano de datos (en próximos epígrafes este concepto será ampliado).

Conmutadores híbridos

En los conmutadores hibrido, la tecnología SDN y los protocolos tradicionales de conmutación corren de manera simultánea. Un administrador de red configura al controlador SDN para descubrir y controlar cierto flujo de tráfico mientras los protocolos distribuidos tradicionales de red continúan direccionando el resto del tráfico en la red.

Redes híbridas

Una red hibrida es una red donde los conmutadores tradicionales y los conmutadores SDN, sin importar si son puros o híbridos, trabajan en el mismo ambiente de red.

Northbound API

Relativo a la Figura 1.1, las northbound API son las interfaces de aplicaciones programables que habilitan la comunicación entre la capa de control y la capa de aplicaciones.

Southbound API

Relativo a la Figura 1.1, las southbound API son la interfaces de aplicaciones programables que habilitan la comunicación entre la capa de control y la capa de infraestructura, entre los protocolos que permiten esta comunicación se encuentra OpenFlow.

Según la ONF, la capa de infraestructura puede estar constituida por un amplio rango de dispositivos ya sean físicos o virtuales, entre ellos enrutadores y conmutadores [3].

En la Figura 1.1 se muestran cinco dispositivos de red que emplean un mismo sistema operativo instalado en un controlador externo, desde donde todos se gestionan. En la parte superior de la figura, entre las capas de control y aplicación, existen northbound API (Interfaces de Programación de Aplicaciones *o Application Programming Interface*, por sus siglas en inglés) abiertas, también denominadas API “hacia el norte”, que brindan nuevas funcionalidades a esta tecnología, haciendo posible la implementación de servicios de red comunes, incluyendo enrutamiento, multidifusión, seguridad, control de acceso, gestión de velocidad de transmisión, ingeniería de tráfico, QoS, optimización del almacenamiento, uso de energía y políticas de gestión que se diseña para satisfacer los objetivos de negocio.

Los conmutadores delegan su inteligencia al controlador y pasan a ser simples unidades de conmutación de tráfico. Las SDN permiten gestionar toda la red a través de la orquestación inteligente de los sistemas de aprovisionamiento. La ONF está estudiando

southbound API abiertas, también denominada interfaz “hacia el sur”, para promover la gestión de equipamiento de múltiples proveedores, lo que facilita la asignación de recursos, la verdadera virtualización de las redes y seguridad en los servicios de la nube.

En el centro de la arquitectura de las SDN se encuentra la capa Control con el controlador SDN, que es quien gestiona los flujos. El controlador equivale al sistema operativo de la red que controla todas las comunicaciones entre las aplicaciones y los dispositivos. El controlador SDN se encarga de traducir las necesidades o requisitos de la capa Aplicación a los elementos de red, y de proporcionar información relevante a las aplicaciones SDN, pudiendo incluir estadísticas y eventos.

En las SDN la inteligencia de la red se encuentra lógicamente centralizada en controladores, basados en software, que mantienen una visión global de la red. Como resultado, la red aparece frente a las aplicaciones y a las decisiones de política como un conmutador lógico y único. De esta forma, las empresas que utilizan la tecnología SDN logran la independencia de los proveedores y adquieren el control sobre toda la red desde un único punto lógico, lo que simplifica en gran medida el diseño y operación de sus redes.

Las SDN también permiten simplificar los dispositivos de red, pues ya estos no tienen que entender y procesar varios protocolos estándares, sino simplemente aceptar instrucciones de los controladores SDN.

La capa Aplicaciones permite comunicar al controlador SDN, mediante las API, sus necesidades y el comportamiento que desean de la red. La interfaz de los controladores SDN hacia las aplicaciones es un conjunto de interfaces ya que la definición de aplicaciones SDN es muy amplia, cubriendo desde servicios de red, como QoS, hasta aplicaciones de negocio. Las diversas aplicaciones SDN tienen que hablar con la red de maneras diferentes y requerir interfaces diferentes.

Quizás lo más importante de las SDN es que los operadores de red pueden configurar, mediante programación y de forma centralizada, la red y sus servicios; modificando su comportamiento en tiempo real se logran desplegar nuevas aplicaciones y servicios en cuestión de horas o días, en lugar de las semanas o meses necesarios en la tecnología de redes anteriores. Al estar la red centralizada en la capa de control, las SDN ofrecen a los operadores de red flexibilidad para configurar, administrar, proteger y optimizar los recursos de la red a través de programas dinámicos y automatizados. De esta forma, los

administradores no tienen que esperar a que los fabricantes lancen sus programas, lo cual es muestra de la relativa independencia de esta tecnología del desarrollo de software por parte de las empresas proveedoras. Entre las principales innovaciones de las SDN se tienen:

- Separación de los planos de control y datos.
- La centralización del plano de control.
- Programación del plano de control.
- Control basado en flujo.
- Normalización de las Interfaces de Programación de Aplicaciones (API).

1.3.1 Separación del plano de control y plano de datos

Los estándares de red a menudo se organizan en tres planos: datos, control y gestión. El plano de datos consta de todos los mensajes que son generados por los usuarios. Para el transporte de estos mensajes, la red tiene que realizar algún trabajo interno, como encontrar la ruta más corta con protocolos de enrutamiento como OSPF (*Open Shortest Path First,* por sus siglas en inglés) o protocolos de reenvío como STP (Protocolo de Árbol Extendido o *Spanning Tree Protocol*, por sus siglas en inglés). Los mensajes que se utilizan para este propósito se denominan mensajes de control y son esenciales para el funcionamiento de la red, permitiendo a los administradores de la red llevar un control de las estadísticas de tráfico y el estado de los distintos equipos de la red. La gestión, aunque importante, es diferente del control ya que puede ser opcional y con frecuencia no se realiza en redes pequeñas como las redes domésticas [3].

Una de las innovaciones clave de las SDN es que el plano de control se desacopla del plano de datos, en el que se procesan los paquetes usando las tablas de flujos implementadas por el plano de control. La lógica de control se separa y se implementa en un controlador que crea las tablas de flujos. Los conmutadores transportan los datos a partir de las tablas que le envía el controlador, simplificando su implementación y reduciendo la complejidad y el costo de los mismos.

1.3.2 Centralización del plano de control

En las arquitecturas tradicionales los planos de datos y de control se encontraban distribuidos en el equipamiento. Por ejemplo, cada enrutador contribuye a preparar las tablas de enrutamiento e intercambian información de disponibilidad con sus vecinos y

estos con sus otros vecinos, y así sucesivamente. Este paradigma de control distribuido fue uno de los pilares del diseño de Internet, incuestionable hasta hace pocos años.

La centralización de los servicios, que fue considerado una deficiencia, ahora es considerada una ventaja. En la vida real, se tiene que la mayoría de las organizaciones y los equipos trabajan con control centralizado. Por ejemplo, si un empleado cae enfermo, él simplemente llama al jefe, y el jefe hace los preparativos para la continuación del trabajo. En una organización totalmente distribuida, Juan el empleado enfermo, tendría que llamar a todos sus compañeros y decirle que está enfermo. Ellos le dirán a otros compañeros el estado de Juan. Demorará un poco hasta que todos los empleados sepan sobre la enfermedad de Juan, luego entre todos buscarán, si existe, una solución para realizar el trabajo de Juan hasta que se recupere. Esto es bastante ineficiente, pero es la forma actual de los protocolos de control de Internet. La centralización del control permite un ajuste dinámico en función de los cambios de estado de la red, mucho más rápido que con los protocolos distribuidos.

Es claro que la centralización también tiene algunos problemas, entre los que se encuentra la escalabilidad por lo que necesitan métodos distribuidos. Por ello, se divide la red en subconjuntos o zonas que son lo suficientemente pequeños para tener una estrategia de control común. Una clara ventaja del control centralizado es que los cambios de estado o cambios de política se propagan más rápido que en un sistema distribuido. Frente a fallos de los controladores centralizados, controladores de respaldo se pueden utilizar para tomar el control si presenta problemas el controlador principal. El plano de datos sigue siendo totalmente distribuido. En la arquitectura de las SDN debe existir un equilibrio entre la complejidad de la gestión y el control centralizado pues al aumentar la cantidad de controladores es más difícil la gestión de la red ya que se necesita configurar y monitorizar cada uno de ellos.

1.3.3 Plano de control programable

El plano de control al estar ubicado en un controlador central, facilita que se puedan implementar políticas de control, simplemente cambiando el software. Utilizando *northbound API*, se pueden implementar una serie de políticas y modificarlas de forma dinámica de acuerdo al estado de la red. Este plano de control programable es el aspecto más importante de las SDN, pues permite que la red se divida en varias redes virtuales que poseen políticas muy diferentes y residan en una misma infraestructura de hardware. Los cambios dinámicos en las políticas de control serían muy difíciles y lentos con un

plano de control totalmente distribuido. En la Figura 1.2 se muestra la arquitectura de las SDN con el controlador interactuando con las API.

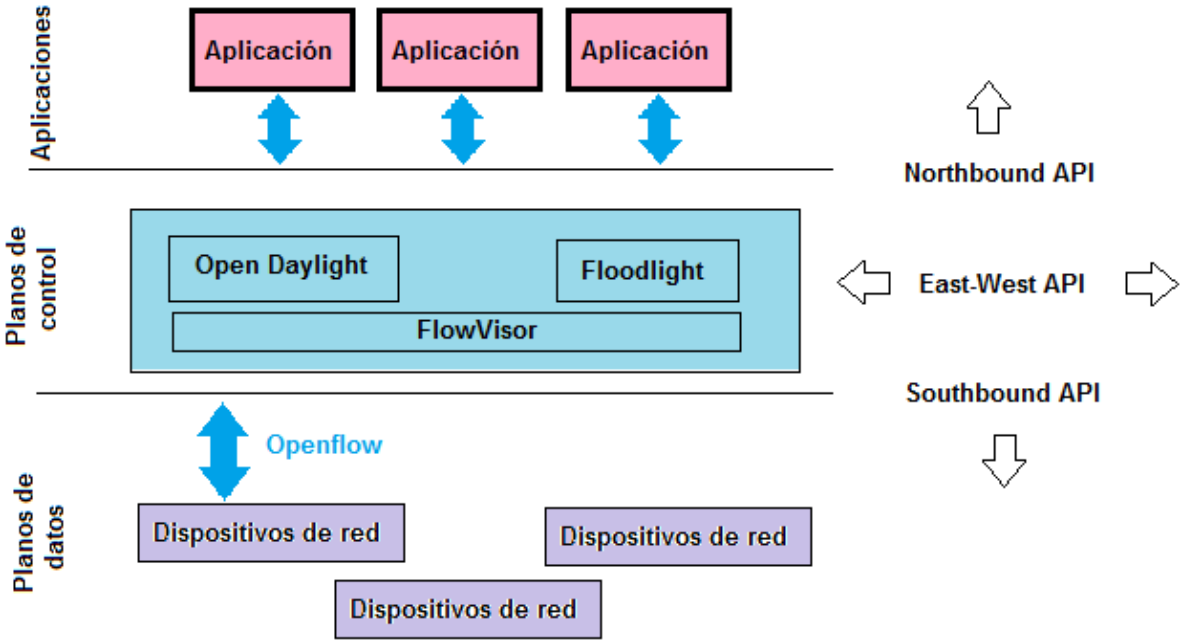

Figura 1.2 Arquitectura SDN y sus API

1.3.4 Control basado en flujos

En los últimos 30 años (desde la normalización del primer estándar de Ethernet), el tamaño de los archivos ha crecido de acuerdo a la ley de Moore. Sin embargo, el tamaño de los paquetes, se ha mantenido igual (aproximadamente 1518 byte en tramas Ethernet). Debido a esto, gran parte del tráfico hoy consta de una secuencia de paquetes en lugar de uno solo. Por ejemplo, un archivo de gran tamaño puede requerir la transmisión de cientos de paquetes. El transporte de media generalmente consta de un flujo de paquetes que se intercambian durante un largo período de tiempo. En tales casos, la información de control se solicita por el conmutador cuando el primer paquete de un flujo se recibe y se utiliza para todos los paquetes del mismo flujo. Este flujo puede ser identificado por máscaras en las cabeceras de los paquetes y por el puerto de entrada en que se recibió el mismo. El control basado en flujos reduce significativamente el tráfico entre el controlador y los dispositivos de reenvío.

1.4 Beneficios de las SDN

De manera resumida se exponen a continuación los beneficios de esta arquitectura [4]:

- Visión unificada de la estructura de la red: con las SDN se tiene una visión unificada simplificando la gestión y el aprovisionamiento de los recursos de la red.
- Alta utilización: una ingeniería de tráfico centralizada proporciona una visión global de la red, tanto en la oferta como en la demanda de recursos en la red. Al gestionar a través de esta visión global los caminos de extremo a extremo logran un alto aprovechamiento de los enlaces.
- Manejo más rápido de los fallos: los fallos, en un enlace o nodo, se gestionan más rápido. Además, los sistemas convergen más rápido hacia el objetivo óptimo y su comportamiento es predecible.
- Entorno de prueba de alta fidelidad: la red dorsal se emula completamente mediante software, lo que ayuda en la comprobación, verificación y en la gestión de posibles escenarios que se puedan dar.
- Actualizaciones sin impactos: el desacople del plano de control respecto al plano de datos, permite llevar a cabo actualizaciones de software sin pérdida de paquetes o degradación de la capacidad posibilitando la configuración de los dispositivos en tiempo real.
- Rápida innovación: al poder controlar los dispositivos y los servicios a través del controlador, sin necesidad de configurar el equipamiento específico de la red, las SDN permiten la innovación en nuevos protocolos, limitado sólo por el intelecto de los programadores de red.
- Control más granular de la red: las SDN posibilitan la aplicación de una amplia variedad de políticas a diferentes niveles, como sesiones, usuarios, dispositivos y aplicaciones.
- Mejor experiencia del usuario: las aplicaciones aprovechan los beneficios de control y de gestión centralizados haciendo que la red se adapte a las necesidades del usuario de forma transparente.

Al finalizar con la caracterización de la tecnología SDN, el próximo capítulo se centra en uno de los dispositivos de red más empleados en la implementación de este tipo de redes, los conmutadores SDN OpenFlow, se analizan los puntos de convergencia y las diferencias entre estos dispositivos y los conmutadores tradicionales.

CAPITULO 2 . COMPARACIÓN ENTRE CONMUTADORES TRADICIONALES Y CONMUTADORES SDN OPENFLOW.

2.1 Introducción

A principio de la década de los ‘90, se disponían de tres tipos de dispositivos de interconexión LAN: los concentradores, los puentes y los enrutadores. Posteriormente otro dispositivo de interconexión se popularizó por las ventajas que ofrecía en cuanto a prestaciones: el conmutador Ethernet.

Los conmutadores Ethernet aparecieron en 1991 cuando la industria Kalpana lanzó al mercado la EtherSwitch. Kalpana fue adquirida por Cisco en 1994. Desde la perspectiva Ethernet, los conmutadores son puentes multipuertos de alto desempeño. Los conmutadores son una parte fundamental de la mayoría de las redes. Estos hacen posible que varios usuarios envíen información a través de una red al mismo tiempo sin disminuir la velocidad de transmisión entre sí [5].

Existen diferentes tipos de conmutadores y redes, los conmutadores que proporcionan una conexión separada para cada nodo en la red interna de una compañía se denominan conmutadores LAN. Esencialmente, un conmutador LAN crea una serie de redes instantáneas que contiene sólo los dos dispositivos que se comunican entre sí en ese momento en particular. Este trabajo se centra en los conmutadores LAN utilizados en las redes Ethernet.

Un conmutador tiene el potencial de cambiar radicalmente la forma en que los nodos se comunican entre sí. Pero surge la pregunta: en qué se diferencia un conmutador tradicional de un conmutador SDN OpenFlow. Este capítulo se centra en algunas de sus diferencias y similitudes al realizar la caracterización de cada uno por separado y al final del capítulo en una tabla resumir los puntos de comparación. La caracterización del conmutador SDN OpenFlow será más amplia ya que constituye el objetivo principal de este trabajo.

2.2 Conmutadores Ethernet tradicionales

Los conmutadores Ethernet son dispositivos lógicos de interconexión de equipos que operan en la capa de enlace de datos del modelo OSI, aunque existen conmutadores más avanzados que funcionan en la capa de red (L3). Como los puentes, los conmutadores reenvían y filtran tramas usando direcciones MAC de destino, y automáticamente construyen tablas de reenvío usando las direcciones MAC de origen de las tramas que atraviesan los dispositivos. La mayor diferencia entre puentes y conmutadores es que los puentes usualmente tienen 2 puertos mientras los conmutadores pueden tener hasta varias docenas de ellos. Un conmutador es un dispositivo de propósito especial diseñado para resolver problemas de rendimiento en la red, ocasionados por anchos de banda pequeños y embotellamientos. El conmutador puede agregar mayor ancho de banda, acelerar la salida de paquetes, reducir tiempo de espera y bajar el costo por puerto.

El conmutador al segmentar económicamente la red dentro de pequeños dominios de colisiones, obtiene un alto porcentaje de ancho de banda para cada estación final; además reduce o casi elimina que cada estación compita por el medio, dando a cada una de ellas un ancho de banda comparativamente mayor.

Los conmutadores poseen la capacidad de aprender y almacenar las direcciones de capa 2 (direcciones MAC) de los dispositivos alcanzables a través de cada uno de sus puertos. En una LAN Ethernet, una trama contiene un paquete normal como carga útil (*payload*) de la trama con un encabezado especial que incluye la información de la dirección MAC relativa al origen y al destino del paquete.

Los conmutadores basados en paquetes utilizan uno de los siguientes métodos para enrutar el tráfico:

Almacenamiento y reenvío (*Store and Forward*)

Los conmutadores de almacenamiento y reenvío guardan cada trama en un *buffer* antes del intercambio de información hacia el puerto de salida. Mientras la trama está en el *buffer*, el conmutador calcula el CRC y mide el tamaño de la misma. Si el CRC falla, o el tamaño es muy pequeño o muy grande (un cuadro Ethernet tiene entre 64 bytes y 1518 bytes) la trama se descarta. Si todo se encuentra en orden la trama se encamina hacia el puerto de salida.

Este método asegura operaciones sin error y aumenta la confianza de la red. Pero el tiempo que se utiliza para guardar y chequear cada trama añade un tiempo de demora

importante al procesamiento de las mismas. La demora o *delay* total es proporcional al tamaño de las tramas: cuanto mayor es la trama, más tiempo toma este proceso.

Método de corte (*Cut-through*)

Los conmutadores de método de corte se diseñaron para reducir la latencia. Esos conmutadores minimizan el *delay* leyendo solo los 6 primeros bytes de datos de la trama, que contiene la dirección de destino MAC, e inmediatamente la encaminan.

El problema de este tipo de conmutador es que no detecta tramas corruptas causadas por colisiones (conocidos como *runts*), ni errores de CRC. Cuanto mayor sea el número de colisiones en la red, mayor será el ancho de banda que consume al encaminar tramas corruptas.

Existe un segundo tipo de conmutador de método de corte denominados libres de fragmentos, que se proyectaron para eliminar este problema. El conmutador siempre lee los primeros 64 bytes de cada trama, la razón para ello es que la mayoría de los errores y colisiones tienen lugar durante los primeros 64 bytes iniciales de un paquete, de esta manera se asegura que tenga por lo menos el tamaño mínimo, y se evita el encaminamiento de *runts* por la red.

Método de corte adaptativo *(Adaptative Cut-Through)*

Estos conmutadores procesan tramas en el modo adaptativo, son compatibles con almacenamiento y reenvío y con el método de corte. Cualquiera de los modos se puede activar por el administrador de la red, o el conmutador puede ser lo bastante inteligente como para escoger entre los dos métodos, basado en el número de tramas con error que pasan por los puertos.

Cuando el número de tramas corruptas alcanza un cierto nivel, el conmutador puede cambiar del modo de método de corte al de almacenamiento y reenvío, volviendo al modo anterior cuando la red se normalice.

Los conmutadores LAN varían según su diseño físico. Actualmente entre las configuraciones en uso se encuentran:

- **Memoria compartida:** almacena todos los paquetes entrantes en un *buffer* de memoria común compartido por todos los puertos del conmutador (conexiones de entrada y salida), luego los envía al puerto correcto para el nodo de destino.

- ***Matrix*:** este tipo de conmutador tiene una grilla interna con los puertos de entrada y de salida cruzándose entre sí. Cuando se detecta un paquete en un puerto de entrada, la dirección MAC se compara con la tabla de búsqueda para encontrar el puerto de salida apropiado. El conmutador efectúa entonces una conexión en la grilla en el punto donde se intersecan estos dos puertos.
- **Arquitectura de bus:** en lugar de una grilla, una ruta de transmisión interna (bus común) se comparte por todos los puertos que utilicen TDMA. Un conmutador basado en esta configuración tiene un *buffer* de memoria dedicado para cada puerto y un ASIC para controlar el acceso interno al bus [6].

2.3 *Bridging* transparente

El *bridging* transparente adquiere ese nombre porque es un proceso que se ejecuta de manera invisible a los usuarios de la red a los cual se le presta servicios. Este proceso añade conocimientos e información de la red al inspeccionar las direcciones de origen de todas las tramas que llegan a las interfaces del conmutador [7].

La mayoría de los conmutadores LAN Ethernet utilizan este método para crear sus tablas de búsqueda de direcciones. El *bridging* transparente es una tecnología que permite a un conmutador aprender todo lo que necesita acerca de la ubicación de los nodos en la red sin que el administrador de red tenga que ejecutar ninguna acción. Este método tiene cinco partes:

- Aprendizaje
- Inundación
- Filtrado
- Envío
- Envejecimiento

A continuación se describe el funcionamiento de este conmutador:

- El conmutador se agrega a la red y los diversos segmentos se conectan a los puertos del conmutador.
- Una computadora (Nodo A) del primer segmento (Segmento A) envía datos a una computadora (Nodo B) que se encuentra en otro segmento (Segmento C).
- La tabla MAC debe completarse con las direcciones MAC y sus puertos correspondientes. El proceso de aprendizaje permite que estos mapeos se adquieran dinámicamente durante el funcionamiento normal. A medida que cada

trama ingresa al conmutador, el conmutador analiza la dirección MAC de origen. Mediante un proceso de búsqueda, el conmutador determina si la tabla ya contiene una entrada para esa dirección MAC. Si no existe ninguna entrada, el conmutador crea una nueva entrada en la tabla MAC utilizando la dirección MAC de origen y asocia la dirección con el puerto en el que llegó la entrada. Ahora, el conmutador puede utilizar esta asignación para reenviar tramas a este nodo. Este proceso se conoce como ***aprendizaje.*** En el ejemplo mencionado el conmutador obtiene el primer paquete de datos del Nodo A. Lee la dirección MAC y la guarda en la tabla de búsqueda para el Segmento A. El conmutador ahora sabe encontrar al Nodo A cada vez que un paquete esté dirigido al mismo.

- El conmutador no sabe llegar al Nodo B y envía el paquete a todos los segmentos excepto a aquel por el que ha llegado (Segmento A). Cuando un conmutador envía un paquete hacia todos los segmentos para encontrar un nodo específico, esto se denomina ***inundación.***
- El Nodo B obtiene el paquete y envía un paquete de regreso al Nodo A como confirmación.
- El paquete proveniente del Nodo B llega al conmutador. Ahora el conmutador puede agregar la dirección MAC del Nodo B a la tabla de búsqueda para el segmento C. El conmutador ya conoce la dirección del Nodo A y envía el paquete directamente hacia el mismo. El Nodo A esta en un segmento diferente que el Nodo B, el conmutador debe de conectar los dos segmentos para enviar el paquete. El ***reenvío selectivo*** es el proceso por el cual se analiza la dirección MAC de destino de una trama y se le reenvía al puerto correspondiente. Ésta es la función principal del conmutador. Cuando una trama de un nodo llega al conmutador y el conmutador ya aprendió su dirección MAC, dicha dirección se hace coincidir con una entrada de la tabla MAC y la trama se reenvía al puerto correspondiente, en lugar de saturar la trama hacia todos los puertos.
- El siguiente paquete desde el Nodo A hacia el Nodo B llega al conmutador. El conmutador ahora tiene la dirección del Nodo B también, de modo tal que envía el paquete directamente al Nodo B.
- El Nodo C envía información al conmutador para el Nodo A. El conmutador verifica la dirección MAC del Nodo C y la agrega a la tabla de búsqueda del Segmento A. El conmutador ya tiene la dirección del Nodo A y determina que ambos nodos se encuentran en el mismo segmento, por lo que no se necesita conectar el

Segmento A a otro segmento para que los datos viajen desde el Nodo C hasta el Nodo A, entonces el conmutador ignora los paquetes entre nodos ubicados en el mismo segmento. Existen otros casos en los que las tramas no se reenvían, uno de los usos del filtrado ya se describió: un conmutador no reenvía una trama al mismo puerto en el que llega. El conmutador también descarta una trama corrupta y otra razón por la que una trama se filtra es por motivos de seguridad. Un conmutador tiene configuraciones de seguridad para bloquear tramas hacia o desde direcciones MAC selectivas o puertos específicos. Esto se conoce como ***filtrado.***

- El aprendizaje y la inundación continúan a medida que el conmutador agrega nodos a las tablas de búsqueda. La mayoría de los conmutadores tienen la suficiente cantidad de memoria para mantener las tablas de búsquedas, pero borran la información más antigua de modo tal que el conmutador no pierda tiempo buscando entre direcciones antiguas. Para optimizar el uso de esta memoria, los conmutadores utilizan una técnica denominada ***envejecimiento***. Básicamente cuando se agrega una entrada relativa a un nodo en una tabla de búsqueda, se le adjudica una etiqueta temporal. Cada vez que se recibe un paquete desde un nodo, la etiqueta temporal se actualiza. El conmutador tiene un temporizador configurable por el usuario que borra la entrada después de que haya transcurrido una determinada longitud de tiempo sin actividad proveniente de ese nodo. Esto libera valiosos recursos de memoria para otras entradas.

El *bridging* transparente es una manera muy buena y esencialmente libre de mantenimiento para agregar toda la información que un conmutador necesita para hacer su trabajo.

En el ejemplo que se describe anteriormente dos nodos comparten cada segmento. En una red LAN conmutada ideal, cada nodo tendría su propio segmento, esto elimina la posibilidad de que existan colisiones [6].

2.4 Conmutador de capa 3

Mientras la mayoría de los conmutadores operan en la Capa de Datos (L2) del Modelo de Referencia OSI, algunos incorporan las características de un enrutador y operan también en la Capa de Red (L3). De hecho, un conmutador L3 es increíblemente similar a un enrutador, son los conmutadores que, además de las funciones tradicionales de la capa 2,

incorporan algunas funciones de enrutamiento o *routing*, por ejemplo la determinación del camino basado en informaciones de capa de red y soporte a los protocolos de enrutamiento tradicionales (ejemplo RIP, OSPF).

Cuando un enrutador recibe un paquete, observa las direcciones de origen y destino de la Capa de Red para determinar la ruta que el paquete debe tomar. Esto se considera actividad de *networking* de capa 3 (de Red). Un conmutador estándar se basa en las direcciones MAC para determinar el origen y el destino de un paquete, lo cual es *networking* de capa 2 (de Datos). La diferencia fundamental entre un enrutador y un conmutador de capa 3 es que los conmutadores de capa 3 tienen un hardware optimizado para procesar los datos tan rápido como los conmutadores de capa 2, y aun así toman decisiones acerca de cómo transmitir el tráfico de capa 3, al igual que lo haría un enrutador. Dentro del entorno LAN, un conmutador de capa 3 usualmente es más rápido que un enrutador porque está construido sobre un hardware de conmutación. De hecho los conmutadores de capa 3, entre ellos los de Cisco, son realmente enrutadores que operan más rápido porque están construidos en base a hardware de conmutación con chips personalizados dentro de la caja.

2.5 Caracterización del conmutador SDN OpenFlow

La idea básica para crear un conmutador SDN OpenFlow es simple: se explota el hecho de que los conmutadores y enrutadores Ethernet más modernos cuentan con tablas de flujos, típicamente construidas con TCAM (*Ternary Content-Addressable Memory*, por sus siglas en inglés) que se utilizan en la implementación de cortafuegos, NAT (Traducción de Dirección de Red o *Network Address Translation*, por sus siglas en inglés), QoS y para recolectar estadísticas. Mientras que cada tabla de flujo de los vendedores es diferente, se identifican un grupo interesante de funciones en común que corren en numerosos conmutadores y enrutadores. OpenFlow explota este grupo común de funciones para acceder al manejo y control de numerosos dispositivos de numerosos fabricantes de igual forma.

Un conmutador OpenFlow consiste de al menos tres partes: (1) una tabla de flujo, con las acciones asociadas a cada entrada de flujo, para notificarle al conmutador como debe ser procesado el flujo, (2) un canal seguro que conecta al conmutador con el proceso controlador remoto (llamado controlador), los comandos autorizados y los paquetes se envían entre estos dispositivos a través del (3) protocolo OpenFlow, que provee una

forma abierta y estándar para que un controlador se pueda comunicar con un conmutador. El protocolo OpenFlow a través de interfaces estandarizadas define externamente las entradas en las tablas de flujos, así se evita la necesidad de administradores y/o investigadores para programar al conmutador [8].

Es útil categorizar los conmutadores dentro de conmutadores OpenFlow dedicados, que no soportan el procesado de capa 2 y capa 3 normal, y conmutadores y enrutadores Ethernet de propósito general y comercial en los cuales se habilita OpenFlow, en donde el protocolo OpenFlow y las interfaces se añaden como nuevas características.

Conmutadores OpenFlow dedicados: un conmutador OpenFlow dedicado es un elemento de hardware que reenvía paquetes entre puertos, gestionado por un proceso de control remoto. En estos dispositivos los flujos están ampliamente definidos, y limitados solo por las capacidades en particular de la implementación de las tablas de flujo. Por ejemplo, un flujo puede ser una conexión TCP, todos los paquetes provenientes de una dirección MAC en específico o desde una dirección IP específica, todos los paquetes con la misma etiqueta de una VLAN, o todos los paquetes que provienen del mismo puerto de un conmutador. Para experimentos que involucran a paquetes que no son IPv4, un flujo puede ser definido como todos aquellos paquetes que coincidan con una cabecera (aunque no sea estándar) [8].

El camino de datos en un conmutador OpenFlow consiste en una tabla de flujo, y una acción asociada con cada entrada de flujo. El grupo de acciones que soporta un conmutador OpenFlow es extensible, pero a continuación se describen los requerimientos mínimos que deben cumplir todos los conmutadores. Para un gran desempeño y un bajo costo del camino de datos debe existir un grado de flexibilidad cuidadosamente predefinido, esto supone renunciar a la posibilidad de tener un manejo arbitrario para cada paquete. Entonces, se definen un grupo de acciones básicas requeridas para todos los conmutadores OpenFlow [8].

- Reenviar los paquetes de flujos hacia un puerto dado (o puertos dados). Esto permite a los paquetes ser enrutados a través de la red.
- Encapsular y reenviar los paquetes del flujo hacia un controlador. Los paquetes se entregan al canal seguro, se encapsulan y se envían al controlador. Típicamente se usa para el primer paquete de un flujo nuevo, entonces el controlador decide si

el flujo es añadido o no a la tabla de flujo. En algunos experimentos se utiliza para reenviar todos los paquetes hacia un controlador para su procesamiento.

- Desechar los paquetes del flujo. Se usa en seguridad, para restringir los ataques a servicios, o reducir la difusión espuria del tráfico de descubrimiento proveniente de terminales finales.

Una entrada de la Tabla de Flujo tiene tres campos: (1) una cabecera para definir el flujo, (2) la acción, que define como el paquete es procesado, y (3) estadísticas, que mantiene el número de paquetes y *bytes* para cada flujo, y el tiempo desde que el último paquete coincidió con el flujo (para ayudar a remover aquellos flujos inactivos). Ver Figura 2.1.

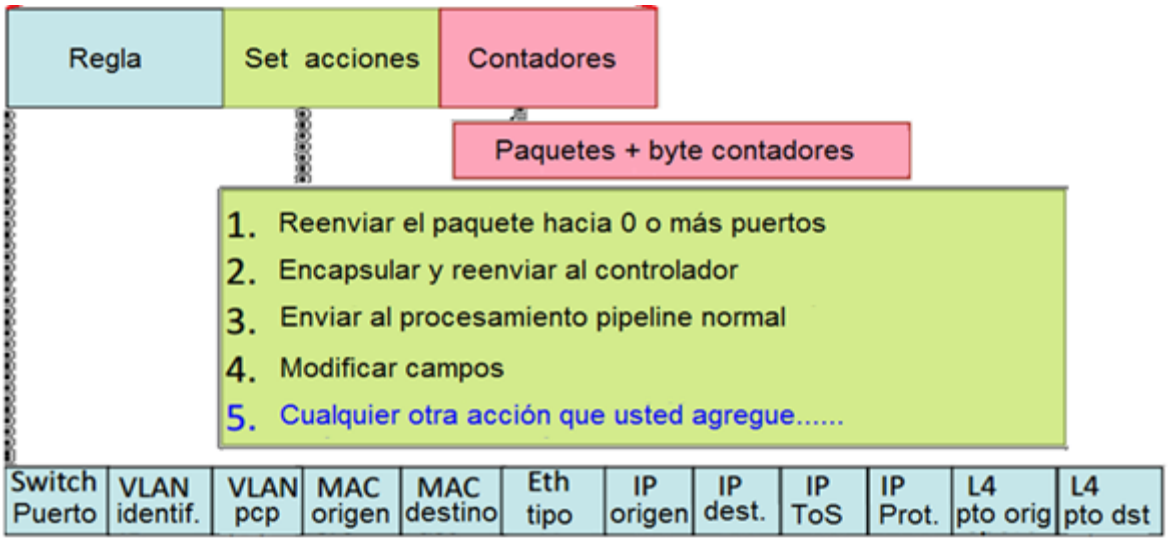

Figura 2.1 Entrada de flujos y campos que definen a una entrada de flujo.

Conmutadores con OpenFlow habilitado: algunos conmutadores comerciales, enrutadores y puntos de acceso se habilitan como dispositivos OpenFlow. El OpenFlow se habilita añadiendo la Tabla de Flujo, el Canal Seguro y el protocolo OpenFlow, se agregan de manera que corran en un sistema operativo de un conmutador. En una red de conmutadores y puntos de acceso comerciales con OpenFlow habilitado, todas las Tablas de Flujo se manejan por el mismo controlador.

En este tipo de dispositivos el objetivo es habilitar experimentos que tengan lugar en una red de producción sin interferir con el tráfico regular y las aplicaciones. Entonces, para ganar la confianza del administrador de red, los conmutadores con OpenFlow habilitado aíslan el tráfico experimental (que se procesa por las tablas de flujo del conmutador OpenFlow) del tráfico de producción que se procesa por las colas de procesos normales

de capa 2 y de capa 3 del conmutador. Existen dos formas de diferenciar los paquetes de tráfico experimental de los de tráfico de producción, al añadir una cuarta acción:

- Reenviar los paquetes de estos flujos a través del procesamiento normal de las colas de proceso del conmutador.

O al definir un grupo separado de VLAN para tráfico experimental y para tráfico de producción. Ambos acercamientos permiten que el tráfico de producción normal no forme parte de los experimentos para que se procese de manera usual a como lo haría el conmutador. Todos los conmutadores con OpenFlow habilitado están obligados a soportar una de las 2 formas; algunos inclusive soportan las 2.

2.6 Componentes del conmutador OpenFlow

Un conmutador OpenFlow consiste en una o más tablas de flujos y en un grupo de tablas, los cuales realizan búsquedas de paquetes y reenvíos, en un canal de comunicación seguro hacia un controlador externo y en un protocolo que permita la comunicación entre controlador y dispositivo, en este caso el protocolo OpenFlow. El controlador maneja al conmutador a través del protocolo OpenFlow. Al usar este protocolo, el controlador añade, actualiza, y elimina entradas de flujos, de dos formas, reactivamente (en respuesta a paquetes) o proactivamente (se instalan antes de la llegada de cualquier paquete). Cada tabla de flujo en el conmutador contiene un grupo de entradas de flujos, cada entrada de flujo contiene campos de coincidencias, contadores, y un grupo de instrucciones para aplicar a los paquetes coincidentes. Las instrucciones asociadas con cada entrada de flujo describen el reenvío de los paquetes, la modificación de los paquetes, el grupo de tablas del procesamiento y el procesamiento por colas.

2.7 Procesamiento *Pipeline*

Los conmutadores que obedecen a OpenFlow vienen de dos formas: OpenFlow-puros, y OpenFlow-híbridos. Los conmutadores OpenFlow puros soportan únicamente operaciones OpenFlow, en estos conmutadores todos los paquetes se procesan por el *pipeline* de OpenFlow, y no pueden ser procesados de otra manera.

Los conmutadores OpenFlow híbridos soportan ambas operaciones, las de OpenFlow y las de conmutación normal de Ethernet (conmutación tradicional Ethernet capa 2, separación de redes a través de VLAN, enrutamiento capa 3, y procesado según la calidad de servicio). Estos conmutadores pueden proveer un mecanismo de clasificación

a parte de OpenFlow que enrute el tráfico hacia el *pipeline* OpenFlow o hacia el *pipeline* normal. Por ejemplo, un conmutador puede usar una bandera de VLAN o el puerto de entrada del paquete para decidir a cuál de los dos procesos *pipeline* enviar el paquete, o puede direccionar todos los paquetes hacia el *pipeline* OpenFlow. Este mecanismo de clasificación se encuentra fuera de los objetivos de estudio de esta tesis. Un conmutador OpenFlow hibrido puede también permitir que un paquete pase del proceso *pipeline* OpenFlow hacia el proceso *pipeline* normal a través de puertos virtuales NORMALES y de FLOOD [9].

El proceso *pipeline* OpenFlow de cada conmutador OpenFlow contiene múltiples tablas de flujos, cada tabla de flujo contiene múltiples entradas de flujos. Este proceso define cómo los paquetes interactúan con esas tablas de flujos. Un conmutador OpenFlow con una única tabla de flujo es válido, en este caso el procesamiento se simplifica grandemente.

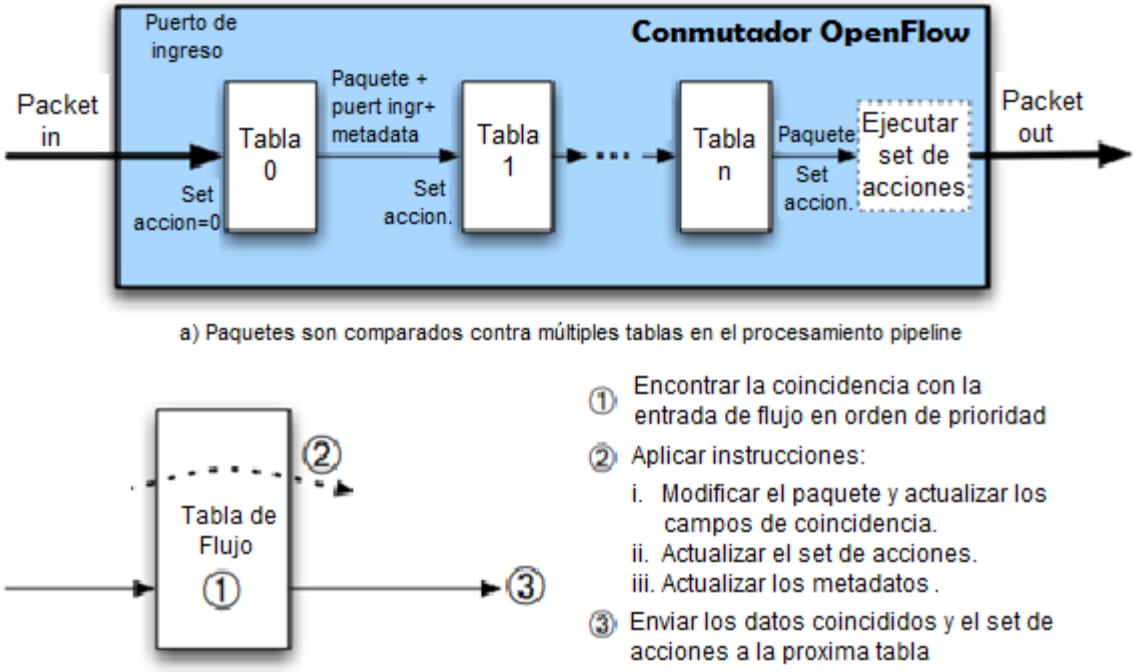

Figura 2.2 Esquema de un conmutador SDN OpenFlow. (Fuente:[7])

Como se aprecia en la Figura 2.2 las tablas de flujo en un conmutador OpenFlow se enumeran secuencialmente, al empezar desde 0. El proceso *pipeline* siempre comienza en la primera tabla de flujo: el paquete se hace coincidir, en caso de ser posible, con la primera entrada de flujo de la tabla de flujo número 0. Otras tablas de flujos quizás se usen en dependencia de la acción asociada a la regla que coincidió en la primera tabla de

flujo. Si el paquete concuerda con una entrada de flujo en una tabla de flujo, la instrucción correspondiente se ejecuta. La instrucción en la entrada de flujo puede explícitamente direccionar el paquete hacia otra tabla de flujo (al utilizar la instrucción *Go-to*), donde el mismo proceso se repite nuevamente. El conmutador comienza por realizar una búsqueda en la primera tabla de flujo, y basado en el procesamiento *pipeline*, puede realizar búsqueda de entradas en otras tablas de flujo. Los campos de coincidencia destinados para búsquedas en las tablas dependen del tipo del paquete. En la Figura 2.3 se muestra el esquema lógico del proceso *pipeline* para ayudar a la comprensión del mismo.

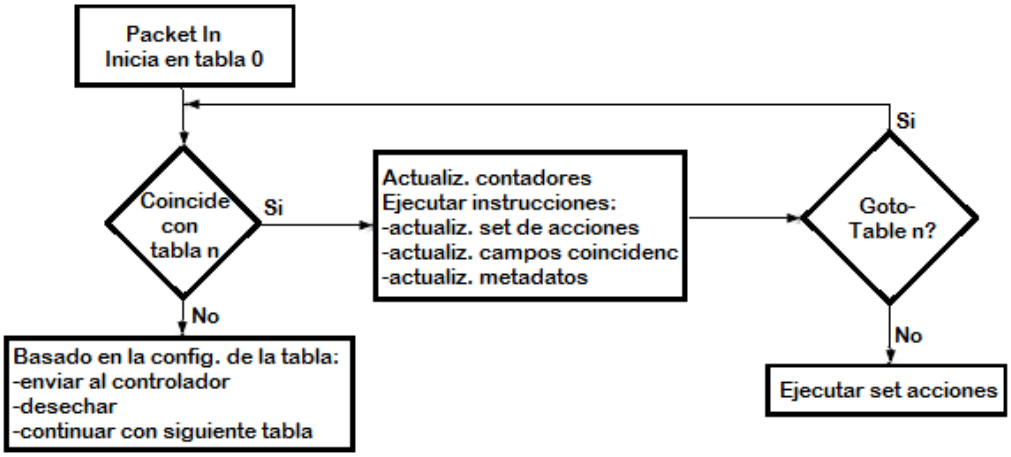

Figura 2.3 Esquema lógico del proceso pipeline del paquete al atravesar el conmutador OpenFlow.

Un paquete corresponde a una entrada de la tabla de flujo si los valores de los campos de la cabecera que se usan para la búsqueda coinciden con los valores que se establecieron en la tabla de flujo. Si un campo de la tabla de flujo tiene un valor de ANY, corresponde a todos los valores posibles en el encabezado.

Para manejar los diversos tipos de entramado de Ethernet, se analiza el contenido de la trama del paquete. En general, el tipo de Ethernet coincidente con OpenFlow es la única descripción considerada por OpenFlow como la carga útil del paquete. Si el paquete tiene etiquetas VLAN, el tipo de Ethernet coincidente con OpenFlow es el que se encuentra tras todas las etiquetas VLAN. Una excepción a la regla son los paquetes con etiquetas MPLS donde OpenFlow no puede determinar el tipo de Ethernet de la carga útil del paquete MPLS [9].

Una entrada de flujo puede direccionar un paquete solamente hacia una tabla de flujo con un número mayor al número de la tabla de flujo a la cual pertenece la entrada de flujo que direccionó al paquete, en otras palabras el procesamiento *pipeline* solamente puede

enviar hacia adelante y no hacia atrás. Obviamente, las entradas de flujos de la última tabla del proceso *pipeline* no pueden incluir la instrucción *Go-to*. Si la entrada de flujo que concuerda no direcciona el paquete hacia otra tabla de flujo, el procesamiento *pipeline* concluye. Cuando el proceso *pipeline* concluye, el paquete se procesa con la acción asociada y usualmente se reenvía.

Si el paquete no concuerda con ninguna entrada de flujo en las tablas de flujos, esta es una tabla perdida. El comportamiento de una tabla perdida depende de la configuración de la tabla, lo normal es enviar el paquete al controlador a través del canal de control vía mensaje *packet-in*, otra opción es desechar el paquete. Una tabla puede también especificar que en caso de una tabla perdida el procesamiento del paquete debe continuar, en este caso el paquete se procesa por la siguiente tabla enumerada de manera secuencial.

2.8 Campos de coincidencia

La Figura 2.4 muestra los campos de coincidencia de los paquetes entrantes contra los que se realiza la comparación. Cada entrada contiene un valor específico o ANY, que concuerda con cualquier valor. Los campos en el estándar OpenFlow se listan en la Figura 2.4. En adición a la cabecera de los paquetes, el proceso de coincidencia puede realizarse teniendo en cuenta el puerto de ingreso y el campo de metadato, que se usa para pasar información entre tablas en un conmutador.

Figura 2.4 Campos de paquetes usados para comparar con las entradas de flujos en busca de coincidencias.

2.9 Instrucciones

Cada entrada de flujo contiene un grupo de instrucciones que se ejecutan cuando un paquete corresponde con dicha entrada. Estas instrucciones dan como resultado cambios para el paquete, ponen en marcha alguna acción y/o procesamiento *pipeline*. Las instrucciones soportadas incluyen:

Apply-Action(s): aplica la(s) acción(es) específica (s) inmediatamente, sin ningún cambio al Grupo de Acciones. Esta instrucción se usa para modificar el paquete entre dos tablas o ejecutar acciones múltiples del mismo tipo. Las acciones se especifican como una lista de acción.

Clear-Actions: despeja todas las acciones del grupo de acciones de manera inmediata.

Write-Action(s): añade la(s) acción(es) específica(s) en el grupo actual de acción. Si una acción del tipo dado existe en el conjunto actual, se sobrescribe, sino se adiciona.

Write-Metadata metadatos/máscara: escribe el valor oculto de metadatos en el campo de metadatos. La máscara especifica cuál de los bits del registro de metadatos se modifica (o sea *new metadata =old metadata & ~mask j value & mask).*

Goto-Table próximo identificador de tabla: indica la siguiente tabla en el proceso *pipeline*. El identificador de la tabla será mayor que el identificador de la tabla actual. Los flujos de la última tabla del *pipeline* no incluyen esta instrucción.

El conjunto de instrucciones asociadas con una entrada de flujo contiene un máximo de una instrucción de cada tipo. Las instrucciones del conjunto se ejecutan en el orden especificado por la lista anterior. En la práctica, las únicas restricciones establecen que la instrucción de *Clear-Actions* se ejecuta antes de la instrucción de *Write-Actions*, y que *Goto-Table* se ejecuta por último.

Un conmutador desecha una entrada de flujo si es incapaz de ejecutar las instrucciones asociadas con la entrada de flujo. En cuyo caso, el conmutador devuelve un error de flujo no soportado. Las tablas de flujo pueden que no soporten cada una de las coincidencias y cada una de las instrucciones [9].

2.10 Acciones

No se requiere que un conmutador soporte todo tipo de acciones, simplemente aquellas marcadas como acciones requeridas como se especifica más adelante. Al conectarse al controlador, un conmutador indica cuál de las acciones optativas soporta.

Acción requerida: salida. La acción de salida reenvía un paquete a un puerto específico. Los conmutadores OpenFlow soportan el reenvío hacia puertos físicos y hacia puertos virtuales definidos en el conmutador. Los puertos estándar se definen como puertos físicos, puertos virtuales en el conmutador, o puerto LOCAL si se soporta. Los

conmutadores OpenFlow soportan el reenvío hacia los siguientes puertos virtuales reservados:

ALL: enviar el paquete hacia todos los puertos estándar, pero no hacia el puerto de admisión o los puertos que se configuran como OFPPC_NO_FWD.

CONTROLADOR: encapsular y enviar el paquete al controlador.

TABLA: enviar el paquete a la primera tabla de flujo con el fin de que este se procese a través del *pipeline* normal de OpenFlow. Solo válido en el conjunto de acciones de un mensaje *packet-out*.

IN PORT: envía el paquete hacia fuera a través del puerto de admisión.

Acción opcional: salida. El conmutador opcionalmente soporta el reenvío para los siguientes puertos virtuales reservados:

LOCAL: envía el paquete a la pila local del sistema de redes del conmutador. El puerto local le permite a las entidades remotas interactuar con el conmutador por la red OpenFlow, en vez de por una red de control separada.

NORMAL: procesar el paquete usando el *pipeline* tradicional no-OpenFlow del conmutador. Si el conmutador no puede remitir los paquetes del *pipeline* OpenFlow al *pipeline* normal, debe indicar que no soporta esta acción.

FLOOD: inundar el paquete a través del *pipeline* normal del conmutador. En general, enviar el paquete fuera a través de todos los puertos estándar, pero no por el puerto de admisión, o los puertos que están dentro del estado de OFPPS_BLOCK. El conmutador también puede usar el ID de VLAN del paquete para seleccionar cuáles puertos inundar.

Los conmutadores OpenFlow puros no soportan acciones de salida en el puerto NORMAL y en el puerto de FLOOD, mientras los conmutadores híbridos-OpenFlow sí las soportan. El reenvío de paquetes al puerto de FLOOD depende de la implementación del conmutador y la configuración.

Acción opcional: *set-queue.* La acción de establecer colas (queue) activa el identificador de cola para un paquete. Cuando el paquete se reenvía a un puerto usando la acción de salida, el identificador de la cola decide que cola se adjuntó a este puerto que se utiliza para reenviar el paquete. El comportamiento de reenvío se define por la configuración de la cola y se usa para proveer soporte básico de Calidad de Servicio (QoS).

Acción requerida: *drop* (desechar). No existe acción explícita para representar la acción de desechar. En lugar de eso, los paquetes cuyo conjunto de acciones no tiene acciones de salida se desechan. Este resultado sucede por conjuntos de instrucciones vacíos o por conjunto de acciones del procesamiento *pipeline* vacíos, o luego de ejecutar una instrucción de *Clear-Actions.*

Acción requerida: *group.* Procesa al paquete a través del grupo que se especifica. La interpretación exacta depende del tipo del grupo.

Acción opcional: *push-tag/pop-tag*. Los conmutadores soportan la habilidad de *push/pop* de etiquetas. Para auxiliar la integración con redes existentes, se sugiere que la habilidad de *push/pop* de etiquetas VLAN se incluya.

Las etiquetas insertadas (*pushed)* deben ser siempre las etiquetas más externas. Cuando una nueva etiqueta VLAN se inserta, debe ser la etiqueta VLAN que aparece inmediatamente después del encabezado de Ethernet. Asimismo, cuando una nueva etiqueta MPLS se inserta debe aparecer como un encabezado después de cualquier etiqueta VLAN.

Acción opcional: *set-field*. Varias acciones de *set-field* modifican los valores de los campos en el encabezado del paquete. Para auxiliar la integración con redes existentes, se sugiere que se incluyan las acciones de modificación de VLAN. Las acciones de *set-field* siempre serán aplicadas al encabezado más externo [9].

2.11 Canal OpenFlow

El canal OpenFlow es la interfaz que conecta cada conmutador OpenFlow con el controlador. A través de esta interfaz, el controlador configura y maneja a los conmutadores, recibe acontecimientos desde el conmutador, y envía paquetes hacia el conmutador.

Entre el canal de datos y el canal OpenFlow, la interfaz se especifica en la implementación, sin embargo todos los mensajes del canal OpenFlow deben formatearse según el protocolo OpenFlow. El canal OpenFlow es usualmente encriptado usando TLS (*Transport Layer Security*, por sus siglas en inglés) pero puede correr directamente sobre TCP. El soporte de múltiples controladores simultáneos está actualmente indefinido.

2.12 Protocolo OpenFlow

En el año 2008 la Universidad de Stanford y la compañía HP inician un proyecto conjunto denominado *Ethane*, que se considera el predecesor directo del protocolo OpenFlow. La idea original era utilizar las redes en producción para correr experimentos de red, aislando el tráfico de los usuarios finales. Lo que no se vislumbró en ese momento es que al hacer esto posible, se abrió un mundo de posibilidades donde la inteligencia de la red se traslada a un controlador y las aplicaciones residen fuera de los conmutadores de red [10].

Como se define en la documentación técnica de la ONF: "OpenFlow es un protocolo de control que permite acceder directamente y manipular el plano de reenvío de dispositivos de red tales como conmutadores y enrutadores, ya sean físicos o virtuales (basados en *hypervisor*)". Este protocolo es un mecanismo que viabiliza la implementación de las SDN. OpenFlow facilita la abstracción de la red, al suministrar al controlador un método para comunicarse con dispositivos de varios fabricantes y de múltiples tipos (enrutadores, conmutadores, balanceadores de carga) al utilizar una interfaz estandarizada.

La elección del protocolo OpenFlow se debe a la adopción extensiva como habilitador muy importante para la programabilidad de la red. La dirección principal de las SDN aún está ligada al protocolo OpenFlow estandarizado por la ONF, constituyendo la mayor innovación de esta organización.

En la actualidad, la mayoría de los fabricantes de equipos de conectividad de red brindan dispositivos (conmutadores y controladores) con soporte para OpenFlow, por ejemplo HP, CISCO, IBM, Brocade y Juniper, y la vasta mayoría de los controladores SDN actuales se basan en este protocolo, por lo que su elección es evidente. Existen en la actualidad muchas redes experimentales y en explotación basadas en el protocolo OpenFlow. Ejemplos de despliegue e implementación de OpenFlow se encuentran en universidades de Alemania, Brasil, Canadá y Estados Unidos, ejemplo de universidades con SDN, Stanford y Princeton; también se conoce que antes del comienzo del año 2012, la red interna de Google funcionaba completamente con OpenFlow [11].

Precisando, la propuesta que se presenta requiere el soporte del protocolo OpenFlow, por los conmutadores y el controlador, para su implementación.

2.13 Controlador OpenFlow

Un controlador OpenFlow ofrece una interfaz de programación para los conmutadores OpenFlow de tal forma que a través de las aplicaciones de gestión se realicen tareas de monitorización y control. La descripción general de un controlador SDN es un sistema software, o colección de sistemas, que juntos ofrecen [12]:

- Gestión del estado de la red, que implica una base de datos. Estas bases de datos sirven como un repositorio para la información de los elementos de red gestionados, al incluir el estado de la red, información de configuración temporal e información sobre la topología de la red.
- Un modelo de datos de alto nivel que captura las relaciones entre los recursos gestionados, las políticas y otros servicios prestados por el controlador.
- Un mecanismo de descubrimiento de dispositivos, topología y servicio; y un sistema de cálculo de ruta.
- Una sesión de control segura sobre el Protocolo de Control de Trasmisión (TCP,*Transmission Control Protocol* en inglés) entre el controlador y los agentes asociados en los elementos de la red.
- Un protocolo basado en estándares (OpenFlow) para obtener el estado de la red impulsado por las aplicaciones de los elementos de red.
- Un conjunto de API que exponen los servicios del controlador a las aplicaciones de gestión. Esto facilita la mayor parte de la interacción del controlador con estas aplicaciones.

El controlador es un elemento de gestión crítico en las SDN. Los controladores OpenFlow ofrecen algunos servicios de gestión (además del aprovisionamiento y descubrimiento), y son responsables del estado asociado a sus entidades de red [13].

Los conmutadores OpenFlow notifican sus eventos al controlador OpenFlow a través del *framework* de notificaciones OpenFlow.

2.14 Mensajes del protocolo OpenFlow

El protocolo OpenFlow soporta tres tipos de mensaje, controlador-conmutador, asíncrono, y simétrico, cada uno con múltiples subtipos. Los mensajes controlador-conmutador se inician por el controlador y se usan para manejar directamente al conmutador o para inspeccionar el estado de los mismos. Los mensajes asíncronos se inician por el conmutador y se usan para actualizar al controlador de eventos de la red y/o cambios en

el estado de los conmutadores. Los mensajes simétricos se inician por el conmutador o por el controlador y se envían sin solicitud. Los tipos de mensaje que usa OpenFlow se describen a continuación.

2.14.1 Controlador-Conmutador

Los mensajes del controlador-conmutador se inician por el controlador y pueden o no requerir una respuesta del conmutador.

Características: el controlador pide las capacidades de un conmutador enviando una petición de características; el conmutador responde con una respuesta de características que especifique las capacidades del conmutador. Esto se realiza comúnmente en el establecimiento del canal OpenFlow.

- Configuración: el controlador pregunta y establece los parámetros de configuración del conmutador.

El conmutador solo responde a preguntas provenientes del controlador.

- *Modify-State*: estos mensajes los envía el controlador para manejar el estado en los conmutadores. Su propósito primario es añadir/borrar y modificar flujos/grupos en las tablas OpenFlow y establecer las propiedades en los puertos del conmutador.
- *Read-State*: estos mensajes se usan por el controlador para recolectar estadísticas del conmutador.
- *Packet-out:* estos mensajes se usan por el controlador para enviar paquetes de un puerto específico en el conmutador, y para reenviar paquetes recibidos vía mensajes *Packet-in*. Los mensajes *Packet-out* contienen un paquete completo o un identificador del *buffer* que contiene almacenado un paquete en el conmutador. El mensaje también contiene una lista de acciones para ser aplicadas en el orden que se especifican; una lista de acción vacía desecha el paquete.
- *Barrier*: la petición /respuesta de estos mensajes se usan por el controlador para asegurar dependencias de mensajes que son conocidos o para recibir notificaciones por operaciones completadas.

2.14.2 Asíncronos

Los mensajes asíncronos se envían sin necesidad de solicitud por parte del controlador desde los conmutadores. Los conmutadores envían mensajes asíncronos al controlador para denotar la llegada de un paquete desconocido, el cambio en el estado de un

conmutador, o un error. Los cuatro tipos de mensaje asíncronos se describen a continuación:

- *Packet-in*: para todos los paquetes que no tienen una entrada de flujo que les hace juego, un evento de p*acket-in* se envía al controlador (en dependencia de la configuración de la tabla). Para todos los paquetes que se envían para el puerto virtual CONTROLLER, un evento de *packet-in* debe enviarse siempre al controlador. Si el conmutador tiene memoria suficiente para el almacenamiento *(buffer*) del paquete que se envió al controlador, el evento de *packet-in* contiene alguna fracción de la cabecera del paquete (por defecto 128 bytes) y un identificador de *buffer,* que usa el controlador para cuando el conmutador esté listo reenviar el paquete. Los conmutadores que no soportan el almacenamiento interno (*buffering)* envían el paquete completo al controlador como parte del evento. Los paquetes que se almacenan generalmente se procesan vía mensajes *Packet-out* desde el controlador, o automáticamente expiran luego de algún tiempo.
- *Flow-Removed:* cuando una entrada de flujo se añade al conmutador por un mensaje de modificación de flujo, un valor de *idle-timeout* indica cuando una entrada se remueve por falta de actividad, así como también un valor *hard-timeout* que muestra cuando el ingreso se remueve sin importar la actividad. Los mensajes de flujo removido especifican al controlador cuando el conmutador elimina un flujo no válido. Una solicitud de borrar flujo debe generar mensajes de flujos removidos para cualquier flujo con el conjunto OFPFF_SEND_FLOW_REM.
- *Port-status*: el conmutador está en espera para enviar los mensajes de *port-status* al controlador ante cambios en la configuración de los puertos. Estos eventos incluyen cambio en los estados de los puertos (por ejemplo, si lo apaga directamente un usuario).
- *Error*: el conmutador notifica al controlador de problemas usando mensajes de error.

2.14.3 Simétrico

Los mensajes simétricos se envían sin solicitud, en cualquier dirección.

- *Hello*: los mensajes *Hello* se intercambian entre el conmutador y el controlador ante un arranque de conexión.
- *Echo*: los mensajes de petición/respuesta a *Echo* se envían ya sea por el conmutador o el controlador. Se usan para medir la latencia o el ancho de banda

de una conexión del controlador-conmutador, así como también verificar su vitalidad.

- *Experimenter*: los mensajes de *experimenter* proveen una forma estándar a los conmutadores OpenFlow para añadir funcionalidades adicionales dentro del espacio de tipo de mensaje OpenFlow. Este es un área de preparación para características destinadas a revisiones futuras del OpenFlow.

2.15 Resumen de comparación entre conmutadores tradicionales y conmutadores SDN OpenFlow

Para finalizar este capítulo, en la Tabla 2.1 se puntualizan los elementos tratados en cada tipo de conmutador, sus similitudes y diferencias.

Tabla 2.1 Comparación entre conmutadores tradicionales y conmutadores SDN OpenFlow.

Características	Conmutadores tradicionales	Conmutadores SDN OpenFlow
Nivel de Capa OSI	Capa 2 (Capa 3)	Capa 2 (Capa 3)
Segmentan la red en pequeños dominios de colisión	Sí	Sí
Aprendizaje	Sí	Sí, definido por programación.
Inundación	Sí	Sí, definido por programación.
Reenvío	Sí	Sí, definido por programación.
Filtrado	Sí	Sí, definido por programación.
Envejecimiento	Sí	Sí, definido por programación.
Tablas de flujos	Sí, los más actuales.	Sí
Llenado de tablas de flujos	Definido por software implementado en los dispositivos.	Definido por programación.
Reglas de reenvío	Basada únicamente en direcciones MAC.	Basadas en direcciones MAC, y demás campos de cabecera del paquete.
Software y hardware (control y datos)	Altamente ligado	Separado uno de otro.
Plano de control	Distribuido en dispositivos de red. Definido por protocolos implementados y por software de dispositivos.	Centralizado y programable

Control basado en flujos	No	Sí

Como conclusión del capítulo se mostró que los conmutadores SDN OpenFlow presentan similitudes en cuanto a principios de funcionamiento con los conmutadores tradicionales, son dispositivos de red diseñados para adquirir el comportamiento que un programador especifique en un *script* de programación, esto permite gran flexibilidad en el análisis y conformación de estos dispositivos, y posibilidad de experimentación.

En el paquete Mininet existen *script* de programación que definen el comportamiento de conmutadores de capa 2 y conmutadores de capa 3 para implementarlos en los conmutadores SDN OpenFlow. En el próximo capítulo se emula un conmutador SDN OpenFlow utilizando uno de estos *script* de programación. Este *script* se modificó para obtener algunas de las funciones que estos dispositivos tienen en común con los conmutadores tradicionales.

CAPITULO 3 . FUNCIONALIDADES DE CONMUTADORES TRADICIONALES SIMULADAS EN CONMUTADORES OPENFLOW

3.1 Introducción

En el presente capítulo se realizan pruebas sobre varias topologías de redes, con el fin de demostrar las funcionalidades de los conmutadores OpenFlow que coinciden con los conmutadores tradicionales Ethernet. A continuación se listan las funcionalidades que se analizan en el presente trabajo, debido al gran número de funciones que estos dispositivos pueden desarrollar, este trabajo se limita a:

- Redes LAN virtuales (VLAN).
- Enlace troncal para comunicación de VLAN.
- Lista de Control de Acceso (basado en direcciones MAC).
- Cortafuego basado en direcciones IP.
- Reglas de calidad de servicio (*QoS*).

En capítulos anteriores se realiza una exposición del marco teórico referente a las redes definidas por software y a los conmutadores SDN OpenFlow con el fin de obtener los conocimientos teóricos necesarios para realizar la simulación de una red SDN utilizando la herramienta Mininet, ver Anexo I para preparar el entorno de trabajo.

Mininet es una plataforma de simulación de red que crea, interactúa, personaliza y comparte redes definidas por software virtuales totalmente escalables de forma rápida y eficiente gracias al uso de una característica conocida como virtualización ligera *(light-weight virtualization)*, que están contenidas en una PC que utiliza procesamiento Linux. Tiene como característica que las configuraciones realizadas en la simulación pueden moverse a las implementaciones físicas sin mayores cambios; además es una solución poco costosa al estar disponible de forma gratuita, soporta topologías personalizadas y tiene una API basada en *Python* incluida.

La plataforma Mininet permite implementar nodos con protocolo OpenFlow, esto la convierte en muy práctica para el análisis de las SDN. Existen en el mercado otras

herramientas similares a Mininet, sin embargo Mininet se destaca por ser la más rápida de todas y la que tiene una mayor escalabilidad.

3.2 VLAN. Un conmutador y cuatro terminales

Una VLAN (acrónimo de *virtual LAN*, red de área local virtual) es un método para crear redes lógicas independientes dentro de una misma red física. Varias VLAN pueden coexistir en un único conmutador físico o en una única red física. Se utilizan para reducir el tamaño del dominio de difusión y ayudan en la administración de la red, separan segmentos lógicos de una red de área local (como departamentos de una empresa) que no deben intercambiar datos usando la red local (aunque se puede hacer a través de un enrutador o un conmutador de capa 3 y 4).Una VLAN consiste en una red de ordenadores que se comporta como si se conectaran al mismo conmutador, aunque en realidad estén físicamente en diferentes segmentos de una red de área local. Los administradores de red configuran las VLAN mediante software en lugar de hardware, lo que las hace extremadamente flexibles.

Para el desarrollo de este trabajo se utiliza una topología personalizada, a partir de una de las topologías que vienen por defecto en el paquete de Mininet.

Con el objetivo de demostrar que es posible la creación de VLAN, limitando los dominios de difusión y el tráfico en la red, para que sólo se comuniquen aquellos puertos que pertenecen a la misma VLAN, se conforma una topología de un conmutador SDN y cuatro terminales conectados al conmutador, como se aprecia en la Figura 3.1.

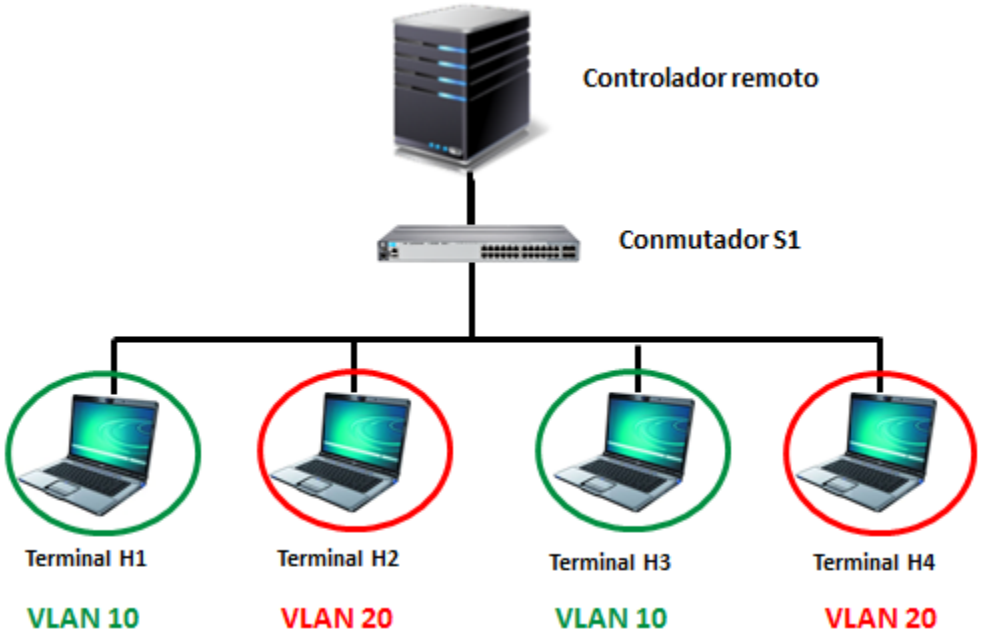

Figura 3.1 Topología usada para demostrar funcionalidad de VLAN

H1 y H3 pertenecen a la VLAN 10 y H2 y H4 pertenecen a la VLAN 20, las direcciones IP se especifican a partir de un *script* de configuración o una vez simulada la red con los comandos pertinentes, el controlador sino se especifican, por defecto establece las siguientes direcciones IP para los terminales:

H1 IP: 10.0.0.1/24
H2 IP: 10.0.0.2/24
H3 IP: 10.0.0.3/24
H4 IP: 10.0.0.4/24

Las direcciones IP de los terminales de la red las asigna el controlador, teniendo en cuenta el orden en que aparecen en el *srcript* de programación de creación de la topología (.py), ver Anexo II. Para levantar la topología personalizada de la red, ver Anexo III.

La topología de la Figura 3.1 se somete, antes de crear las VLAN, a una prueba *pingall* para verificar el estado de conexión entre los terminales de la red, esta prueba realiza un *ping* desde cada uno de los terminales y hacia cada uno de los terminales de la red, el comando que se utiliza, desde el CLI de Mininet es:

>pingall

En la Figura 3.2 se observa el resultado de la ejecución del comando anterior.

Figura 3.2 Resultados de la prueba pingall.

Como se observa en los resultados de la prueba, la conexión entre los terminales de la red es nula, la misma no tiene un mecanismo para descubrir los elementos de la red y los flujos necesarios para establecer la conexión entre ellos. Se abre un *xterm* para el conmutador s1 con el comando:

>xterm s1

De esta manera se trabaja desde el conmutador; esto permite eliminar el controlador que se le asignó, modificar el modo de trabajo del conmutador así como modificar el estado de sus puertos, estas funciones se utilizan para establecer las VLAN. Desde el *xterm* para s1 se utiliza el comando:

 # ovs-vsctl show

El comando (*ovs, OpenFlow vswitch; vsctl, vswitch controller*) muestra del conmutador OpenFlow: su controlador, su modo de trabajo y el estado de sus puertos. En la Figura 3.3se observa una imagen del comando y la respuesta al mismo:

```
root@mininet-vm:~/mininet/custom# ovs-vsctl show
39f1e02b-7043-438d-8ff7-b202899dad97
    Bridge "s1"
        Controller "tcp:127.0.0.1:6633"
        Controller "ptcp:6634"
        fail_mode: secure
        Port "s1-eth1"
            Interface "s1-eth1"
        Port "s1-eth3"
            Interface "s1-eth3"
        Port "s1-eth4"
            Interface "s1-eth4"
        Port "s1"
            Interface "s1"
                type: internal
        Port "s1-eth2"
            Interface "s1-eth2"
    ovs_version: "2.0.1"
```

Figura 3.3 Resultado del comando ovs-vsctl show.

Para establecer las VLAN se elimina el controlador del conmutador, para demostrar que el conmutador por si solo es capaz de reconocer la asignación de banderas VLAN entre sus puertos y permitir la comunicación entre ellos, se cambia el modo de trabajo de el conmutador y se definen los puertos que pertenecen a las VLAN con banderas 10 y 20, ver Anexo IV.Para verificar los cambios se vuelve a ejecutar el comando:

 # ovs-vsctl show

La Figura 3.4 muestra que los cambios se guardaron en el conmutador:

```
root@mininet-vm:~/mininet/custom# ovs-vsctl show
39f1e02b-7043-438d-8ff7-b202899dad97
    Bridge "s1"
        fail_mode: standalone
        Port "s1-eth1"
            tag: 10
            Interface "s1-eth1"
        Port "s1-eth3"
            tag: 10
            Interface "s1-eth3"
        Port "s1-eth4"
            tag: 20
            Interface "s1-eth4"
        Port "s1"
            Interface "s1"
                type: internal
        Port "s1-eth2"
            tag: 20
            Interface "s1-eth2"
    ovs_version: "2.0.1"
```

Figura 3.4 Resultado del comando ovs-vsctl show con los cambios realizados.

En el entorno del Mininet se repite la prueba *pingall*, para observar el correcto funcionamiento de la red con las VLAN establecidas.

Al ejecutar el comando *pingall*, se observa que la conexión en la red sólo se estableció entre los terminales conectados a los puertos que pertenecen a la misma VLAN. En este caso entre los terminales H1 (eth1) y H3 (eth3) con bandera de VLAN 10 y entre los terminales H2 (eth2) y H4 (eth4) con bandera de VLAN 20, como se observa en la Figura 3.5.

Figura 3.5 Resultados de la prueba pingall

3.3 Enlace troncal. Dos conmutadores y cuatro terminales (VLAN)

En este epígrafe se buscó demostrar que era posible tener enlaces troncales que permiten la comunicación entre diferentes VLAN creadas en la red, logrando que solamente se comuniquen aquellos puertos que pertenezcan a las misma VLAN, para ello se conformó una topología de dos conmutadores SDN y cuatro terminales, como se puede apreciar en la Figura 3.6:

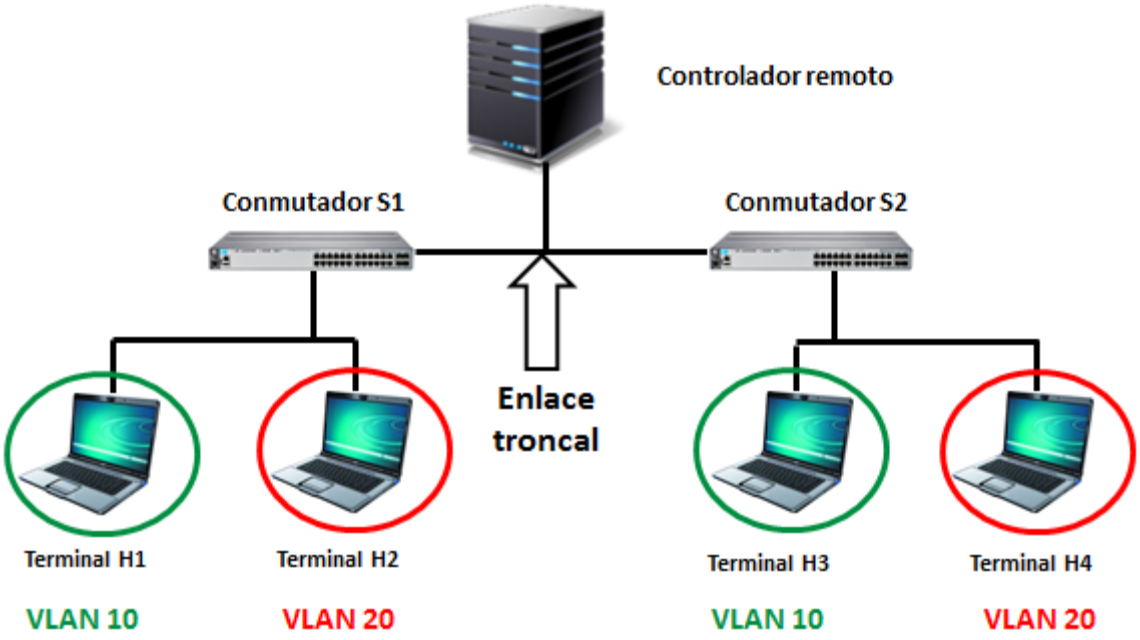

Figura 3.6 Topología usada para demostrar funcionalidad del enlace troncal con VLAN.

H1 y H3 pertenecen a la VLAN 10 y el H2 y H4 pertenecen a la VLAN 20, las direcciones IP se especifican a partir de un *script* de configuración o una vez simulada la red con los comandos pertinentes, el controlador sino se especifican, establece por defecto las siguientes direcciones IP:

H1 IP: 10.0.0.1/24
H2 IP: 10.0.0.2/24
H3 IP: 10.0.0.3/24
H4 IP: 10.0.0.4/24

Las direcciones IP de los terminales de la red las asigna el controlador, teniendo en cuenta el orden en que aparecen en el *srcript* de programación de creación de la topología (.py), ver Anexo V. Para levantar la topología personalizada de la red, ver Anexo VI.

La topología de la Figura 3.6 se somete, antes de crear las VLAN, a una prueba *pingall* para verificar el estado de conexión entre los terminales de la red, esta prueba realiza un *ping* desde cada uno de los terminales y hacia cada uno de los terminales de la red, el comando que se utiliza, desde el CLI de Mininet es:

>pingall

En la Figura 3.7se observa el resultado de la ejecución del comando anterior:

Figura 3.7 Resultados de la prueba pingall.

Como se observa en los resultados de la prueba, la conexión entre los terminales de la red es nula, la misma no tiene un mecanismo para descubrir los elementos de la red y los flujos necesarios para establecer la conexión entre ellos.

Para realizar las modificaciones en los conmutadores se realizan desde un *xterm* del conmutador s1 o desde un *xterm* del conmutador s2, en este caso se realizan los cambios desde el *xterm* del conmutador s1, abrimos un xterm para el conmutador s1con el comando:

>xterm s1

De esta manera se trabaja desde el conmutador; esto permite eliminar el controlador que se le asignó, modificar el modo de trabajo del conmutador así como modificar el estado de sus puertos, estas funciones se utilizan para establecer las VLAN. Desde el *xterm* para s1 se utiliza el comando:

 # ovs-vsctl show

Este comando (*ovs, OpenFlow vswitch; vsctl, vswitch controller*) muestra de los conmutadores OpenFlow: su controlador, su modo de trabajo y el estado de sus puertos. En la Figura 3.8 se observa una imagen del comando y la respuesta al mismo:

```
root@mininet-vm:~/mininet/custom# ovs-vsctl show
39f1e02b-7043-438d-8ff7-b202899dad97
Bridge "s1"
    Controller "ptcp:6634"
    Controller "tcp:127.0.0.1:6633"
    fail_mode: secure
    Port "s1-eth1"
        Interface "s1-eth1"
    Port "s1-eth3"
        Interface "s1-eth3"
    Port "s1"
        Interface "s1"
            type: internal
    Port "s1-eth2"
        Interface "s1-eth2"
Bridge "s2"
    Controller "ptcp:6635"
    Controller "tcp:127.0.0.1:6633"
    fail_mode: secure
    Port "s2-eth3"
        Interface "s2-eth3"
    Port "s2"
        Interface "s2"
            type: internal
    Port "s2-eth1"
        Interface "s2-eth1"
    Port "s2-eth2"
        Interface "s2-eth2"
ovs_version: "2.0.1"
```

Figura 3.8 Resultado del comando ovs-vsctl show.

Para establecer las VLAN se elimina el controlador de los conmutadores, para demostrar que el conmutador por si solo es capaz de reconocer la asignación de banderas de VLAN entre sus puertos y permitir la comunicación entre ellos,se cambiael modo de trabajo de los conmutadores y se definen los puertos que pertenecen a las VLAN con banderas 10 y 20, ver Anexo VII.

Para verificar los cambios se ejecuta el comando:

 # ovs-vsctl show

En la Figura 3.9 se muestra que los cambios se guardan en ambos conmutadores:

```
root@mininet-vm:~/mininet/custom# ovs-vsctl show
39f1e02b-7043-438d-8ff7-b202899dad97
Bridge "s1"
    fail_mode: standalone
    Port "s1-eth1"
        tag: 10
        Interface "s1-eth1"
    Port "s1-eth3"
        Interface "s1-eth3"
    Port "s1"
        Interface "s1"
            type: internal
    Port "s1-eth2"
        tag: 20
        Interface "s1-eth2"
Bridge "s2"
    fail_mode: standalone
    Port "s2-eth3"
        Interface "s2-eth3"
    Port "s2"
        Interface "s2"
            type: internal
    Port "s2-eth1"
        tag: 10
        Interface "s2-eth1"
    Port "s2-eth2"
        tag: 20
        Interface "s2-eth2"
ovs_version: "2.0.1"
```

Figura 3.9 Resultado del comando ovs-vsctl show con los cambios realizados.

En el entorno del Mininet se repite la prueba *pingall*, para observar el correcto funcionamiento de la red con las VLAN establecidas.

Al ejecutar el comando *pingall*, se observa que la conexión en la red sólo se estableció entre los terminales conectados a los puertos que pertenecen a la misma VLAN. En este caso entre los terminales H1 (eth1) y H3 (eth3) con bandera de VLAN 10 y entre los terminales H2 (eth2) y H4 (eth4) con bandera de VLAN 20, como se observa en la Figura 3.10.

Figura 3.10 Resultados de la prueba pingall

El resultado de esta prueba muestra que el enlace entre los puertos eth3 de ambos conmutadores es un enlace troncal que permite que las VLAN a lo largo de la red se comuniquen.

3.4 Lista de Control de Acceso (ACL), basado en direcciones MAC

Una lista de control de acceso o ACL (del inglés, *Access Control List*) es un concepto de seguridad informática que se usa para fomentar la separación de privilegios. Es una forma de determinar los permisos de acceso apropiados a un determinado objeto, al tener en cuenta aspectos del proceso que hace el pedido. Las ACL permiten controlar el flujo del tráfico en equipos de redes, tales como enrutadores y conmutadores. Su principal objetivo es filtrar tráfico, es decir, permitir o denegar el tráfico de red de acuerdo a alguna condición. Las listas de control de acceso se configuran generalmente para controlar tráfico entrante y saliente y en este contexto son similares a un cortafuego. Un Sistema de Control de Accesos, administra el ingreso a áreas restringidas, y evita así que personas no autorizadas o indeseables tengan la libertad de acceder a estas áreas.

Si se tiene un punto de acceso del campus de una universidad o de una empresa, se sobrentiende que el acceso a las redes de estas instituciones únicamente se habilite para aquellos equipos que pertenecen a personal autorizado de estos centros, por eso se crea una lista con todas las direcciones MAC que el personal puede utilizar para acceder a estas redes, ya sean de teléfonos móviles, laptop, tablet, agendas personales, etc.

El objetivo es que al identificar un dispositivo cuya dirección MAC no se incluya en la lista, se debe de instalar una entrada de flujo en la cual se defina que todo el tráfico que intente ingresar desde esa dirección se deseche.

En el Anexo VIII se visualiza el *script* de programación del componente l2_learning, componente que se utiliza de base sobre el cual se realizan las modificaciones pertinentes a fin de obtener los componentes de todo el trabajo.

Este componente obliga al conmutador OpenFlow a comportarse como un conmutador capaz de aprender direcciones de capa 2 (direcciones MAC).Las entradas de flujos que se instalan deben coincidir con la mayor cantidad de campos posible, de manera que para diferentes conexiones TCP resulta en diferentes flujos instalados, aunque las direcciones de capa 2 coincidan, de igual manera sucede con los demás campos de la regla de coincidencia de la entrada de flujo.

En los entornos de prueba que se muestran en los próximos epígrafes, se pueden utilizar las topologías que están creadas en el paquete del Mininet o alguna personalizada. En este trabajo se empleará la topología sencilla de un conmutador y cuatro terminales (esta aclaración es válida para el resto del trabajo), esta topología es práctica para evaluar y mostrar resultados con fines académicos.

En la Figura 3.11 se muestra la topología:

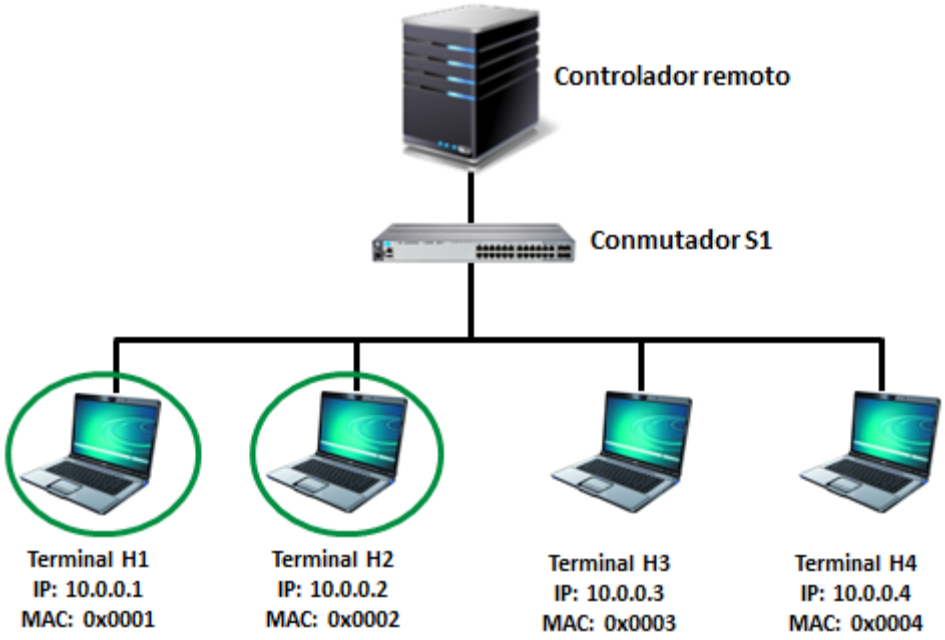

Figura 3.11 Topología empleada.

Antes de realizar los cambios se ejecuta en la red una prueba *pingall*, y se comprueba que la conectividad entre los terminales de la red es completa.

Para obtener el ACL se crea un arreglo o un diccionario, que es un tipo de arreglo que relaciona dos valores, dentro del *script* de programación del componente l2_learning.py, en el cual se incluyen las direcciones MAC que pertenecen a la lista de control de acceso.

Es necesario tener cuidado en relación a la posición en donde se añaden las líneas de código referente al ACL en el *script* de programación, pues es importante darle el grado de globalidad que requiere para que se invoque desde cualquier función o método del *script*, en este ejemplo, de los cuatro terminales de la red dos se incluyen en el ACL, el terminal 1 y 2, ver Anexo IX.

Al definir las direcciones que pertenecen al ACL, éstas serán las únicas direcciones que se autorizan a utilizar la red. Al crear las líneas de código que limita el tráfico se coloca con la prioridad suficiente para definir quién se comunica y a la vez no entorpecer con el funcionamiento del dispositivo. Como se explicó en capítulos anteriores, una vez que al conmutador llega un paquete desconocido, este encapsula parte del paquete en un mensaje *packet_in* y lo envía al controlador para decidir el tratamiento del nuevo paquete, que define un nuevo flujo. En la función *_handle_PacketIn,* que se encarga de analizar los

mensajes *packet_in* que llegan al controlador, se agregan las líneas de código que dan vida al ACL y permiten la creación de entradas de flujo para eliminar los paquetes que provienen de las direcciones ajenas al ACL.

Una vez que se inicie la red con los cambios guardados en el *script* de programación, se prueba el componente.

En la sesión SSH que tiene activa la topología se realiza una prueba *pingall* para conocer el estado de conectividad entre los terminales de la red, los resultados se muestran en la Figura 3.12:

Figura 3.12 Resultados de la prueba pingall.

Se observa que la conectividad tuvo éxito sólo entre los terminales cuyas direcciones MAC aparecen en el ACL, en este caso el terminal 1 y el terminal 2. Las entradas de flujos que se instalan especifican que para las direcciones que no constan en el ACL, la acción asociada a los paquetes de estos flujos es desechar, esto se apreciar al introducir el comando:

dpctl dump-flows

Este comando permite acceder a las entradas de flujo instaladas en el conmutador, el resultado se observa en la Figura 3.13:

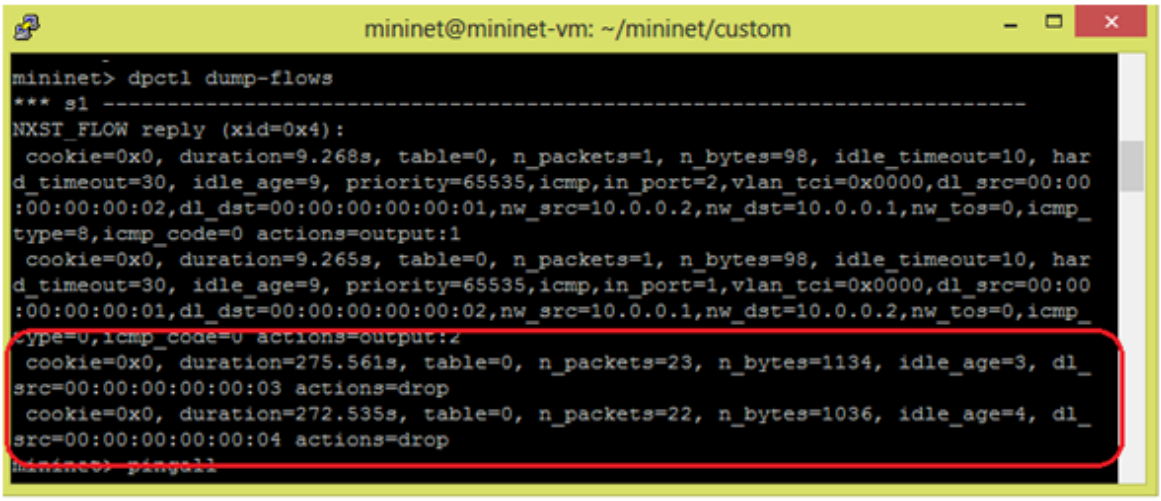

Figura 3.13 Tabla de flujos del conmutador.

Al verificar que las entradas de flujos se instalaron se corrobora que es posible la implementación de una lista de control de acceso en un conmutador SDN OpenFlow a partir de la modificación de una *script* de programación que describe el comportamiento de uno de los componentes que ya vienen creados (en este caso un conmutador L2).

3.5 Cortafuego basado en direcciones IP

Un cortafuego es una parte de un sistema o una red que se diseña para bloquear el acceso no autorizado, y permitir, al mismo tiempo, comunicaciones autorizadas. Se trata de un dispositivo o conjunto de dispositivos que se configuran para permitir, limitar, cifrar, descifrar el tráfico entre los diferentes ámbitos sobre la base de un conjunto de normas y otros criterios.

Los cortafuegos se implementan en hardware, en software, o una combinación de ambos. Se utilizan con frecuencia para evitar que los usuarios de Internet no autorizados accedan a redes privadas conectadas a Internet, especialmente intranets. Todos los mensajes que entren o salgan de la intranet pasan a través del cortafuego, que examina cada mensaje y bloquea aquellos que no cumplen los criterios de seguridad que se especifican.

En este caso se realiza un cortafuego basado en direcciones IP, un ejemplo para su aplicación es en la red de la Universidad de Oriente donde se quiere separar los segmentos de red de profesores y estudiantes del segmento de red de Secretaria, dando acceso a este último a aquellas direcciones IP de los terminales que pertenecen a profesores autorizados a consultar estos datos.

Antes de realizar los cambios se ejecuta en la red una prueba *pingall*, y se comprueba que la conectividad entre los terminales de la red es completa.

De igual manera que en el ACL se crea una lista con todas las direcciones IP autorizadas a tener el acceso a este segmento, una vez que al conmutador llegue un paquete que proviene de una dirección IP limitada los paquetes se desechan acorde a las entradas de flujos previamente instaladas, para este ejemplo se crea en el *script* de programación un diccionario donde se relacionen las direcciones IP autorizadas y dentro de la clase *LearningSwitch* una función llamada *firewall(),*que se invoca para la verificación de accesibilidad de los terminales, ver Anexo X.

Se recalca la necesidad de tener cuidado al insertar las líneas de códigos que definen el cortafuego en el *script* de programación, pues es importante darle el grado de globalidad que requiere para invocarlo desde cualquier función o métodos del *script*, en este ejemplo de los 4 terminales de red, dos se incluyen en el cortafuegos, los terminales 1 y 2.

Una vez que se active el controlador con los cambios guardados y se cree la topología, en la sesión SSH que tiene activa la topología se realiza una prueba *pingall* para conocer el estado de conectividad entre los terminales de la red, los resultados se muestran en la Figura 3.14:

Figura 3.14 Resultados de la prueba pingall.

Se observa que la conectividad tuvo éxito solo entre los terminales cuyas direcciones IP aparecen en la lista, en este caso el terminal 1 y el terminal 2. Las entradas de flujos que se instalan especifican que para las direcciones que no constan en la lista, la acción asociada a los paquetes de estos flujos es desechar, esto se aprecia al introducir el comando:

dpctl dump-flows

Este comando permite acceder a las entradas de flujo instaladas en el conmutador. Se observa que existe un gran número de entradas de flujos destinadas a descartar paquetes con igual direcciones IP porque el componente original para establecer las entradas de flujos hace que coincidan la mayor cantidad de campos posible. Si se mantiene la dirección IP de origen y varía la dirección IP de destino se instala una nueva entrada de flujo, sucede de igual manera si se varía la conexión TCP, el protocolo que se utilice o algún otro campo de la cabecera del paquete, el resultado del comando se observa en la Figura 3.15:

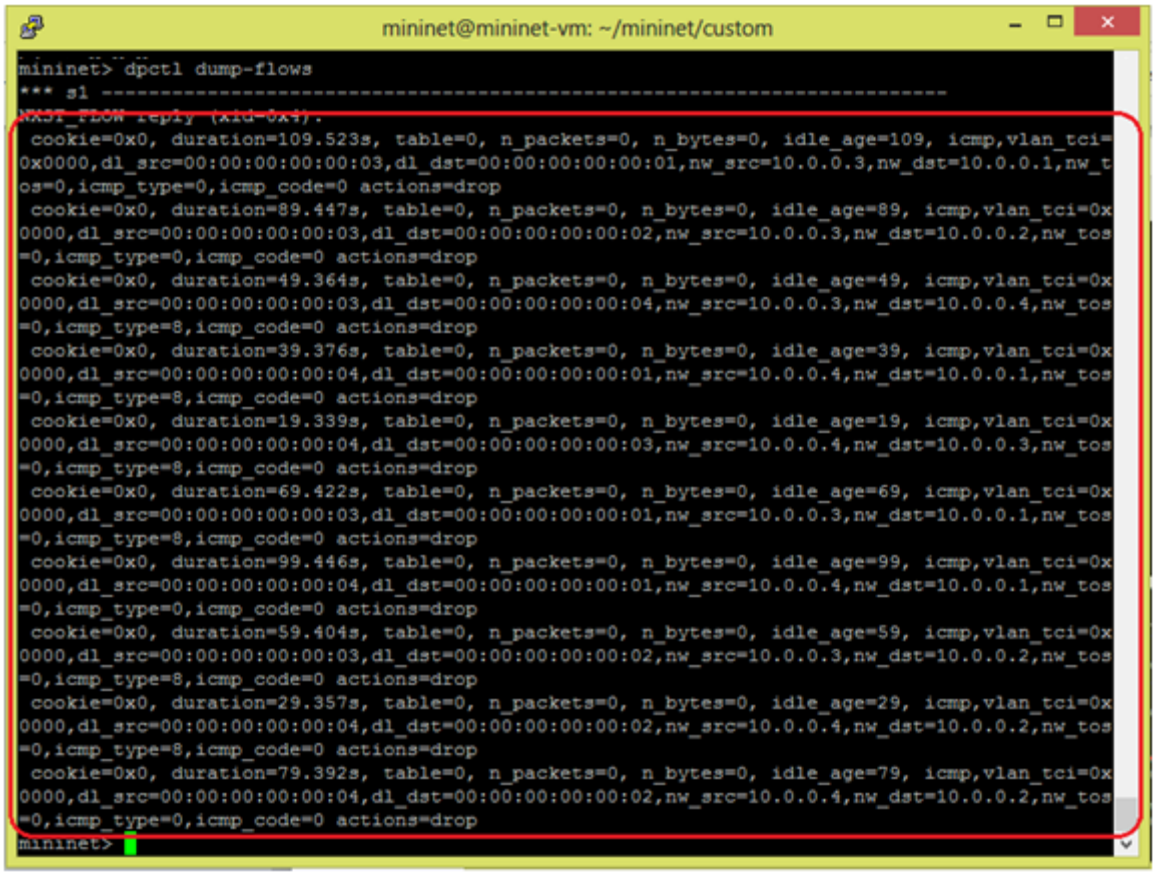

Figura 3.15 Tabla de flujos del conmutador.

Puede ocurrir que algunas entradas de flujos no se encuentren en la imagen, esto se debe a que se eliminaron porque expiró el tiempo de *idle_timeout* y *hard_timeout*, generalmente estos tiempos se mantienen bajos de manera que las tablas de flujos del conmutador no se saturen de entradas de flujos inutilizadas.

Al verificar que las entradas de flujos se instalan, se corrobora que es posible la implementación de un cortafuego basado en direcciones IP en un conmutador SDN

OpenFlow, a partir de la modificación de una *script* de programación, que describe el comportamiento de uno de los componentes que ya vienen creados (en este caso un conmutador L2).

El cortafuegos se basó en direcciones IP, pero con la creación de la función *firewall(),*se logra que esta se invoque cada vez que el programador lo necesite, ya sea para eliminar tráfico teniendo en cuenta una dirección IP de origen o destino, el puerto del conmutador de origen o destino, el puerto de conexión de origen o destino, o cualquier otra condición o criterio que se desea implementar.

3.6 Reglas de Calidad de Servicio (QoS) trabajando con velocidad de transmisión

Calidad de Servicio o QoS (*Quality of Service*, en inglés) es el rendimiento promedio de una red de telefonía o de computadoras, particularmente el rendimiento visto por los usuarios de la red. Para medir cuantitativamente la calidad de servicio se consideran varios aspectos del servicio de red, tales como tasas de errores, ancho de banda, velocidad de transmisión, rendimiento, retraso en la transmisión, disponibilidad, *jitter*, etc.

Se dice que una red o un proveedor ofrece QoS cuando garantiza el valor de uno o varios de los parámetros que definen la calidad de servicio que ofrece la red. Si el proveedor no se compromete en ningún parámetro se dice que lo que ofrece es un servicio '*besteffort'*.

La calidad de servicio es particularmente importante para el transporte de tráfico con requerimientos especiales. En este epígrafe, se crea una entrada de flujo que define la velocidad de transmisión para el tráfico en un puerto fijo. Este parámetro se comprueba con la prueba *iperf* que calcula, mediante la creación de tráfico, la velocidad de transmisión entre dos puntos finales de una comunicación. El objetivo de esta prueba es dar prioridad o tratamiento especial a ciertos enlaces sobre otros, en dependencia del tipo de información que transmitan.

En este epígrafe se define en el terminal 1 un cliente y en el terminal 3 y 4 servidores cuyos puertos de escucha serán el 5001 y el 444 respectivamente, ver Figura3.16:

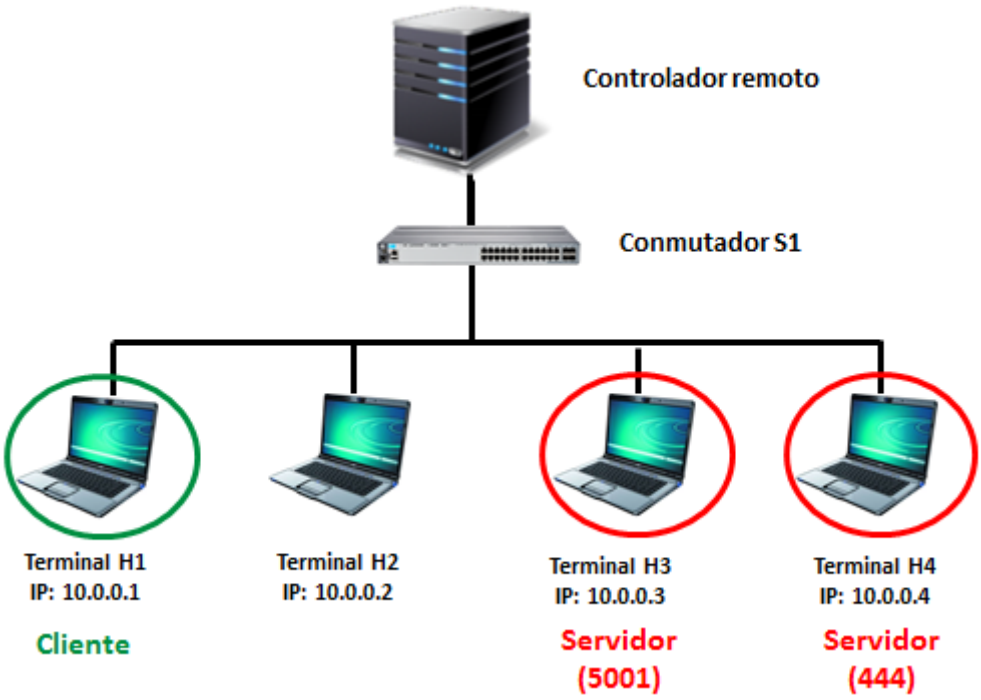

Figura 3.16 Esquema de trabajo empleado.

En el *xterm* del terminal 1 se especifica que será un cliente (-c, cliente) a la hora de comprobar las velocidades de transmisión de la conexión con los servidores del terminal 3 y del terminal 4, en ambos sentidos, de cliente a servidor y de servidor a cliente (-r, respuesta), esta prueba se realiza antes de establecer la regla de calidad de servicio, se utilizan los siguientes comandos respectivamente, dándole el tiempo necesario para que la tarea se complete exitosamente:

iperf –c 10.0.0.3 –p 5001 –r

iperf –c 10.0.0.4 –p 444 –r

Al ingresar estos comandos los resultados de las pruebas se muestran en la Figura 3.17:

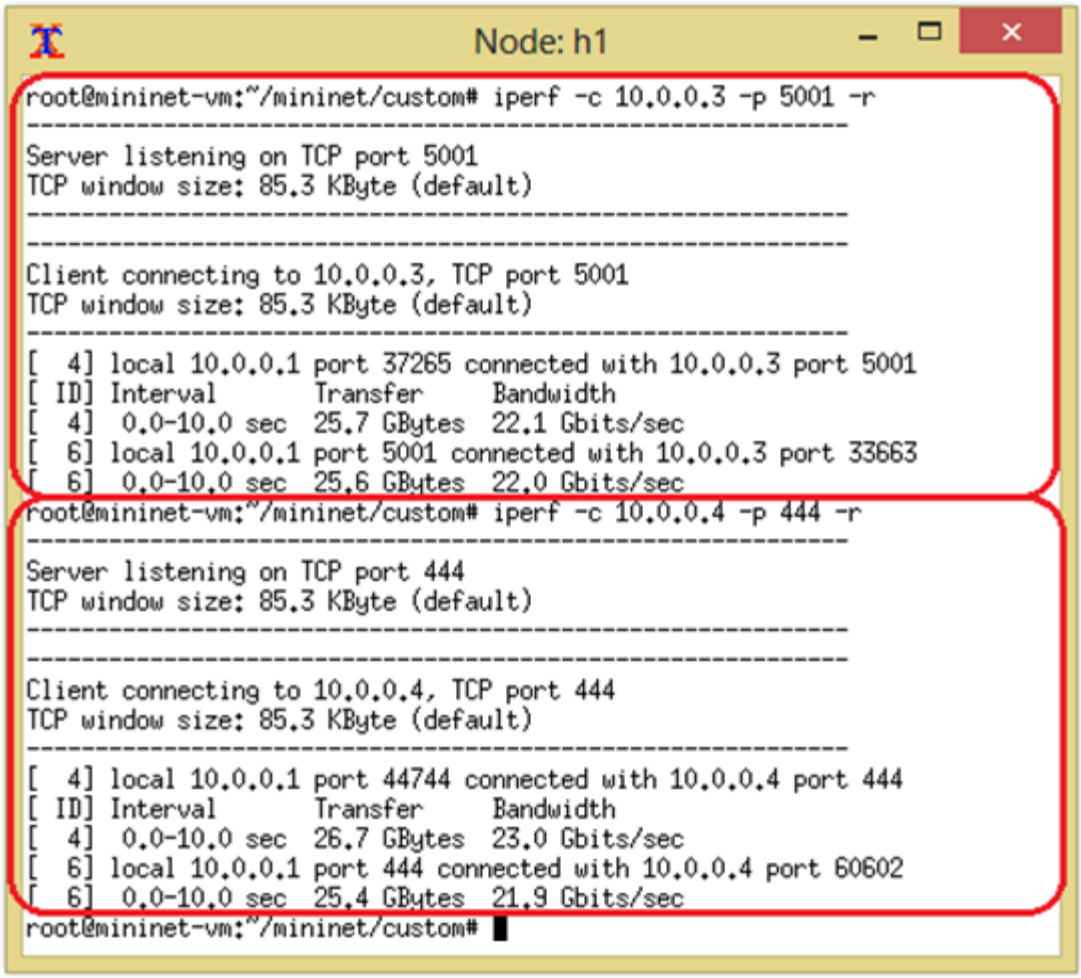

Figura 3.17 Resultados de las pruebas iperf hacia terminal 3 y terminal 4.

Como se aprecia en la figura anterior la velocidad de transmisión por defecto en los enlaces del Mininet oscilan alrededor de los 20 Gbits/sec.

La regla de calidad de servicio que se instala restringe todo el tráfico saliente en el puerto s1-eth1 del conmutador, es decir de conmutador a terminal, con una velocidad de transmisión que oscila entre valores de 4Mbits/sec, ver Anexo XI.

Al establecer la regla de QoS se repiten las pruebas *iperf* para observar el resultado de los cambios realizados, los cuales aparecen en la Figura 3.18:

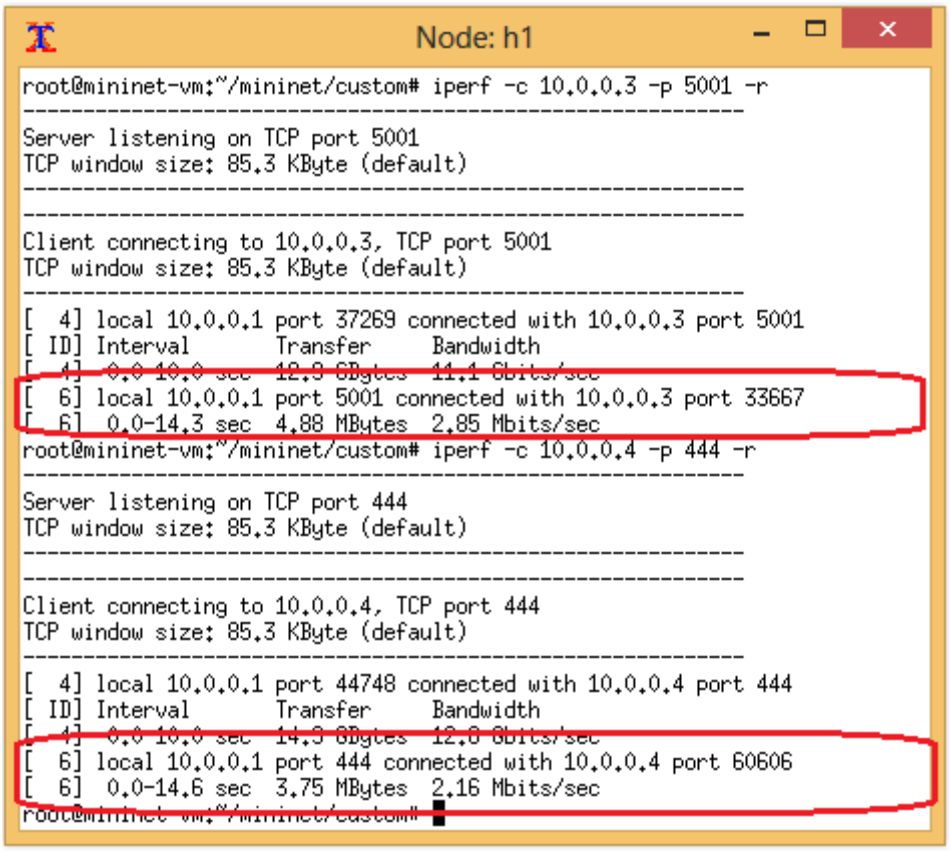

Figura 3.18 Resultados de las pruebas iperf hacia terminal 3 y terminal 4 con la regla de QoS implementada.

Como se observa en la figura anterior los valores de velocidad de transmisión de los enlaces de servidor a cliente se encuentran alrededor de los 4Mbit/sec, como se estableció en la regla de QoS, se observa que este método es solo de egreso, es decir, cuando el conmutador reenvía paquetes desde su interfaz eth1 hacia el terminal 1.

Un efecto secundario de este método es la reducción de la velocidad de transmisión de ingreso en ese puerto, que se reduce a la mitad de su valor original. Esto resulta del cuello de botella que surge en el procesamiento del conmutador virtualizado.

3.6.1 Calidad de servicio selectiva, basado en puertos TCP

En este epígrafe, en vez de trabajar con una sola cola (*queue* 0, cola que se aplica por defecto) en el puerto s1-eth1, se trabaja con 2 colas, en una se coloca el tráfico que fluye sin restricciones (con una velocidad de transmisión máxima de 1Gbits/sec), y en la otra el tráfico que se regula a una velocidad de transmisión de 4Mbits/sec. El tráfico que se regula tiene como puerto de origen o puerto de destino el puerto 444, el restante tráfico fluye sin restricciones. Para regular el tráfico que fluye a través del puerto 444, se añade

el código que verifica el puerto de origen y destino del tráfico, y en caso de utilizar el puerto 444 las entradas de flujos que se instalan especifican que el mismo se procesa por la cola 1 (*queue* 1=@q1) a la salida del puerto s1-eth1, donde se implementaron las reglas de QoS en nuestra red, ver Anexo XII.

Al repetir las pruebas se obtienen los resultados que se muestran en la Figura 3.19:

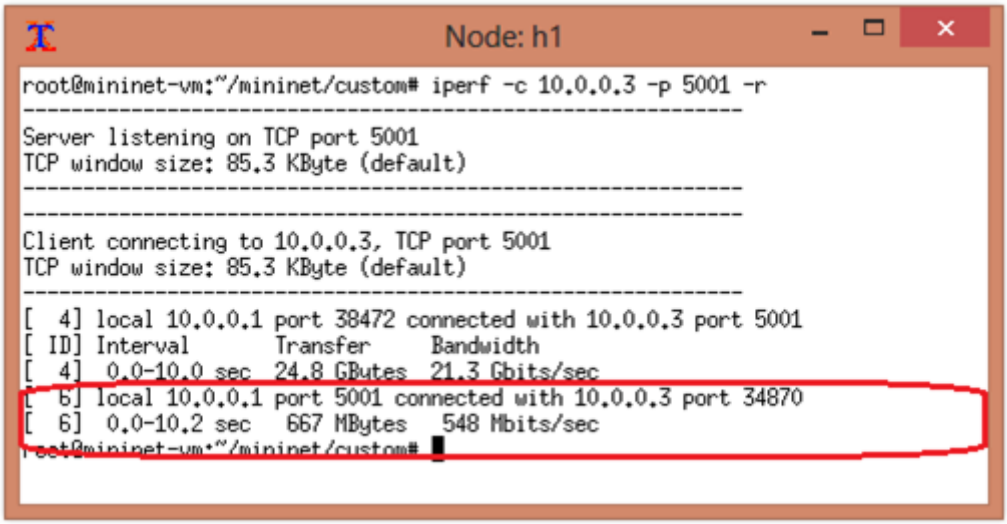

Figura 3.19 Resultado de las prueba iperf hacia terminal 3 con la regla de QoS selectiva implementada.

En la figura anterior se observa que el tráfico sin restricciones se transmite con una velocidad de transmisión que oscila entre los 900Mbits/sec ~ 200Mbits/sec, sujeto a la cola 0 que se establece por defecto para el tráfico sin relación con el puerto 444, a la salida del puerto s1-eth1. En la Figura 3.20se observa como el tráfico que tiene relación con el puerto 444 en la salida s1-eth1 del conmutador s1 (de conmutador a terminal) se regula a una velocidad de transmisión que oscila en valores cercanos a los 4Mbits/sec, como se define en la cola 1:

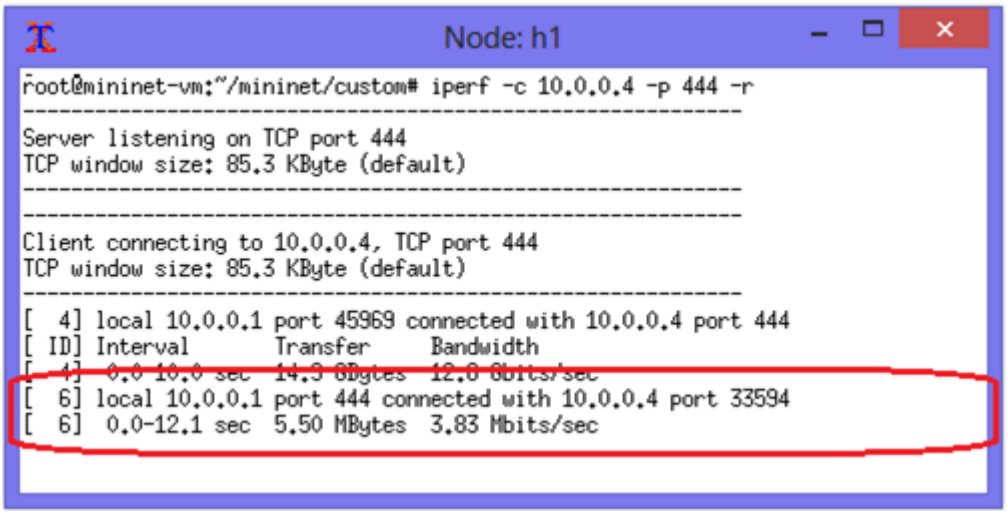

Figura 3.20 Resultado de las prueba iperf hacia el terminal 4 con la regla de QoS selectiva implementada.

Para corroborar que las entradas de flujos se establecen se utiliza en la sesión SSH donde se trabaja con la red el comando:

dpctl dump-flows

En la Figura 3.21se observa el resultado de ejecutar el comando, aparece la tabla de flujos con las entradas de flujos que se instalan al realizar la pruebas *iperf*, las entradas de flujos enmarcadas en rojo definen el tratamiento al tráfico del puerto 444, la acción asociada es procesar los paquetes en la cola 1:

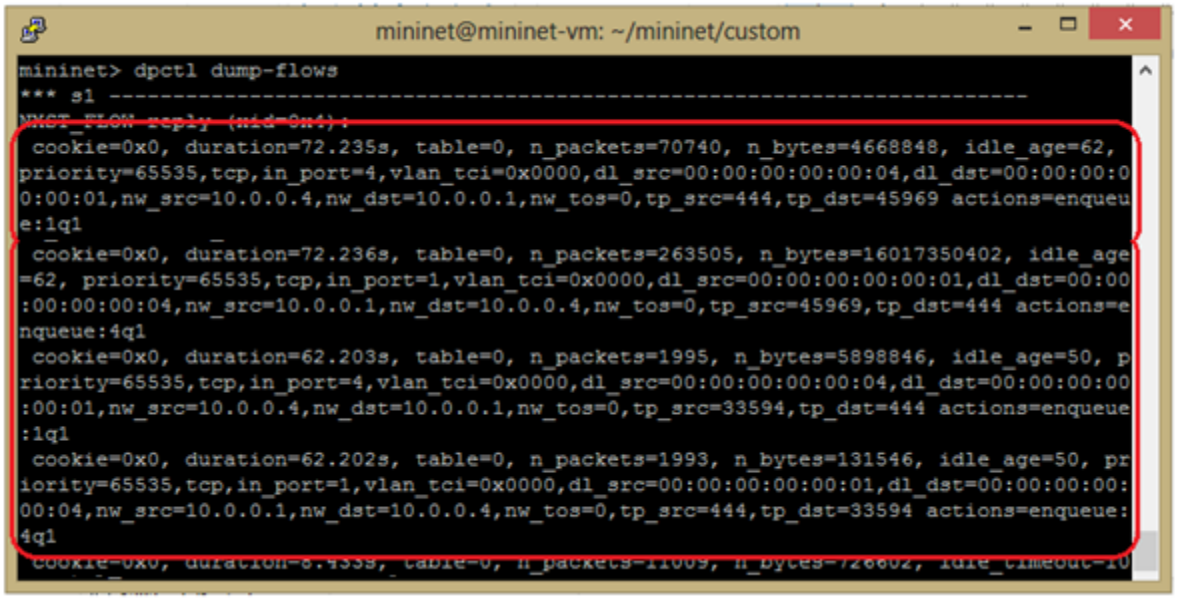

Figura 3.21 Tabla de flujos del conmutador s1.

Al verificar en los resultados de las pruebas *iperf* que el tráfico se limita a una velocidad de transmisión definida y que en la tabla de flujos las entradas de flujos que se crean utilizan las reglas de calidad de servicio, se comprueba la aplicación de QoS en conmutadores SDN OpenFlow.

Las reglas de calidad de servicio no se limitan a puertos TCP sino que se implementan también teniendo en cuentas otros requisitos, ejemplo direcciones IP, direcciones MAC, puertos del dispositivo, etc.

Como resumen del capítulo se presenta la Tabla 3.1, donde se listan las funcionalidades que se implementaron en conmutadores SDN OpenFlow y que tienen en común con los conmutadores tradicionales:

Tabla 3.1 Resumen de algunas de las similitudes en cuanto a funcionalidades entre conmutadores tradicionales y conmutadores SDN OpenFlow.

Funcionalidades	Conmutadores tradicionales	Conmutadores SDN OpenFlow
VLAN	Sí,	Sí
Enlace troncal (VLAN)	Sí	Sí
ACL (MAC)	Sí	Sí
Cortafuegos (IP)	Sí	Sí
QoS (puerto TCP)	Sí	Sí

La implementación de las funcionalidades va a depender de la capacidad y el ingenio del programador a la hora de escribir el código para darle vida al componente que se desea crear. Las funciones que se emplean en el trabajo se pueden combinar para crear componentes más especializados y específicos. Si se tiene un número limitado de terminales autorizados a acceder a una red, con zonas restringidas para algunos usuarios y que brinda servicios con determinada calidad de servicio, sería necesario combinar el ACL, el cortafuego y las reglas de calidad de servicios que se probaron anteriormente. Es importante darle tal orden de prioridad a las funciones que permita que una no entorpezca a la otra y la red se comporte como se desee.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

En el presente trabajo se realizó una caracterización de la tecnología SDN y del protocolo OpenFlow, se analizaron las particularidades de los conmutadores SDN OpenFlow y se realizó una comparación, en cuanto a principios de funcionamiento y funcionalidades, entre este y los conmutadores Ethernet tradicionales. Al culminar este trabajo se arribaron a las siguientes conclusiones:

1. Las SDN, como tecnología novedosa, introducen una nueva etapa en el desarrollo de las redes que es necesario conocer por el personal que las administra.
2. Se mostró que los conmutadores SDN OpenFlow presentan similitudes en cuanto a principios de funcionamiento con los conmutadores tradicionales en el proceso de *bridging* transparente, en ambos se ejecutan las 5 partes del proceso.
3. Los conmutadores SDN OpenFlow son dispositivos de red diseñados para ejecutar el comportamiento que un programador especifique en un *script* de programación, esto permite gran flexibilidad en el análisis y conformación de estos dispositivos, y posibilidades de experimentación.
4. Se comprobó mediante la emulación del conmutador SDN OpenFlow, en Mininet, un grupo de funciones que tienen en común los conmutadores tradicionales y los conmutadores SDN OpenFlow.
5. Se mostró que es posible la sustitución de los conmutadores tradicionales por conmutadores SDN OpenFlow, que facilitan el trabajo de los administradores de red y disminuyen el costo de operación y montaje de las redes de datos.
6. La importancia de las SDN en su implementación, radica en la facilidad que brinda a los administradores de red a la hora de configurar, mediante programación y de forma centralizada, la red y sus servicios en tiempo real logrando desplegar nuevas aplicaciones y servicios en cuestión de horas o días en lugar de semanas o meses necesarios en las tecnologías de redes tradicionales.

7. La adopción de las SDN permiten mejorar significativamente tanto la capacidad de gestión y control, como la escalabilidad y agilidad de la red. Las SDN son de fácil implementación, incrementan la velocidad y el valor de los servicios de la red.

Recomendaciones

- Se propone al sistema de educación superior y en particular a la Universidad de Oriente promover el estudio de esta novedosa tecnología, al incluir este contenido en los objetivos de las asignaturas de la rama de redes de computadoras.
- Se propone crear un sitio en la red en donde el estudiante pueda acceder a bibliografía sobre las SDN, y encontrar prácticas de laboratorio para familiarizarse con esta tecnología.

REFERENCIAS BIBLIOGRÁFICAS

[1] E. P. Sánchez, mayo 2015. [En línea]. Available: http://latablilla.uo.edu.cu/inauguran-nueva-infraestructura-informática-en-la-universidad-de-Oriente.

[2] «Software-Defined Networking: The New Norm For Networks ONF White Paper». 13 April 2012.

[3] F. U. Corporate Headquarters Fort Lauderdale, C. U. Silicon Valley Headquarters Santa Clara, S. EMEA Headquarters Schaffhausen, I. India Development Center Bangalore y O. Div, SDN 101: An Introduction to Software Defined Networking",.

[4] M. Rouse, June 2012. [En línea]. Available: http://whatis.techtarget.com/definition/OpenFlow.

[5] J. R. Martinez, «Switch Ethernet Capa 2».

[6] F. P.-C. N. A. P.-. R. Academy. [En línea]. Available: Como funcionan los switches LAN.

[7] H. N. Sean Odom, Cisco Switching Black Book, 2001.

[8] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, S. Shenker, L. Peterson, J. Rexford y J. Turner, «OpenFlow: Enabling Innovation in Campus Networks,» 2008.

[9] B. Pfaff, B. Lantz, B. Heller, C. Barker, D. Cohn, D. Talayco, D. Erickdon, E. Crabbe, G. Gibb, G. Appenzeller, J. Tourrilhes, J. Pettit, K. Yap, P. Balland, L. Poutievski, M. Casado, M. Takahashi, M. Kobayashi, N. McKeown, P. Balland, R. Ramanathan, P. Reid, R. Sherwood, S. Das, T. Yabe, Y. Yiakoumis y Z. Lajos kis, «OpenFlow Switch Specification Version 1.1.0 Implemented». 28 February 2011.

[10] April 2012. [En línea]. Available: http://searchnetworking.techtarget.com/news/2240248631/Open-Networking-Summit-2012-Take-yourpropietary-networking-and.

[11] E. E. B. V. Rebeca Mayumi Park Campos, «Despliegue y evaluación de desempeño de una red OpenFlow,» Noviembre.

[12] 2008. [En línea]. Available: www.OpenFlow.org/wk/index.php/OpenFlow_Tutorial.

[13] B. D. Heller, Thesis augmented, Standford, California, 2013.

GLOSARIO DE TÉRMINOS

ACL: *Acces Control List*, una lista de control de acceso en español, es un concepto de seguridad informática usado para fomentar la separación de privilegios. Es una forma de determinar los permisos de acceso apropiados a un determinado objeto, dependiendo de ciertos aspectos del proceso que hace el pedido. Las ACL permiten controlar el flujo del tráfico en equipos de redes, tales como enrutadores y conmutadores. Su principal objetivo es filtrar tráfico, permitiendo o denegando el tráfico de red de acuerdo a alguna condición.

Conmutadores OpenFlow: conmutadores que tienen implementado el protocolo OpenFlow usados en las redes SDN.

Cortafuegos: es una parte de un sistema o una red que está diseñada para bloquear el acceso no autorizado, permitiendo al mismo tiempo comunicaciones autorizadas. Se trata de un dispositivo o conjunto de dispositivos configurados para permitir, limitar, cifrar, descifrar, el tráfico entre los diferentes ámbitos sobre la base de un conjunto de normas y otros criterios.

Dijkstra, Algoritmo: también llamado algoritmo de caminos mínimos, es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de vértices en un grafo con pesos en cada arista. Su nombre se refiere a Edsger Dijkstra, quien lo describió por primera vez en 1959.

Mininet: software diseñado para gestionar y simular redes SDN.

ONF: Fundación de Redes Abiertas en español, es el grupo que más relacionado esta con el desarrollo y la estandarización de las SDN.

OpenFlow: protocolo de control que permite la comunicación entre el controlador y el o los conmutadores en la red.

Python: es un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional.

SDN: Redes Definidas por Software en español, tecnología de redes que propone la separación del plano de control del plano de datos en los dispositivos de red, de manera que el control de la red sea gestionado y administrado por una sola entidad, el controlador, capaz de realizar los cambios en la misma a modo de adaptarse en tiempo real de una manera eficaz a las demandas de la red.

TDMA: La multiplexación por división de tiempo (*Time Division Multiple Access*) es una técnica que permite la transmisión de señales digitales y cuya idea consiste en ocupar un canal (normalmente de gran capacidad) de transmisión a partir de distintas fuentes, de esta manera se logra un mejor aprovechamiento del medio de transmisión. El acceso múltiple por división de tiempo (TDMA) es una de las técnicas de TDM más difundidas.

VLAN: método que permite crear redes lógicamente independientes dentro de una misma red física, muy utilizados en las redes SDN para separar el tráfico de producción del tráfico experimental.

ANEXOS

Anexo I. Preparando el entorno de trabajo.

A continuación se detallan las herramientas de software necesarias para realizar las simulaciones que hacen posible que el trabajo entre administrador de red y red sea más fácil.

Componentes de software.

Para la simulación de una red SDN con Mininet, los elementos de software necesarios son:

- Un administrador de máquinas virtuales.
- Los sistemas de ventanas X, que se instalan en la máquina física que contiene el administrador de máquinas virtuales.
- Una imagen de una máquina virtual que traiga habilitado el software Mininet.
- Un cliente de Secure Shell (SSH), que también se instala en este dispositivo.

Instalando Mininet

El administrador de máquinas virtuales VMWare, que escogió el diplomante, te permite correr varias máquinas virtuales dentro de una máquina física, y es gratis y habilitado para Windows, Linux, y para Mac, bajo el nombre de VMWare Fusion.

Usted necesita descargar los archivos correspondientes con el sistema operativo en el que trabaje, por ejemplo, para este trabajo se descargó la imagen comprimida de la siguiente máquina virtual:**mininet-2.1.0p2-140718-ubuntu-14.04-server-i386**

Activar la máquina virtual

Importar la imagen de la máquina virtual. Si la imagen .ovf de la máquina virtual ya está previamente descargada en su ordenador:

- Iniciar el administrador de máquinas virtuales, una vez iniciado el VMware, que fue el administrador que seleccionó el diplomante, seleccionar *File>Open*… y luego una vez en la carpeta contenedora de la imagen seleccionar el archivo .ovf que descargó.
- Posteriormente, presionar el botón *Import*, este paso puede requerir algún tiempo debido al tamaño de la imagen importada.

- Continuar con el botón de Finalizar.

Para finalizar la activación de la máquina virtual necesita completar un paso más antes de terminar con la instalación. Seleccione su máquina virtual y en *Setting Tab*. En *Network>Adapter 2*. Seleccionar *"Enable adapter*", y añada*"host-only network*". Esto le permite acceder fácilmente a su máquina virtual a través de su máquina física.

En este punto usted debe ser capaz de iniciar su máquina virtual. Presione el ícono de "*Power on this virtual machine".*

Una vez en la ventana de consola de la máquina virtual, introducir el nombre de usuario y la contraseña de su máquina virtual.

Note que el usuario es un *user*, entonces usted podrá ejecutar comandos con permisos de *root* al escribir ***sudo commands***, donde ***commands*** es el nombre del comando que usted desea ejecutar.

Acceder a la VM vía SSH

En este paso, usted verifica que se puede conectar desde su computadora personal a la VM alojada vía SSH. Desde la consola de la máquina virtual, *log* en la máquina virtual, luego introduzca:

$ ifconfig –a

Usted debe ver tres interfaces (eth0, eth1, lo), ambas eth0 y eth1 deben de tener una dirección IP asignada. Si este no es el caso, escriba:

$ sudo dhclient ethX

Reemplazar ethX con el nombre de la interface apagada, en algunos casos los puertos eth aparecen como eth2 o eth3, usted puede cambiar esto editando /etc/udev/rules.d/70-persisten-net.rules y eliminando la línea de configuración existente.

Anote la dirección IP (probablemente la 192.168…) para la red host-only. En dependencia del sistema operativo que esté usando regístrese. En caso de estar trabajando con Linux introduzca la contraseña para la imagen de su VM. Luego, trate de iniciar un terminal X usando:

$ xterm

Una nueva ventana de terminal debe aparecer. Si usted tiene éxito, así culmina con la instalación básica. Cierre el *xterm*. Si usted obtiene un "*xterm: DISPLAY is not set error*", verifique la instalación de su servidor X.

Windows

Para utilizar aplicaciones X11 tales como *xterm* y *wireshark*, el servidor Xming debe de estar corriendo, y debe de establecer una conexión ssh con el X11 *forwarding* habilitado.

1. Inicie Xming (ej. doble *click* en el ícono). Ninguna ventana debe de aparecer, pero si usted lo desea puede verificar que correal observar los procesos en el administrador de tareas de Windows.
2. Establezca una conexión ssh con el X11 *forwarding* habilitado.

Si usted inicio puTTY como una aplicación GUI, usted puede conectarse al introducir la dirección IP de su máquina virtual y habilitar el X11 *forwarding,* ver Figura A1.1 y A1.2.

Para habilitar el X11 *forwarding* de la GUI puTTY, *click* puTTY>*Connection*>SSH>X11, entonces *click* en *Forwarding*>*Enable* X11 *Forwarding*.

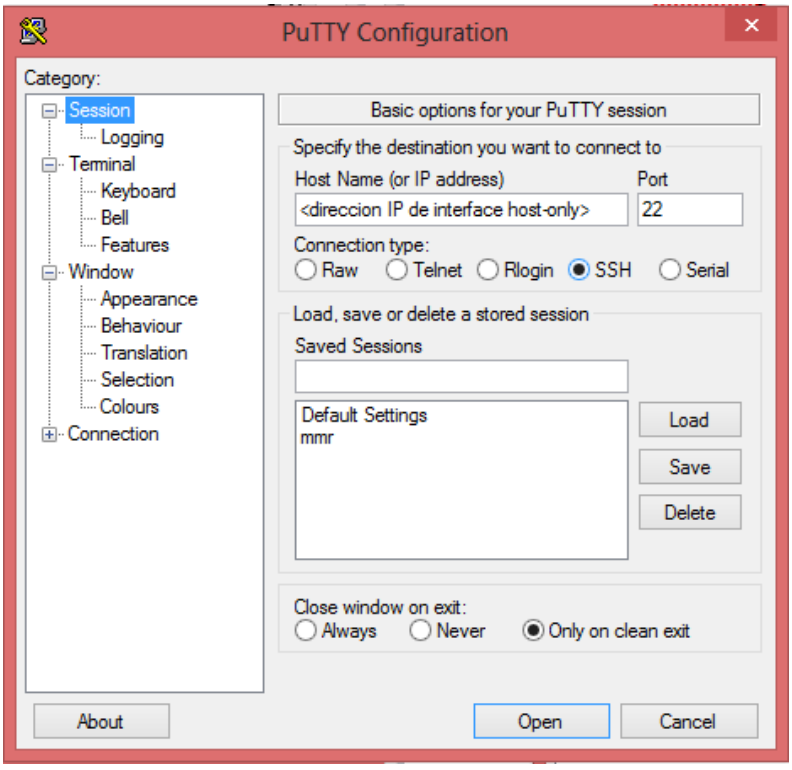

Figura A1.1 Configuración del PuTTY.

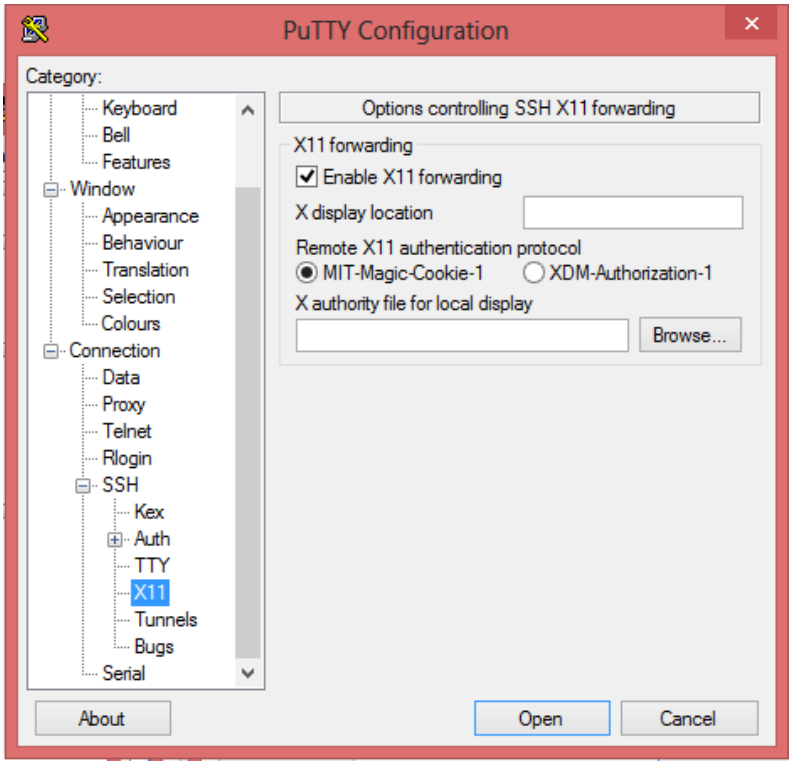

Figura A1.2 Habilitación del X11 forwarding.

Anexo II. Código de topología de un conmutador y cuatro terminales.

```
from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        Host1 = self.addHost( 'h1' )
        Host2 = self.addHost( 'h2' )
        Host3 = self.addHost( 'h3' )
        Host4 = self.addHost( 'h4' )
        Switch = self.addSwitch( 's1' )

        # Add links
        self.addLink( Host1, Switch )
        self.addLink( Host2, Switch )
        self.addLink( Host3, Switch )
        self.addLink( Host4, Switch )

topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Anexo III. Creación de topología de un conmutador y cuatro terminales.

Se posiciona en la carpeta que contiene el *script* de programación de creación de topologías, en la máquina virtual que se utiliza para este trabajo, al iniciar el Mininet se ejecuta el comando de Linux:

$ cd mininet/custom

Dentro de la carpeta *custom* para levantar la topología se utiliza el siguiente comando, donde se especifican algunas particularidades de la red que se desea montar:

$sudo mn --custom VLAN.py --topo mytopo --switch ovsk --control remote

En este caso, se especificó el archivo que constituye la topología personalizada (VLAN.py), se estableció la biblioteca que se utilizó para activar las funciones de la topología (mytopo), que el conmutador a usar en la simulación sea un conmutador del tipo OpenFlow (ovsk), y se estableció que la red funcione con un controlador remoto, en este caso como no se especifica ni IP, ni puerto para el controlador el comando activa el controlador remoto desde la propia máquina virtual (IP:127.0.0.1) en la que se trabaja.

Anexo IV. Implementación de VLAN en la red de un conmutador y cuatro terminales.

Para el establecimiento de VLAN se procede a eliminar el controlador del conmutador S1 desde el *xterm* de S1 con el comando:

 # ovs-vsctl del-controller s1

Desde el *xterm* se cambia el modo de trabajo, de modo seguro a modo estándar:

 # ovs-vsctl set-fail-mode s1 standalone

Y se establecen los puertos de cada VLAN:

 # ovs-vsctl set port s1-eth1 tag=10
 # ovs-vsctl set port s1-eth2 tag=20
 # ovs-vsctl set port s1-eth3 tag=10
 # ovs-vsctl set port s1-eth4 tag=20

Anexo V. Código de topología de dos conmutadores y cuatro terminales.

```
from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        Host1 = self.addHost( 'h1' )
        Host2 = self.addHost( 'h2' )
        Host3 = self.addHost( 'h3' )
        Host4 = self.addHost( 'h4' )
        Switch = self.addSwitch( 's1' )

        # Add links
        self.addLink( Host1, Switch )
        self.addLink( Host2, Switch )
        self.addLink( Host3, Switch )
        self.addLink( Host4, Switch )

topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Anexo VI. Creación de topología de dos conmutadores y cuatro terminales.

Se posiciona en la carpeta que contiene el *script* de programación de creación de topologías, en la máquina virtual que se utiliza para este trabajo, al iniciar el Mininet se ejecuta el comando de Linux:

$ cd mininet/custom

Dentro de la carpeta *custom* para levantar la topología se utiliza el siguiente comando, donde se especifican algunas particularidades de la red que se desea montar:

$sudo mn --custom enlace_troncal.py --topo mytopo --switch ovsk --control remote

En este caso, se especificó el archivo que constituye la topología personalizada (enlace_troncal.py), se estableció la biblioteca que se utilizó para activar las funciones de la topología (mytopo), que los conmutadores a usar en la simulación sean conmutadores del tipo OpenFlow (ovsk), y se estableció que la red funcione con un controlador remoto, en este caso como no se especifica ni IP, ni puerto para el controlador el comando activa el controlador remoto desde la propia máquina virtual (IP:127.0.0.1) en la que se trabaja.

Anexo VII. Implementación de VLAN en la red de dos conmutadores y cuatro terminales.

Para el establecimiento de VLAN se procede a eliminar el controlador del conmutador s1 y s2 desde el *xterm* de s1 con el comando:

 # ovs-vsctl del-controller s1

 # ovs-vsctl del-controller s2

Desde el *xterm* se cambia el modo de trabajo, de modo seguro a modo estándar:

 # ovs-vsctl set-fail-mode s1 standalone

 # ovs-vsctl set-fail-mode s2 standalone

Y se establecen los puertos pertenecientes a cada VLAN:

 # ovs-vsctl set port s1-eth1 tag=10

 # ovs-vsctl set port s1-eth2 tag=20

 # ovs-vsctl set port s2-eth1 tag=10

 # ovs-vsctl set port s2-eth2 tag=20

Anexo VIII. Código del *script* de programación que describe al componente l2_learning.py.

```
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.addresses import EthAddr
from pox.lib.util import dpid_to_str
from pox.lib.util import str_to_bool
import time

log = core.getLogger()

# We don't want to flood immediately when a switch connects.
# Can be overriden on commandline.
_flood_delay = 0

class LearningSwitch (object):
def __init__ (self, connection, transparent):
   # Switch we'll be adding L2 learning switch capabilities to
   self.connection = connection
   self.transparent = transparent

   # Our table
   self.macToPort = {}

   # We want to hear PacketIn messages, so we listen
   # to the connection
connection.addListeners(self)

   # We just use this to know when to log a helpful message
   self.hold_down_expired = _flood_delay == 0

   #log.debug("Initializing LearningSwitch, transparent=%s",
   #          str(self.transparent))

def _handle_PacketIn  (self, event):
   """
   Handle packet in messages from the switch to implement above algorithm.
   """
packet = event.parsed
```

```
def flood (message = None):
    """ Floods the packet """

msg = of.ofp_packet_out()

if time.time() - self.connection.connect_time >= _flood_delay:
      # Only flood if we've been connected for a little while...

if self.hold_down_expired is False:
       # Oh yes it is!
       self.hold_down_expired = True
log.info("%s: Flood hold-down expired -- flooding",
           dpid_to_str(event.dpid))

if message is not None: log.debug(message)
      #log.debug("%i: flood %s -> %s", event.dpid,packet.src,packet.dst)
      # OFPP_FLOOD is optional; on some switches you may need to change
      # this to OFPP_ALL.
msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
else:
pass
      #log.info("Holding down flood for %s", dpid_to_str(event.dpid))
    msg.data = event.ofp
    msg.in_port = event.port
self.connection.send(msg)

def drop (duration = None):
    """
    Drops this packet and optionally installs a flow to continue
dropping similar ones for a while
"""
if duration is not None:
if not isinstance(duration, tuple):
duration = (duration,duration)
msg = of.ofp_flow_mod()
      msg.match = of.ofp_match.from_packet(packet)
      msg.idle_timeout = duration[0]
      msg.hard_timeout = duration[1]
      msg.buffer_id = event.ofp.buffer_id
self.connection.send(msg)
elif event.ofp.buffer_id is not None:
```

```
msg = of.ofp_packet_out()
        msg.buffer_id = event.ofp.buffer_id
        msg.in_port = event.port
self.connection.send(msg)

self.macToPort[packet.src] = event.port # 1

if not self.transparent: # 2
if packet.type == packet.LLDP_TYPE or packet.dst.isBridgeFiltered():
drop() # 2a
return

if packet.dst.is_multicast:
flood() # 3a
else:
if packet.dst not in self.macToPort: # 4
flood("Port for %s unknown -- flooding" % (packet.dst,)) # 4a
else:
port = self.macToPort[packet.dst]
if port == event.port: # 5
         # 5a
log.warning("Same port for packet from %s -> %s on %s.%s.  Drop."
            % (packet.src, packet.dst, dpid_to_str(event.dpid), port))
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):
  """
  Waits for OpenFlow switches to connect and makes them learning switches.
  """
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):
  """
  Starts an L2 learning switch.
  """
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))
```

Anexo IX. Implementación de un ACL basado en direcciones MAC.

Se enciende la máquina virtual que contiene el paquete del Mininet instalado y se crea al menos dos sesiones SSH, en una se trabaja con la red y en la otra con el controlador, se debe iniciar el Xming, para acceder a los *xterm* de los dispositivos que aparecen en las redes creadas. Para la realización de esta tarea se utiliza el *script* de programación del controlador *l2_learning.py* que se encuentra en el directorio *~/pox/pox/forwarding*, para acceder a este directorio en una de las sesiones SSH se utiliza el comando:

cd pox/pox/forwarding

Para visualizar todos los *script* de programación que se encuentran en este directorio se utiliza el comando:

ls

Este comando lista todos los archivos y carpetas contenidas en el directorio *forwarding*. Una vez que se tenga visión de todos los archivos, se abre el *script* de programación para modificar el mismo, se necesitan permisos de *root* por lo que se especifica en el comando de ingreso:

sudo nano l2_learning.py

En la otra sesión SSH se accede al directorio donde se encuentra el *script* de la topología mediante el comando:

cd mininet/custom

Una vez en el directorio utilizar el siguiente comando para crear la red:

sudo mn --custom VLAN.py --topo mytopo --switch ovsk --control remote --mac

En el comando la palabra *mac* asegura el último número de las direcciones MAC de los terminales coincidan con el último número de las direcciones IP, recordar que todas las direcciones IP usadas hasta el momento pertenecen a la red 10.0.0.0.

Se crea el diccionario, que es un arreglo que relaciona dos valores, donde se incluyen las direcciones MAC que pertenecen a la lista de control de acceso, la sintaxis del diccionario se muestra a continuación:

mac = {}
mac[EthAddr(“000000000001”) = 1
mac[EthAddr(“000000000002”) = 2

Al definir las direcciones que pertenecen al ACL, estas serán las únicas direcciones que se autorizan a utilizar la red, para crear la lógica que limita el tráfico se coloca con la prioridad suficiente para definir quien se comunica y a la vez no entorpecer con el funcionamiento del dispositivo. Como se observa en la Figura A9.2, las líneas de código se ingresan luego de la invocación de las librerías necesarias para el correcto funcionamiento del componente y antes de la creación de la clase *LearningSwitch*, clase donde se definen todas las funciones y el comportamiento del dispositivo.

Figura A9.2 Diccionario que lista las direcciones MAC del ACL.

En la función *_handle_PacketIn*, que se encarga de analizar los mensajes *packet_in* que llegan al controlador, se escribe el código para crear entradas de flujo que eliminen a los paquetes que provienen de las direcciones ajenas al ACL, la Figura A9.3 muestra el código escrito:

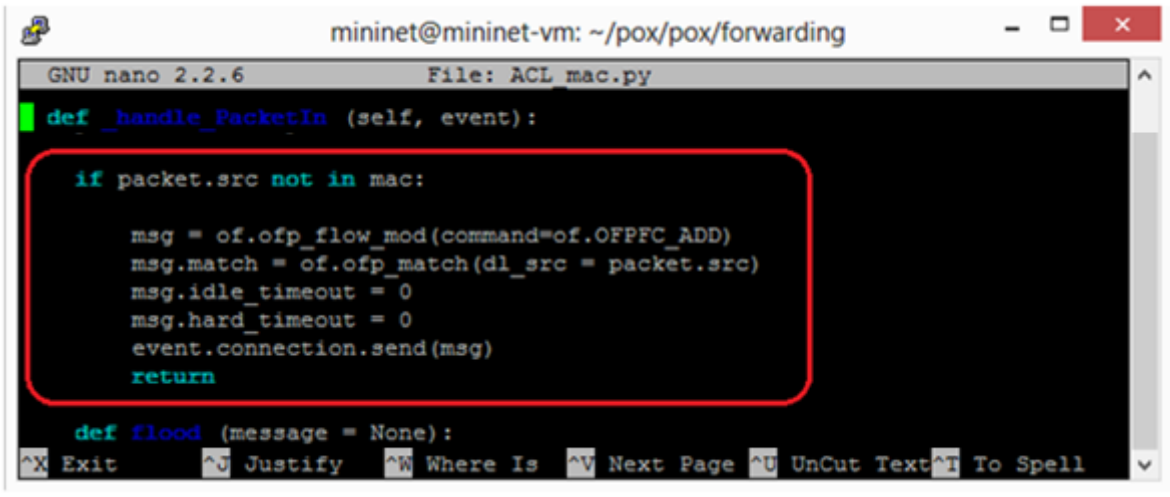

Figura A9.3 Código empleado.

En la imagen anterior la primera línea del código hace referencia a las direcciones que no se especifican en el diccionario ***mac***, donde se encuentra el ACL, entonces se crea un mensaje de modificación de flujo, se especifica que la regla de coincidencia de esta nueva entrada de flujo es la dirección de origen de la capa de enlace de datos (*dl_src, data link source y packet.src*), se especifican los tiempos de *idle_tiemout* y *hard_timeout* de manera que se crean entradas de flujos permanentes para las direcciones no autorizadas, no se especifica la acción a seguir, al hacer esto se indica al conmutador que deseche los paquetes que provienen de estas direcciones, se envía el mensaje al conmutador para instalar la nueva entrada de flujo que define la regla de reenvío y por último en el código se retorna, ya que el tratamiento a estos paquetes finalizó.

Una vez que se inicie la red con los cambios guardados en el *script* de programación, se prueba el componente, para esto en la sesión SSH donde se trabaja con el *script* de programación, se coloca en el directorio *pox* y se introduce el comando:

./pox.py forwarding.l2_learning

Este comando compila el componente en el directorio que se especifique, en este caso en el directorio *forwarding* y el archivo *l2_learning.py* o el nombre del archivo con el que se guardó luego de los cambios. Al compilar el programa sin ningún error, en la Figura A9.4 se muestra el cartel que aparece en la sesión SSH:

Figura A9.4 Cartel de notificación de correcto funcionamiento del controlador.

En la imagen se observa que el controlador está activo y que estableció conexión con un dispositivo, en este caso, con el conmutador de la topología previamente iniciada.

Anexo X. Implementación de un cortafuego basado en direcciones IP.

Se enciende la máquina virtual que contiene el paquete del Mininet instalado y se crea al menos dos sesiones SSH, en una se trabaja con la red y en la otra con el controlador, se debe iniciar el Xming, para acceder a los *xterm* de los dispositivos que aparecen en la red creada. Para la realización de esta tarea se utiliza el *script* de programación del controlador *l2_learning.py* que se encuentra en el directorio *~/pox/pox/forwarding*, para acceder a este directorio en una de las sesiones SSH se utiliza el comando:

cd pox/pox/forwarding

Para visualizar todos los *script* de programación que se encuentran en este directorio se utiliza el comando:

ls

Este comando lista todos los archivos y carpetas contenidas en el directorio *forwarding*. Una vez que se tenga visión de todos los archivos, se abre el *script* de programación para modificar el mismo, se necesitan permisos de *root* por lo que se especifica en el comando de ingreso:

sudo nano l2_learning.py

En la otra sesión SSH se accede al directorio donde se encuentra el *script* de la topología mediante el comando:

cd mininet/custom

Una vez en el directorio utilizar el siguiente comando para crear la red:

sudo mn --custom VLAN.py --topo mytopo --switch ovsk --control remote --mac

Para obtener el cortafuego se crea un arreglo o un diccionario, que es un tipo de arreglo que relaciona dos valores, dentro del *script* de programación del componente *l2_learning.py* sobre el cual se llevan a cabo las modificaciones, en el cual se incluyen las direcciones IP autorizadas, la sintaxis del código se muestra a continuación:

lista = {}
lista[IPAddr(“10.0.0.1”) = 1
lista[IPAddr(“10.0.0.2”) = 2

En este ejemplo de los 4 terminales de red, dos se incluyen en el cortafuegos, el 1 y 2.

Como se observa en la Figura A10.2, las líneas de código se ingresan luego de la invocación de las librerías, necesarias para el correcto funcionamiento del componente, y antes de la creación de la clase *LearningSwitch()*, donde se definen todas las funciones y el comportamiento del dispositivo.

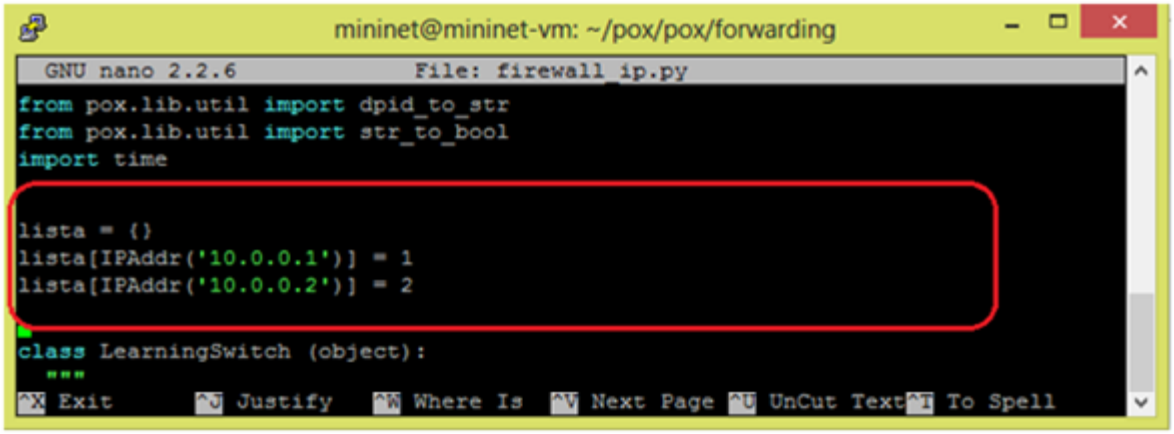

Figura A10.2 Diccionario que lista las direcciones IP del cortafuego.

En la Figura A10.3 se observa que se creó una función llamada *firewall()* dentro de la función *_handle_PacketIn(),*que al invocarse crea un mensaje de modificación de flujo, donde se especifica que la regla de coincidencia en esta nueva entrada de flujo se definen por los valores de la cabecera del propio paquete que provoca la invocación, se especifican los tiempos de *idle_tiemout* y *hard_timeout* de manera que se crean entradas de flujos permanentes para las direcciones no autorizadas, no se especifica la acción a seguir, al hacer esto se indica al conmutador que debe desechar los paquetes que provienen de estas direcciones y se envía el mensaje al conmutador para instalar la nueva entrada de flujo que define la regla de reenvío.

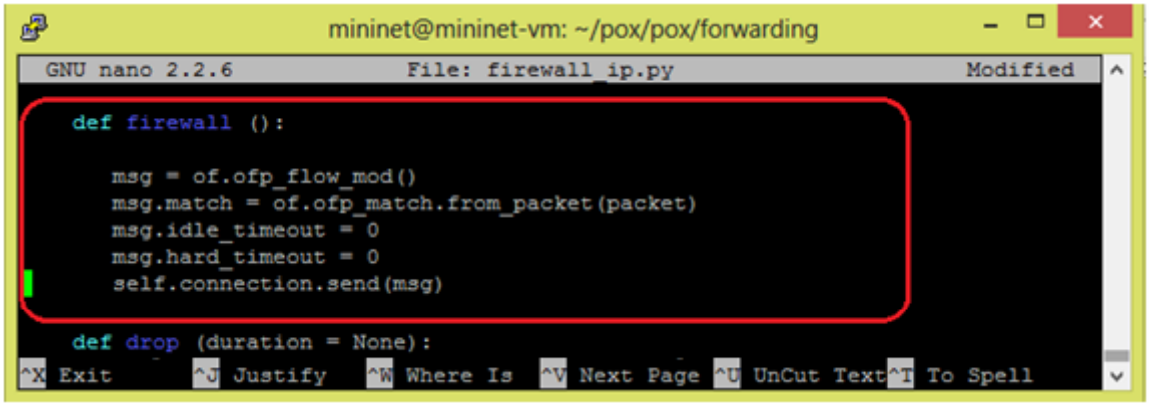

Figura A10.3 Función firewall().

El código que permite la invocación de la función *firewall()*se escribe luego de la declaración de las funciones dentro de la función *_handle_PacketIn(),*entre las líneas que describen el tratamiento que se le da al mensaje *packet_in*, ver Figura A10.4

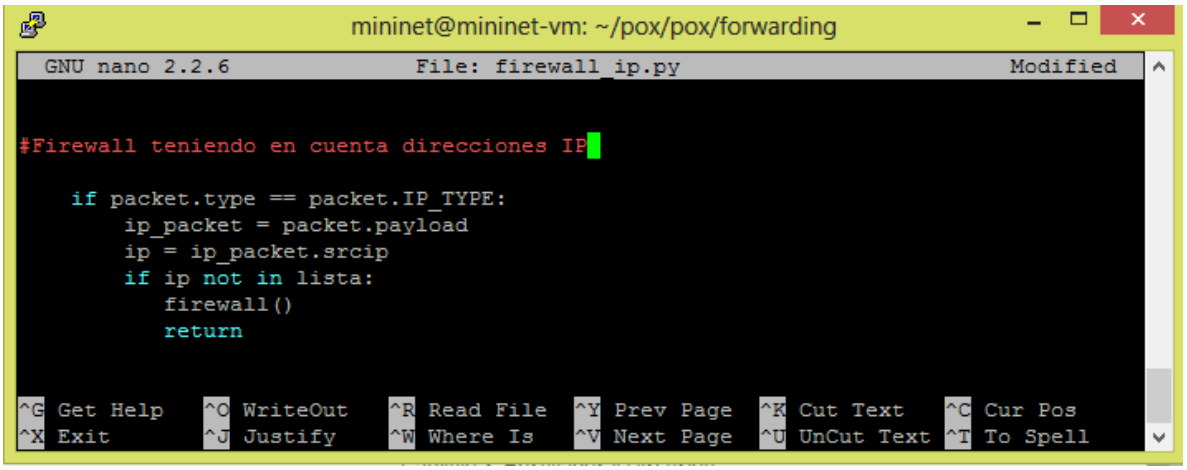

Figura A10.4 Código de invocación de la función firewall().

Este código luego de la llegada del mensaje *packet_in* y de parsear el paquete, busca si dentro del mismo se encuentra algún paquete IP, en caso de ser afirmativo se pasa a guardar el paquete IP en *ip_packet* y se le asigna a la variable local *ip* el valor de la dirección IP de origen del paquete, se analiza si la dirección se encuentra en la lista y en caso de ser negativo se invoca a la función *firewall()* para darle el tratamiento adecuado a los paquetes de direcciones IP no autorizadas.

Una vez que se inicia la red y se salvan los cambios en el *script* de programación, se prueba el componente, para esto en la sesión SSH donde se trabaja con el *script* de programación se posiciona en el directorio *pox* y se introduce el comando:

./pox.py forwarding.l2_learning

Este comando compila el componente en el directorio que se especifique, en este caso en el directorio *forwarding* y el archivo *l2_learning.py* o el nombre del archivo con el que se guardó luego de los cambios. Al compilar el programa sin ningún error, en la Figura A10.5 se muestra el cartel que aparece en la sesión SSH:

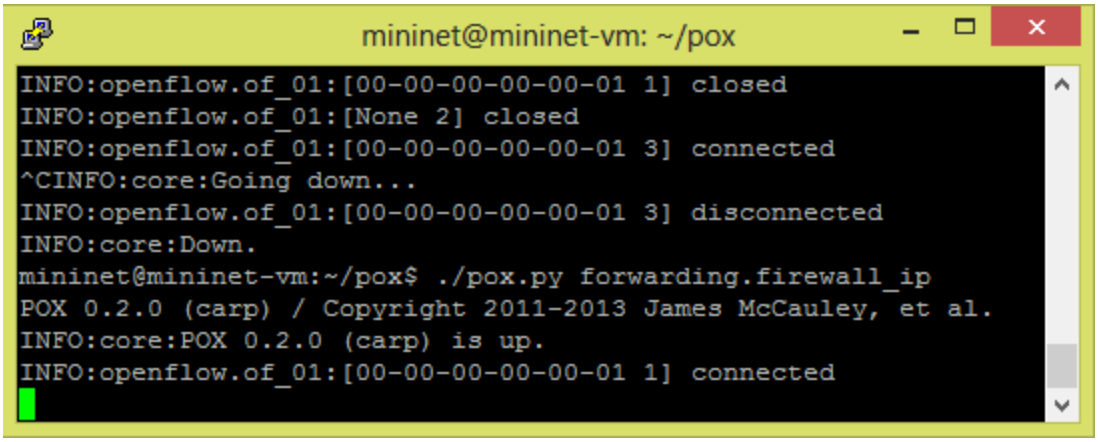

Figura A10.5 Cartel de notificación de correcto funcionamiento del controlador.

En la imagen se observa que el controlador está activo y que estableció conexión con un dispositivo, en este caso, con el conmutador de la topologia previamente iniciada.

Anexo XI. Implementación de QoS con una regla simple.

Para preparar el entorno de trabajo se enciende la máquina virtual que contiene el paquete del Mininet instalado y se crea al menos dos sesiones SSH, en una se trabaja con la red y en la otra con el controlador, además se debe iniciar el Xming, para acceder a los *xterm* de los dispositivos que aparecen en la red creada. Para la realización de esta tarea se utiliza el *script* de programación del controlador *l2_learning.py* que se encuentra en el directorio *~/pox/pox/forwarding*, para acceder a este directorio en una de las sesiones SSH se utiliza el comando:

cd pox/pox/forwarding

Para visualizar todos los script de programación que se encuentran en este directorio se utiliza el comando:

ls

Este comando lista todos los archivos y carpetas contenidas en el directorio *forwarding*. Una vez que se tenga visión de todos los archivos, se abre el *script* de programación para modificar el mismo, se necesitan permisos de *root* por lo que se especifica en el comando de ingreso:

sudo nano l2_learning.py

En la otra sesión SSH se accede al directorio donde se encuentra el *script* de la topología mediante el comando:

cd mininet/custom

Una vez en el directorio utilizar el siguiente comando para crear la red:

sudo mn --custom VLAN.py --topo mytopo --switch ovsk --control remote --mac

Desde el CLI del Mininet y previamente iniciado el Xming, se levantan los *xterm* de los dispositivos de red de la topología sobre los que se trabaja, mediante el comando:

xterm s1 h1 h3 h4

Al ingresar estos comandos en el CLI aparecen 4 ventanas *xterm*, cada una pertenece a un dispositivo de los que se especificó, en el *xterm* del terminal 3 y del terminal 4 se ingresan los siguientes comandos respectivamente:

iperf –s –p 5001 (en el xterm s3)
iperf –s –p 444 (en el xterm s4)

Estos comandos se utilizan para definir que los terminales trabajen como servidores para las pruebas de *iperf* a la que se somete la red (-s, servidor) y que los puertos de escucha son los especificados en el comando (-p XXX). Los *xterm* al ingresar estos comandos, lucen como muestran las Figuras A11.2 y A11.3:

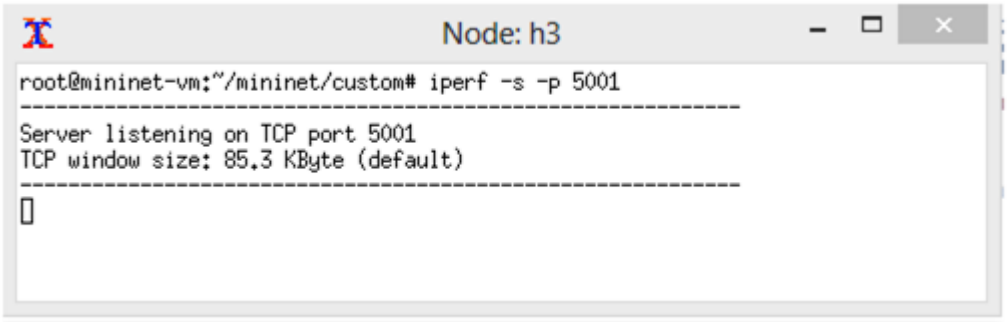

Figura A11.2 Creación del servidor en el terminal 3.

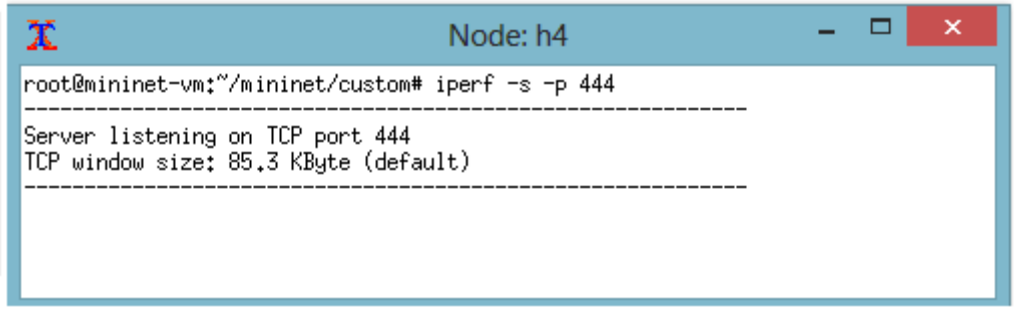

Figura A11.3 Creación del servidor en el terminal 4.

A continuación desde el *xterm* del conmutador s1 se instala una simple regla de *QoS* con una única cola (*Queue*, q0) en el puerto s1-eth1, al ingresar el comando que se muestra en la Figura A11.4:

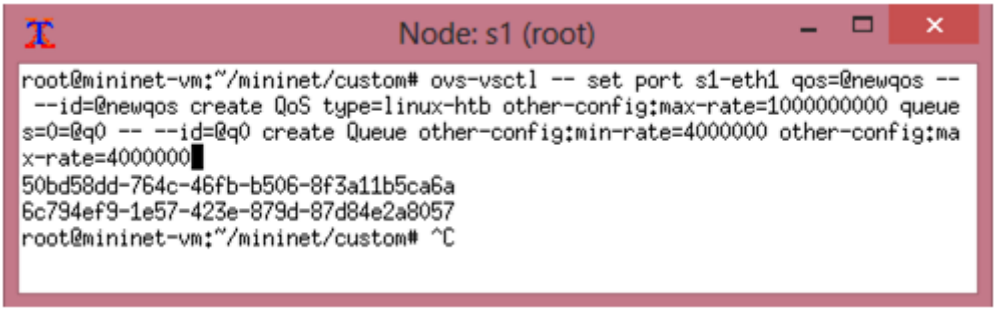

Figura A11.4 Creación de la regla de QoS en s1-eth1.

Se ingresa el comando *net* en el CLI del Mininet para conocer como está dispuesta la conexión entre los dispositivos de red, el resultado se muestra en la Figura A11.5:

Figura A11.5 Resultado del comando net.

Se resalta en recuadros rojos la interfaz del conmutador s1-eth1 que está conectado con h1-eth0, por eso se establece a la hora de realizar las pruebas que el cliente sea el terminal 1.

Anexo XII. Implementación de QoS con 2 colas.

En el trabajo con 2 colas, en una se coloca el tráfico que fluye sin restricciones (con una velocidad de transmisión máxima de 1Gbits/sec), y en la otra el tráfico que se regula a una velocidad de transmisión de 4Mbits/sec. El tráfico que se regula tiene como puerto de origen o puerto de destino el puerto 444, el restante tráfico fluye sin restricciones, el comando que se utiliza para crear las reglas de QoS, se ingresa en el *xterm* del conmutador s1, y aparece en la Figura A12.1:

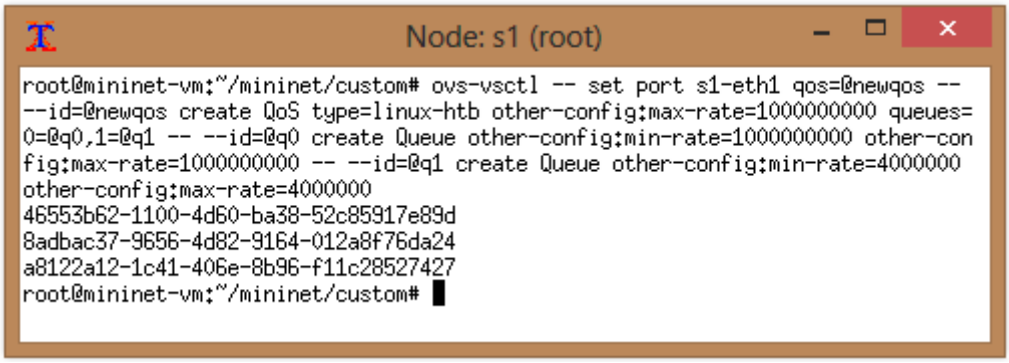

Figura A12.1 Creación de las reglas de QoS selectiva en s1-eth1.

Para regular el tráfico que fluye a través del puerto 444, en el *script* de programación del controlador, recordar que se utilizó el script del componente l2_learning.py, se añaden las líneas de código que se muestran en la Figura A12.2:

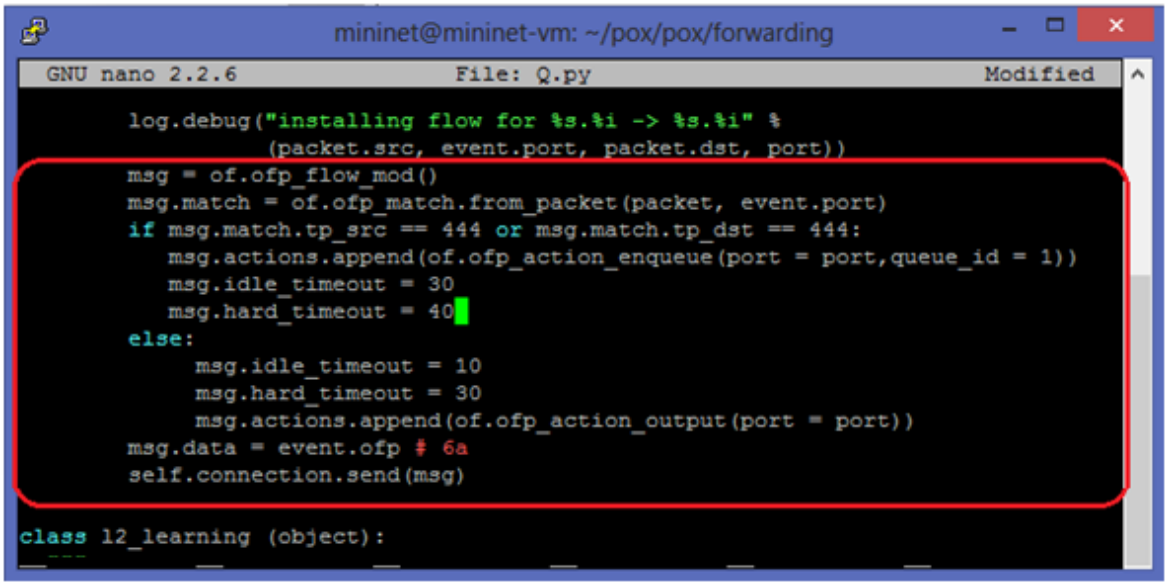

Figura A12.2 Código que discrimina el tráfico relacionado al puerto 444.

El código anterior verifica el puerto de origen y destino del tráfico, y en caso de utilizar el puerto 444 las entradas de flujos que se instalan especifican que el mismo se procesa por la cola 1 (*queue* 1=@q1) a la salida del puerto s1-eth1, donde se implementaron las reglas de *QoS* en la red.

Estas líneas se agregaron al final de la función _handle_PacketIn(),función que analiza cada mensaje *packet_in* que llega al controlador desde el conmutador como resultado de la llegada al conmutador de un paquete de un flujo que no aparece en las tablas de flujo, luego de la declaración de las funciones de manera que no impida el funcionamiento del componente y se logre la implementación de las reglas de *QoS* en la red. Aunque no se especifique en el código el tráfico restante se regula teniendo en cuenta la cola 0, que se toma por defecto al ser creada.