Actualización del componente práctico del curso de redes programables del programa de Ingeniería de Telecomunicaciones

Cristian Alexis Díaz Rodríguez

Informe de Trabajo de grado como requisito para optar al título de Ingeniero en Telecomunicaciones

Tutor

Jaime Alberto Vergara Tejada

Docente

Universidad de Antioquia

Facultad de Ingeniería

ingeniería de Telecomunicaciones

Medellín, Antioquia, Colombia

2024

I. Resumen.

Las redes programables o redes definidas por software (SDN) ofrecen una solución innovadora para abordar los desafíos de las redes actuales. Con SDN, es posible centralizar y simplificar la gestión de la red, lo que facilita la configuración y el monitoreo en tiempo real. Además, SDN permite una mayor flexibilidad y agilidad al separar el plano de control del plano de datos, lo que significa que las políticas de red pueden implementarse de manera más rápida y eficiente. Esta arquitectura también facilita la automatización de tareas repetitivas, mejorando la eficiencia operativa y permitiendo una adaptación más rápida a los cambios en las demandas de la red.

Lo anterior motiva a realizar este trabajo de grado para actualizar el componente práctico del curso de redes programables. Para esto se investiga el estado del arte de las redes programables y se escogen dos temas relevantes de los cuales se realizarán dos prácticas de laboratorio. Luego de diseñar estas prácticas, con ayuda de un software de emulación de redes virtuales llamado Mininet y el controlador POX se realizan los códigos que satisfacen las condiciones planteadas en el diseño de las practicas.

Como entregables se tendrán: Guías de laboratorio que orientarán al estudiante para desarrollar las practicas, códigos de solución a las practicas, debidamente comentados y ampliamente explicados en este informe. Se entregarán todos los recursos que generan las topologías de los escenarios planteados y que a su vez servirán como entornos para realizar las pruebas y, finalmente, las plantillas que usara el estudiante como base para realizar la práctica de laboratorio.

II. Introducción.

Debido a la llegada de los datacenters, la computación en la nube, y los dispositivos IoT, la infraestructura de TI y sus procesos de mantenimiento, están cambiando en las organizaciones [6]. La forma tradicional de configurar los conmutadores y demás equipos de red por lo general es costosa, propensa a errores, consume mucho tiempo y esfuerzo llegando a retrasar las operaciones de las empresas [3]. Las redes programables (SDN por sus siglas en inglés) es un paradigma que, a partir de el desacoplamiento del plano de control y de datos facilita la actualización e integración de nuevas funcionalidades en las redes de datos. Es por esto que la academia y sector productivo participan activamente en la elaboración de estándares y directrices sobre la adopción de las SDN. [8]. Se entiende así entonces la importancia de incluir la enseñanza de SDN en un programa de Ingeniería de Telecomunicaciones.

Lo anterior, motiva a el propósito de este trabajo de grado que es el de aportar a la creación del curso de Redes programables para el programa de Ingeniería de Telecomunicaciones. El aporte se hará con la elaboración de dos practicas complementarias al componente ya existente de laboratorios del curso. Para el diseño de estas dos prácticas, primero se debe realizar una investigación de temas relevantes en el contexto de las redes programables, y de allí seleccionar los temas más atractivos, teniendo como criterio de selección el hecho de que a partir de estos se deberán plantear escenarios prácticos con desafíos en un entorno de laboratorio. Dicho entorno de laboratorio es un software llamado Mininet alojado en una máquina virtual, que permite emular redes y programarlas. El siguiente paso implica la realización de los códigos correspondientes a las plantillas de los escenarios propuestos, junto con el desarrollo de sus soluciones para abordarlos de manera efectiva. El último paso es el de desarrollar las guías de laboratorio con su solución. Esta documentación ayudara a complementar el curso de Redes programables siendo este un aporte actual y relevante en el ámbito de las telecomunicaciones.

Este Informe está dividido de la siguiente manera: la siguiente sección **III** comprende el objetivo general y específicos de este trabajo de grado. En la sección del marco teórico (**IV**) se realiza un compendio sobre el estado del arte de las redes programables, incluyendo conceptos importantes como los controladores POX proactivo y reactivo, los contenedores y los softwares de simulación para redes programables, que se usarán para el desarrollo de este trabajo de grado. Luego, la sección **V** describe detalladamente las actividades realizadas para el diseño de las prácticas. Esto incluye la investigación, la selección de los temas, el diseño de topologías y escenarios, la creación de códigos para resolver las problemáticas planteadas, así como el diseño de guías y plantillas que se proporcionarán al estudiante que tome el curso. La sección **VI**, describe las conclusiones halladas al

terminar de realizar este trabajo de grado y finalmente se anexan los entregables importantes en la sección **VII**, estos entregables son las guías, códigos solución y códigos plantillas para el estudiante. todos debidamente comentados y con su explicación mas a detalle en la sección **V**.

III. Objetivos.

Objetivo General.

- Actualizar el componente práctico del curso de redes programables a través de la elaboración de nuevas prácticas de laboratorio.

Objetivos Específicos.

- Seleccionar temas relevantes para ser incluidos como prácticas de laboratorio en el curso.
- Implementar las prácticas correspondientes a los temas seleccionados en un entorno de laboratorio definido.
- Evaluar la implementación de las practicas a través de un protocolo de pruebas definida.

IV. Marco Teórico.

Para entender el aporte que hacen las redes definidas por software (SDN por sus siglas en Ingles) démosle una rápida mirada al panorama de cómo funcionan los equipos de redes actuales y sus limitaciones.

En la infraestructura convencional, la implementación, configuración, actualización o resolución de problemas de red se deben realizar por personal técnicamente capacitado de alto nivel y para llevar a cabo dichas labores estos deben intervenir uno por uno [1]. Esto toma mucho tiempo, y la variedad y complejidad [2] de los elementos de la red hacen que su mantenimiento sea muy costoso y que la infraestructura subyacente sea menos confiable en caso de fallas frecuentes de la red.

Por otro lado, las redes tradicionales tienen limitaciones que plantean desafíos a medida que aumenta la cantidad de dispositivos de red. Los switches, enrutadores y otros componentes administrados distribuidos por los diversos fabricantes son netamente estáticos. Un diseño de este tipo es muy inflexible e inadecuado para satisfacer las demandas de las aplicaciones actuales de tráfico complejo y ancho de banda programable [3].

No obstante, el paradigma recientemente surgido de las redes definidas por software da un atisbo de esperanza para una nueva arquitectura de red que proporcione más flexibilidad y adaptabilidad.

Redes definidas por software (SDN)

SDN Brinda nuevas perspectivas de cómo se administran las redes y se está convirtiendo rápidamente en la solución ideal para quienes tienen problemas con las limitaciones de las redes tradicionales [4]. SDN define una arquitectura que permite una gestión de la red estrictamente basada en software. Lo anterior implica desacoplar el hardware del software, es decir, separar el plano de control (que determina dónde enviar el tráfico) del plano de datos (que lleva a cabo estas decisiones y reenvía el tráfico). De esta manera es posible controlar y gestionar el hardware desde una aplicación de software centralizada y separada del propio hardware. Esta implementación brinda la capacidad de manejar toda la red desde una sola unidad lógicamente centralizada, llamada punto central, permitiendo una configuración y administración dinámica de los dispositivos de la red, reduciendo así tiempos y costos gracias a una gestión administrativa simplificada que también incrementa la seguridad en la red [4].

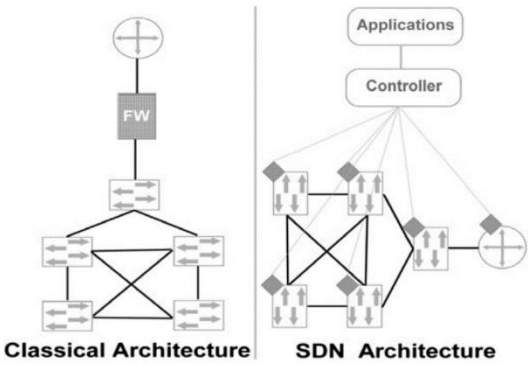

Figura 1. Arquitectura clásica vs SDN [2]

Los componentes principales de la arquitectura SDN son [5]:

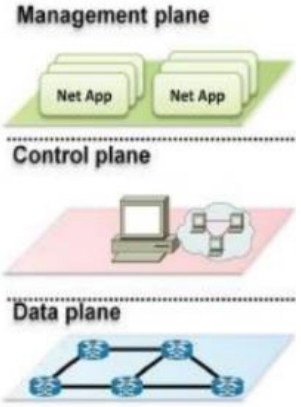

Figura 2. Componentes SDN [5]

- **Plano de datos:** Consisten en elementos de red, como dispositivos físicos y virtuales, que se ocupan del tráfico de datos. Se le conoce como plano de reenvío de SDN y es físicamente responsable de reenviar tramas de paquetes desde su interfaz de entrada a salida utilizando los protocolos utilizados por el plano de control [4].

- **Plano de control:** Esta es una de las áreas cruciales en cualquier arquitectura SDN. El plano de control es donde tiene lugar la acción real. Los controladores con capacidad lógica se encuentran en este plano. Este es considerado como el "Cerebro de Red". Esta es la parte más importante donde toda la lógica descansa y se reenvía a toda la red. La escalabilidad de la red depende de qué tan fuerte sea el controlador. La capacidad de supervivencia de la red se basa en el hecho de que el controlador debería poder soportar los ataques de la red y resistirlos.

- **Plano de gestión:** El plano de gestión incluye los servicios de software, como las herramientas basadas en SNMP [5], que se utilizan para

monitorear y configurar de forma remota la funcionalidad de control. Estos servicios incluyen funciones como aplicaciones de firewall; equilibrio de carga, enrutamiento y monitoreo de los datos en la red [6].

- SDN, usa interfaces northbound y southbound para la creación de aplicaciones y administración de los dispositivos de red. De esta manera, las instrucciones al controlador SDN se envían a través de la interfaz northbound para luego, el controlador SDN enviar las configuraciones específicas a los equipos de red a través de la interfaz southbound.

Mininet

Mininet es un software de emulación para redes definidas por software (SDN), utiliza la API de Python para la creación de redes y es muy adecuada para diseñar y simular una SDN para su evaluacion y estudio. Es un emulador de red que crea una red de hosts, conmutadores, controladores y enlaces virtuales, los hosts ejecutan software de red Linux estándar y sus conmutadores admiten el protocolo OpenFlow para enrutamiento personalizado altamente flexible y SDN. Mininet apoya la investigación, el desarrollo, el aprendizaje, la creación de prototipos, las pruebas, la depuración y cualquier otra tarea que podría beneficiarse de tener una red experimental completa en una computadora portátil o un PC, proporciona un banco de pruebas virtual y un entorno de desarrollo para redes definidas por software (SDN) [7].

Mininet incluye la instalación del controlador POX, que puede utilizarse como un controlador SDN básico mediante el uso de los componentes que la herramienta trae incluidos. El controlador POX permite implementar aplicaciones que hacen uso del protocolo OpenFlow, que es el protocolo de comunicación de facto entre los controladores y los conmutadores. Con el controlador POX se pueden ejecutar diferentes aplicaciones de conmutación, enrutamiento, balanceo de carga y seguridad [9].

Openflow

OpenFlow está definido por la Open Networking Foundation (ONF) como la primera interfaz estándar de comunicaciones entre la capa de control y la de infraestructura de una arquitectura SDN. Proporciona un medio para controlar un conmutador sin necesidad de que los proveedores revelen ningún código fuente de sus dispositivos. En otras palabras, OpenFlow proporciona acceso a la tabla de flujo de los conmutadores y les indica cómo dirigir el tráfico de la

red, permitiendo así a los administradores de la red poder cambiar los flujos de la red en corto tiempo [10].

POX

POX se define como un controlador SDN de código abierto basado en Python, principalmente utilizado para el desarrollo rápido y prototipado de nuevas aplicaciones de red. Básicamente, viene preinstalado en la máquina virtual de Mininet. El controlador POX puede convertir dispositivos OpenFlow simples en dispositivos de hub, switch, balanceador de carga y firewall [11].

El controlador POX presenta dos enfoques de comportamiento: reactivo y proactivo [12].

POX proactivo

En el enfoque proactivo, el controlador llena la tabla de flujos en cada switch con anterioridad. Este método no tiene tiempo adicional de configuración de flujos, ya que las reglas de reenvío están definidas previamente. Si un switch pierde la conexión con el controlador, no interrumpe el tráfico. Sin embargo, para operar la red de esta manera se requiere un manejo estricto, como la necesidad de agregar reglas (comodín) para cubrir todas las rutas.

POX reactivo

En el enfoque reactivo, cuando un switch recibe un paquete que no coincide con ningún flujo instalado, lo envía al controlador para que determine la acción sobre dicho paquete.. Este método utiliza de manera eficiente la memoria de la tabla de flujos existente, pero cada nuevo flujo conlleva un pequeño tiempo adicional de configuración. Sin embargo, este enfoque tiene una dependencia estricta del controlador, lo que significa que si un switch pierde la conexión, su utilidad se ve limitada.

Contenedores

Un contenedor, en el contexto de la informática, se refiere a un paquete de software liviano, independiente y ejecutable que contiene todo lo necesario para ejecutar una aplicación. Encapsula una aplicación y sus dependencias, incluidas bibliotecas, archivos de configuración, tiempo de ejecución y herramientas del sistema, en una sola unidad.

Las características clave de un contenedor incluyen [13]:

- **Aislamiento:** Los contenedores se ejecutan en espacios de usuario aislados, distintos del host y otros contenedores, lo que proporciona aislamiento a nivel de aplicación y al mismo tiempo comparte el mismo kernel del sistema operativo subyacente.

- **Portabilidad:** Los contenedores son altamente portátiles en diferentes entornos, como desarrollo, pruebas y producción, ya que empaquetan una aplicación y sus dependencias en un formato consistente.

- **Eficiencia:** Son eficientes en la utilización de recursos, ya que no requieren un sistema operativo separado, lo que permite que se ejecuten varios contenedores en un solo host sin dejar de estar aislados.

- **Consistencia:** Los contenedores garantizan la coherencia en diferentes entornos, lo que reduce el problema de "funciona en mi máquina" al garantizar que la aplicación se ejecute de manera consistente en varios entornos.

Los contenedores a menudo se crean, administran y organizan mediante plataformas de contenedorización como Docker, Podman o sistemas de orquestación de contenedores como Kubernetes. Estas plataformas facilitan la creación, implementación, escalamiento y administración de contenedores en clústeres de hosts, lo que permite una administración eficiente de aplicaciones complejas.

Es común encontrar que el Frontend y Backend de una aplicación estén desarrollados con tecnologías y lenguajes de programación diferentes, ya que dada su naturaleza exigirán diferentes tipos de componentes y tecnologías para realizar su respectiva función, ahora, para obtener una integración entre ambos frentes debe haber flujo de información constante entre ellos para obtener una aplicación operativa, igualmente, es posible que un desarrollo haga uso de recursos desarrollados por aplicaciones externas para su funcionamiento, esto es llamado consumo de Servicios Web o consumo de interfaces APIs.

Todo este flujo de información se coordina bajo el Protocolo de transferencia de hipertexto (HTTP, por sus siglas en inglés), un protocolo perteneciente a la capa 7, nivel de aplicación del modelo OSI, que por medio del uso de métodos establecidos por el propio protocolo, siendo los más utilizados GET, POST, DELETE y UPDATE permiten el envío y recepción de diferentes tipos de datos, bajo cabeceras que además de contener la información solicitada envían códigos de estados de respuesta de las peticiones para indicar el estado de estas, además de información utilizada por demás protocolos que intervienen

en los diferentes procesos de las capas del modelo OSI y además, del procesamiento de las solicitudes por los sistemas operativos y sus respectivos kernel para resolver las solicitudes.

V. Metodología

1. Seleccionar temas relevantes para ser incluidos como prácticas de laboratorio en el curso

1.1. Investigar temas actuales y relevantes en el área de las redes programables que se puedan abarcar dentro del curso de redes programables del programa Ingeniería de Telecomunicaciones:

En esta actividad, se llevó a cabo una revisión del estado del arte de las guías, códigos e información general disponibles en la web sobre escenarios en redes, y como desarrollarlos con ayuda de las redes programables, más específicamente haciendo uso de herramientas como Mininet y POX. Lo anterior haciendo uso de recursos como: Google Schoolar, GitHub y la propia wiki de POX.

1.2. Seleccionar temas que sean viables y se puedan desarrollar prácticas sobre estos, teniendo en cuenta complejidad y recursos disponibles:

Teniendo en cuenta la información hallada en la actividad anterior, se decidió escoger los siguientes temas: Balanceo de carga en enlaces y redes VLANs. Esta elección se fundamenta en los recursos identificados para cada tema, su relevancia en entornos reales de red, y en la posibilidad de desarrollar prácticas con un nivel de complejidad adecuado para un curso de redes programables.

1.3. Definir los escenarios de experimentación:

Una vez seleccionados los temas, se plantearon posibles topologías para llevar a cabo el desarrollo de las practicas. Con ayuda de Mininet es posible programar las topologías que se enseñaran a continuación:

Balanceo de carga por enlaces:

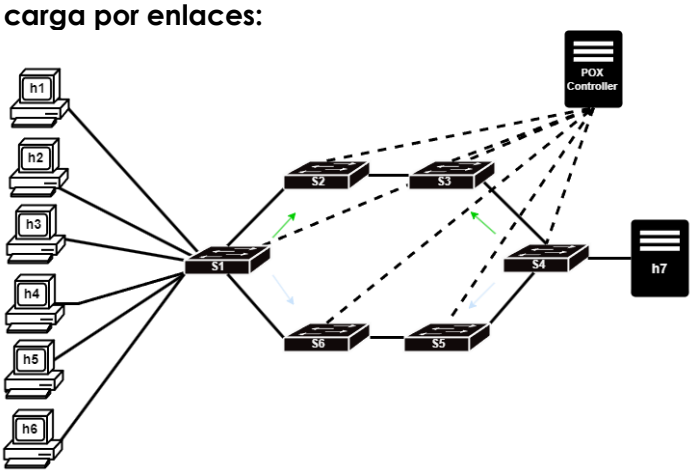

Figura 3: *Topología practica Balanceador de carga de enlaces.*

Esta topología consta de 6 hosts (h1-6) conectados al Switch 1 (S1), los cuales intentaran acceder al servidor h7. Los switches S1-6 forman una topología tipo anillo y están conectados al controlador POX, el cual les indicará como gestionar el tráfico de esta red para evitar bucles, sin necesidad de hacer uso de aplicaciones Spanning tree. De esta forma, garantizar conectividad entre todos los hosts haciendo uso eficiente de los recursos de red.

VLAN:

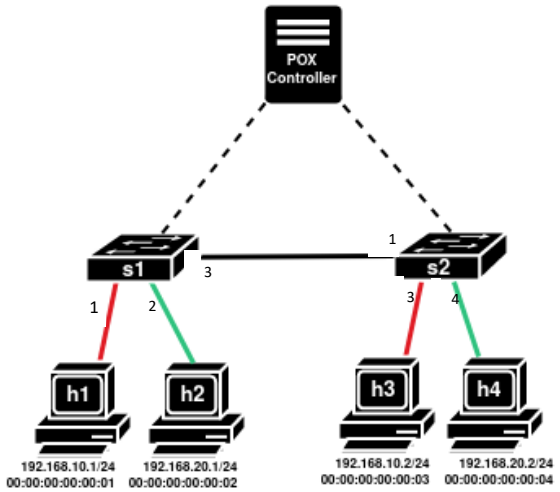

Figura 4: *Topología practica VLAN: Rojo VLAN A, Verde: VLAN B*

Se plantea un escenario en el que existen dos redes diferentes (roja con 192.168.10.0 y verde con red 192.168.20.0). Los switches S1 y S2 están conectados entre sí y también al controlador POX, el cual en modo reactivo o proactivo deberá indicarles a los switches como manejar este tráfico para cumplir las condiciones de dos redes VLAN. Esto es, que solo haya conexión exitosa entre h1 y h3 y h2 y h4.

2. Implementar las prácticas correspondientes a los temas seleccionados en un entorno de laboratorio definido.

Para el diseño y desarrollo de estas prácticas se implementó en el software de Virtual Box una máquina virtual de Ubuntu 22.04.3. En esta se instalo el software de emulación Mininet y un editor de código llamado Virtual Studio Code (VSC).

2.1. Diseñar las practicas:

Teniendo en cuenta los escenarios planteados anteriormente se proponen las siguientes practicas:

Balanceo de carga de enlaces: Se implementa una forma de balanceo de carga conocida como "Equal-Cost Multipath (ECMP)" routing. En este escenario, un enrutador tiene múltiples rutas hacia un destino y distribuye el tráfico de manera equitativa entre esas rutas. Esto se hace para optimizar la utilización de la red y mejorar el rendimiento general. Al enviar tráfico a través de múltiples rutas, el enrutador busca distribuir la carga de manera uniforme, evitando la congestión en una sola ruta. Esto ayuda a utilizar de manera más eficiente los recursos de red disponibles. Al tratarse

de una topología cerrada, para evitar la conocida tormenta de broadcast se propone gestionar este tráfico en sentido horario. Se usará la técnica de ECMP aplicando un filtro basado en un criterio, como por ejemplo la dirección IP de origen, para distribuir el tráfico IP de manera más equitativa entre las dos rutas disponibles.

Con lo anterior, y teniendo en cuenta la **Figura 3** se propone que el estudiante diseñe un código para un controlador POX reactivo que les indique a los switches filtrar los paquetes IP por IP origen. Esto es:

- Los paquetes con origen en h1, h2, o h3 y destino en h7, o con origen en h7 y destino en h1, h2, o h3, seguirán la ruta a través de los switches S1, S2, S3 y S4.

- Los paquetes con origen en h4, h5, o h6 y destino en h7, o con origen en h7 y destino en h4, h5, o h6, seguirán la ruta a través de los switches S1, S6, S5 y S4.

De esta manera se evita sobrecargar los enlaces y se optimiza el uso de los recursos disponibles.

El éxito del desarrollo de esta practica se demuestra con una operación Ping exitosa entre todos los hosts y enseñando los flujos en todos los switches, probando que se cumplen las condiciones de filtrado anteriores.

VLAN: Para esta práctica se le pide al estudiante que realice un código para un controlador POX proactivo que instale los flujos necesarios a los switches S1 y S2, para que gestionen el tráfico de esta red, teniendo en cuenta que los hosts h1 y h3 pertenecen a la red 192.168.10.0 y que h2 y h4 pertenecen a la red 192.168.20.0.

El éxito de esta práctica se evidencia mediante una operación de Ping exitosa exclusivamente entre los hosts pertenecientes a cada VLAN. Es decir, h1 tendrá una conexión exitosa solo con h3 (y viceversa), mientras que h2 tendrá una conexión exitosa solo con h4 (y viceversa). Además, se mostrarán los flujos de cada switch detallando cómo se asignaron las etiquetas a cada VLAN (VLAN ID) y cómo se gestionó el tráfico.

2.2. Realizar el montaje de las practicas sobre los escenarios planteados:

Con ayuda de VSC, los códigos y ejemplos hallados en la web, y en el directorio `/home/pox/pox` del software Mininet, se realizan los códigos para los controladores siguiendo las pautas y requisitos planteadas anteriormente. A continuación, se desglosa como se realizaron los códigos para ambas prácticas.

2.2.1. Balanceo de carga de enlaces:

Para la realización de este código es crucial el entendimiento de como funciona el controlador POX en modo reactivo. Como se explicó en la sección IV, en este modo, el switch inicialmente envía los paquetes que recibe hacia el controlador POX para que este con base en ciertos criterios instale los flujos adecuados.

Las librerías necesarias a importar son:

- **pox.core:** Importa el núcleo de POX, que es la base para la mayoría de las funcionalidades de POX.

- **pox.openflow.libopenflow_01:** Importa las definiciones específicas de OpenFlow 1.0, que se utilizan para comunicarse con switches OpenFlow.

- **pox.lib.util.dpidToStr:** Proporciona funciones para convertir el ID de un switch (DPID) a una cadena legible para humanos.

- **pox.lib.packet:** Este módulo de paquetes de POX contiene las definiciones de varios tipos de paquetes, como Ethernet, IPv4, etc., que son útiles para manipular y analizar paquetes de red.

- **pox.lib.addresses.EthAddr, pox.lib.addresses.IPAddr:** Estas clases se utilizan para representar direcciones Ethernet (MAC) y direcciones IP, respectivamente.

- **pox.lib.revent:** Contiene las clases de eventos de POX, como Event, EventHalt, etc., que se utilizan para definir y manejar eventos en el controlador.

- **pox.lib.util.dpid_to_str:** Otra utilidad para convertir DPID a una cadena, aunque con un nombre de función ligeramente diferente.

- **pox.lib.packet.ethernet:** La clase ethernet del módulo de paquetes de POX, que es utilizada para manejar paquetes Ethernet.

- **pox.lib.packet.ipv4:** La clase ipv4 del módulo de paquetes de POX, que es utilizada para manejar paquetes IPv4.

Todos los mensajes que recibe el controlador por parte de los switches usualmente se conoce como eventos. Por ello, un código para un controlador POX en modo reactivo suele tener una función principal que recibe estos eventos, por lo general se define así:

- `def _handle_PacketIn (event)`

A partir de estos eventos se puede extraer información útil como el puerto de entrada, o la dirección MAC del switch que envió esta información. Por ejemplo:

- `if(dpidToStr(event.dpid)=='00-00-00-00-00-01'):` Pregunta si la dirección MAC del switch que genero el evento en cuestión es `'00-00-00-00-00-01'`
- `if event.port == 1:` Pregunta el puerto de entrada del switch que genero el evento en cuestión es el puerto número 1.

A su vez, se puede también acceder al contenido mas importante de este evento, que no será más que una trama ethernet que genero algún host de la red.

- **`packet = event.parsed`**

De esta manera, se puede conocer información útil como: Dirección MAC origen y destino, tipo de tráfico, dirección IP origen y destino, VLAN ID, tipo de protocolo, entre otros. Sabiendo lo anterior, para el interés de esta práctica debemos saber cómo descartar el tráfico IPV6, distinguir el tipo de tráfico y conocer las direcciones IP origen y destino, para así filtrar el trafico IP. Lo anterior se logra con las siguientes líneas de código:

- **`if event.parsed.type != 34525:`** Descarta paquetes tipo 34525, o sea IPV6.
- **`if packet.type == 2054:`** Si el protocolo del paquete es ARP.
- **`if packet.type == 2048:`** Si el protocolo del paquete es IP.
- **`ds_ip = packet.next.dstip:`** Extrae la dirección IP destino del paquete capturado.
- **`src_ip = packet.next.srcip:`** Extrae la dirección IP destino del paquete capturado.

Lo siguiente a saber es como indicarle al switch como debe gestionar el tráfico ARP e IP. Esto se logra mediante la creación y envío de mensajes OpenFlow que instruyen a los switches sobre las acciones que deben tomar con los paquetes correspondientes.

Como se habló anteriormente, el tráfico ARP se pensó para que viaje de manera horaria, ósea, siguiendo un recorrido S1, S2, S3, S4, S5, S6. Así, garantizamos que se resuelvan exitosamente las solicitudes de este protocolo, crucial para la comunicación de los hosts.

En el caso de S1 y S4 nos interesa que se realice una acción de inundación (FLOOD), ya que los paquetes ARP deberán llegar a todos los hosts. Esta acción replica y conmuta el paquete ARP que ingresa al switch por todos los puertos excepto por el de entrada.

La **Figura 5** nos ayuda a tener una mejor idea de lo que hace la acción FLOOD en S1:

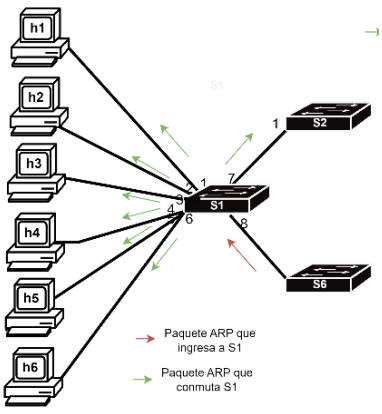

Figura 5: *Acción FLOOD en S1*

Nótese que según la **Figura 5**, cuando un paquete ARP ingresa por el puerto 8 a S1, con la acción FLOOD, este los replica por los demás puertos excepto por el puerto 8.

La **Figura 6** nos ayuda a tener una mejor idea de lo que hace la acción FLOOD en S4:

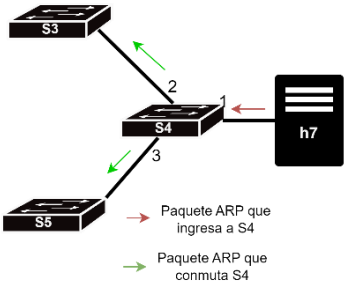

Figura 6: *Acción FLOOD en S4*

Mediante los mensajes Openflow mencionados anteriormente, se puede enviar esta orden al switch de la siguiente manera:

- **`msg = of.ofp_packet_out():`** Crear un mensaje OFP_PACKET_OUT
- **`msg.data = event.ofp:`** Establecer el contenido del mensaje como el paquete original recibido.
- **`msg.actions.append(of.ofp_action_output(port =of.OFPP_FLOOD)):`** Añadir una acción al mensaje: enviar el paquete a través de FLOOD (a todos los puertos excepto el de llegada)
- **`event.connection.send(msg):`** Enviar el mensaje al switch

Con la intención de que si se repite este tipo de tráfico en S1 (o en cualquier otro switch) este no se demore tanto en procesarlos, al tener siempre que preguntar al controlador que hacer con ellos, se puede instalar directamente el flujo, así:

- **`flow = of.ofp_flow_mod():`** Crear un mensaje OFP_FLOW_MOD para agregar una entrada de flujo al switch.
- **`flow.match = of.ofp_match(dl_type =0x0806):`** Definir la coincidencia (match) del flujo para paquetes ARP (0x0806)
- **`flow.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD)):`** Añadir una acción al flujo: enviar el paquete a través de FLOOD
- **`flow.hard_timeout = 100:`** Establecer un tiempo de vida (hard_timeout) para la entrada de flujo.
- **`event.connection.send(flow):`** Enviar el mensaje al switch

El anterior fragmento de código se sitúa dentro de un condicional que pregunta si el tipo de paquete es ARP (**`if packet.type == 2054`**), que a su vez esta dentro de otro condicional que preguntó anteriormente por la MAC del switch (**`if(dpidToStr(event.dpid)=='00-00-00-00-00-01')`**), luego de haber ocurrido un evento capturado por la función **`def _handle_PacketIn (event).`** Mas adelante se entenderá a mayor detalle esta estructura.

Para los demás switches S2, S3, S5 y S6, en vez de hacer una operación FLOOD, se les indica que conmuten los paquetes ARP por el puerto conectado al switch de la derecha, siguiendo un sentido horario. Tal y como se enseña en la **Figura 7**.

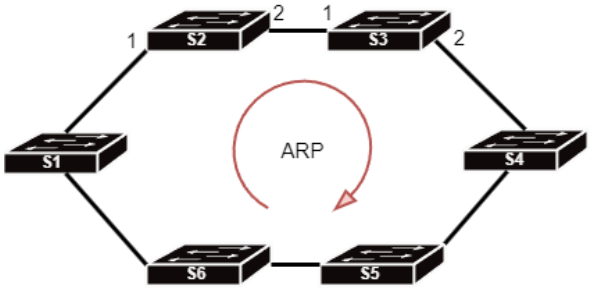

Figura 7: *Gestión paquetes ARP*

En la **Figura 7** se muestra que, en sentido horario hacia la derecha, los switches S2 y S3 están conectados al switch vecino a través del puerto número 2.

En este caso entonces, el siguiente fragmento de código le indica al controlador que, al capturar un paquete ARP proveniente de S2, lo conmute a través del puerto 2, para que lo reciba S3 siguiendo la regla del sentido horario.

```
if(dpidToStr(event.dpid)=='00-00-00-00-00-02'): #------- SW 2 -------#

                if packet.type == 2054: #ARPs
                        msg = of.ofp_packet_out()
                        msg.data = event.ofp
                        msg.actions.append(of.ofp_action_output(port = 2))
                        event.connection.send(msg)

                        #Instala Flujo
                        flow = of.ofp_flow_mod()
                        flow.match = of.ofp_match(dl_type =0x0806)
                        flow.actions.append(of.ofp_action_output(port = 2))
                        flow.hard_timeout = 50
                        event.connection.send(flow)
```

Habiendo solucionado la gestión del tráfico ARP, ahora nos compete gestionar el tráfico IP, de manera similar, simplemente se debe distinguir un paquete IP (`if packet.type == 2048),` y conociendo ya las direcciones IP origen o destino de interés, podemos, mediante un condicional `(if((str(ds_ip))=='10.0.0.7' and ((str(src_ip))=='10.0.0.1' or (str(src_ip))=='10.0.0.2' or (str(src_ip))=='10.0.0.3'))` indicarle al switch con un mensaje Open Flow la acción de conmutarlo por el puerto deseado (`flow.actions.append(of.ofp_action_output(port = 2))`).

A continuación, se enseña un fragmento de código en el que se le indica al controlador, que, al capturar un paquete IP, con destino '10.0.0.5' proveniente de S1, lo conmute a través del puerto 5. Esto, de igual manera, mediante un mensaje Open Flow. Nótese en el código que también se instala el flujo.

```
        if(dpidToStr(event.dpid)=='00-00-00-00-00-01'):
                if packet.type == 2048:
                        if((str(ds_ip)) =='10.0.0.5'):
```

```
                                msg = of.ofp_packet_out()
                                msg.data = event.ofp
                                msg.actions.append(of.ofp_action_output(port= 5))
                                event.connection.send(msg)

                                #Instala Flujo
                                flow = of.ofp_flow_mod()
                                flow.match = of.ofp_match(dl_type =0x0800, nw_dst =
IPAddr("10.0.0.5"))
                                flow.actions.append(of.ofp_action_output(port = 5))
                                flow.hard_timeout = 100
                                event.connection.send(flow)
```

Para finalizar, ya teniendo las herramientas necesarias para filtrar el tráfico IP y teniendo en cuenta el método ECMP, procedemos a indicarle a los switches que conmuten los paquetes con origen en h1, h2, o h3 y destino en h7, o con origen en h7 y destino en h1, h2, o h3, a través de S1, S2, S3 y S4. Y los paquetes con origen en h4, h5, o h6 y destino en h7, o con origen en h7 y destino en h4, h5, o h6, a través de los switches S1, S6, S5 y S4.

2.2.2. VLANs:

Para realizar esta práctica se debe entender que el controlador POX proactivo, como se explicó en **IV,** instala los flujos a los switches desde el momento en que estos realizan una conexión inicial al controlador.

Las librerías necesarias a importar son las mismas de la practica anterior, y, en este caso, al ser un controlador en modo proactivo, solo nos interesan los eventos recibidos a partir de las conexiones iniciales de los switches, para extraer sus direcciones MAC (`dpidToStr(event.dpid)=='00-00-00-00-00-01'`) e instalarles los flujos necesarios para satisfacer las condiciones de éxito planteadas anteriormente.

Lo más relevante para el desarrollo de esta práctica, es la estrategia para gestionar el trafico VLAN mediante SDN; se debe hacer énfasis en lo fácil y versátil que es en comparación a la gestión de este tráfico en una red convencional. En SDN, simplemente se necesita añadir una etiqueta (`vlan_vid`) para agregar el encabezado del protocolo 802.1q a los paquetes a conmutar que fueron originados en una VLAN específica. Esto se realiza con una acción que de igual manera se envía al switch mediante un mensaje Open Flow, así:

- `flow.actions.append(of.ofp_action_vlan_vid(vlan_vid = 10))`

Después de esto, el paquete etiquetado se conmuta en dirección al switch que alberga al host destinatario dentro de la misma VLAN.

En una red convencional se deben asignar en los switches puertos troncales si por estos se espera que viajen varios tipos de VLAN, esto no ocurre con SDN, simplemente se programa por cual(es) puerto(s) se desean conmutar y el switch que lo recibe tiene la capacidad de obtener el VLAN Id de cada paquete, dirección IP destino y conmutarlo al host correcto.

- `flow.match = of.ofp_match(in_port = 1, dl_vlan = 10)`

Gracias a la capacidad de los switches OpenFlow de acceder y modificar las cabeceras de los paquetes, el switch que contiene al host destinatario puede simplemente eliminar el encabezado 802.1q añadido y redirigir el paquete al puerto donde se encuentra el host destino, tratándolo como un paquete estándar.

Así es como se le indica al controlador POX proactivo que envíe al switch la acción de eliminar un encabezado 802.1q.

- `flow.actions.append(of.ofp_action_strip_vlan ())`

Con lo anterior, a continuación, se enseña un tramo de código que enseña como se gestiona el trafico para la VLAN 10, para un mayor entendimiento tener en cuenta la **Figura 4**:

```
def _handle_ConnectionUp (event):

        if dpidToStr(event.dpid)=='00-00-00-00-00-01': ##-- SW 1 --#

                # Paquetes de h1 vlan10 a h3 vlan10
                flow = of.ofp_flow_mod()
                flow.match = of.ofp_match(in_port = 1)
                # Se añade la etiqueta VLAN ID
                flow.actions.append(of.ofp_action_vlan_vid(vlan_vid =
10))
                flow.actions.append(of.ofp_action_output(port = 3))
                event.connection.send(flow)


                # Paquetes de h3 vlan10 a h1 vlan10
                flow = of.ofp_flow_mod()
                flow.match = of.ofp_match(in_port = 3, dl_vlan = 10)
                # Se remueve el protocolo 802.1q
                flow.actions.append(of.ofp_action_strip_vlan ())
                flow.actions.append(of.ofp_action_output(port = 1))
                event.connection.send(flow)


        if dpidToStr(event.dpid)=='00-00-00-00-00-02': ##----SW 2---##

                # Paquetes de h3 vlan10 a h1 vlan10
                flow = of.ofp_flow_mod()
                flow.match = of.ofp_match(in_port = 3)
                # Se añade la etiqueta VLAN ID
                flow.actions.append(of.ofp_action_vlan_vid(vlan_vid =
10))
                flow.actions.append(of.ofp_action_output(port = 1))
                event.connection.send(flow)

                # Paquetes de h1 vlan10 a h3 vlan10
                flow = of.ofp_flow_mod()
                flow.match = of.ofp_match(in_port = 1, dl_vlan = 10)
                # Se remueve el protocolo 802.1q
                flow.actions.append(of.ofp_action_strip_vlan ())
                flow.actions.append(of.ofp_action_output(port = 3))
                event.connection.send(flow)
```

En los anexos finales (Sección **VII**) se encuentran los códigos comentados, y las plantillas de ambas practicas los cuales se le entregaran al estudiante para que realice su práctica.

2.3. Realizar ajustes necesarios según errores hallados en la actividad anterior:

Luego de revisar las practicas, no se hallaron errores, lo cual brindo un poco más de tiempo para pensar en la inclusión de puntos extra para cada práctica. De esta manera si el estudiante los realiza exitosamente además de un posible estimulo académico podrá comprender mas a fondo el funcionamiento de estas herramientas. Así las cosas, para ambas practicas se plantearon puntos extras, que se enseñaran a continuación.

Balanceo de carga:

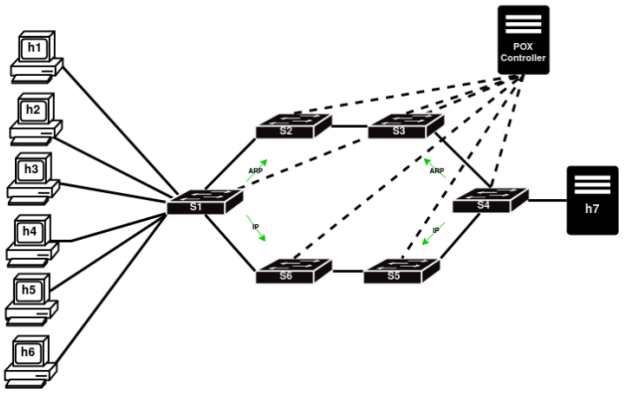

Figura 8: *Topología punto extra practica balanceo de carga en enlaces.*

En este punto extra se propone filtrar no por IP origen como se realizó anteriormente si no por protocolo. De esta manera, como se observa en la **Figura 8**, los paquetes ARP se conmutarán a través de los switches S1, S2, S3 y S4, y los paquetes IP a través de los switches S1, S6, S5 y S4.

El éxito del desarrollo de este punto extra se demuestra con una operación Ping exitosa entre todos los hosts y enseñando los flujos en todos los switches, probando que se cumplen las condiciones de filtrado anteriores.

VLANs:

Como el punto principal de esta práctica fue el de desarrollarla mediante un código para un controlador POX proactivo, para una mayor exploración de este tema se proponen los siguientes puntos extra:

1. **Agregar un nuevo switch y una nueva VLAN:** En el código que se le proporcionará al estudiante para generar la topología de la **Figura 4** y realizar pruebas (Esto se explica con más detalles en 3.1), el estudiante deberá agregar un nuevo switch S3 que este entre S1 y S2 y crear una nueva VLAN.

   De igual manera, el éxito del desarrollo de esta práctica se demuestra mediante una operación Ping exitosa únicamente entre los hosts pertenecientes a cada VLAN. Además, se deberá de enseñar los flujos de cada switch indicando como se asignaron las etiquetas a cada VLAN (VLAN ID) y cómo se gestionó este tráfico.

2. **Realizar el código para un controlador POX reactivo:** Se tiene el mismo objetivo planteado en el ítem **2.2**, solo que esta vez se pide el código para un controlador POX reactivo. Como se vio

anteriormente en **2.2.1**, POX en modo reactivo recibe constantes eventos por parte de los switches, de los que se extrae información de los paquetes de interés. En este caso le match se hará principalmente con la MAC del switch y el puerto de entrada de los paquetes:

- `if(dpidToStr(event.dpid)=='00-00-00-00-00-02'):#-SW 2-#`

  `if event.port == 3:`

Se deberá tener en cuenta los métodos y funciones (Explicados también en la guía y los códigos de los anexos en la sección **VII**) necesarios para identificar el tráfico etiquetado con encabezados del protocolo 802.1Q y acceder a la etiqueta (Id) del encabezado de un paquete capturado. La sintaxis para esto es:

- **`if packet.type == ethernet.VLAN_TYPE:`** Con esta linea de codigo se puede identificar si un paquete capturado es tipo VLAN.

- **`VLANID = packet.next.id:`** Con esta línea de código se puede capturar el VLAN Id de un paquete capturado.

Con lo anterior, a continuación, se enseña una parte del código que enseña cómo se gestiona el tráfico para la VLAN 10, para un mayor entendimiento tener en cuenta la **Figura 4**:

```
def _handle_PacketIn (event):

    packet = event.parsed

    if event.parsed.type != 34525: # Distingue de paquetes IPV6

        if(dpidToStr(event.dpid)=='00-00-00-00-00-01'):#-SW 1-#

            if event.port == 1:

                # se le envia la accion a ejecutar
                msg = of.ofp_packet_out()
                msg.data = event.ofp
                # Se añade la etiqueta VLAN ID
                msg.actions.append(of.ofp_action_vlan_vid(vlan_vid=10))
                msg.actions.append (of.ofp_action_output(port = 3))
                event.connection.send(msg)


            if event.port == 3:

                # Si es un paquete VLAN
                if packet.type == ethernet.VLAN_TYPE:

                    if packet.next.id == 10: #VLAN 10

                        # se le envia la accion a ejecutar
                        msg = of.ofp_packet_out()
                        msg.data = event.ofp
                    # Se remueve el protocolo 802.1q
                        msg.actions.append(of.ofp_action_strip_vlan ())
                        msg.actions.append(of.ofp_action_output(port=1))
                        event.connection.send(msg)

    if(dpidToStr(event.dpid)=='00-00-00-00-00-02'):#-SW 2-#

            if event.port == 3:
```

```
                                        # se le envia la accion a ejecutar
                                        msg = of.ofp_packet_out()
                                        msg.data = event.ofp
                                        # Se añade la etiqueta VLAN ID
                                        msg.actions.append(of.ofp_action_vlan_vid(vlan_vid = 10))
                                        msg.actions.append(of.ofp_action_output(port = 1))
                                        event.connection.send(msg)

                                        if event.port == 1:

                                                # Si es un paquete VLAN
                                                if packet.type == ethernet.VLAN_TYPE:

                                                        if packet.next.id == 10: #VLAN 10

                                                                # se le envia la accion a ejecutar
                                                                msg = of.ofp_packet_out()
                                                                msg.data = event.ofp
                                                        # Se remueve el protocolo 802.1q
                                                                msg.actions.append(of.ofp_action_strip_vlan ())
                                                                msg.actions.append(of.ofp_action_output(port = 3))
                                                                event.connection.send(msg)

def launch ():
        core.openflow.addListenerByName("PacketIn", _handle_PacketIn)
```

En los anexos finales (Sección **VII**) se encuentran los códigos de los puntos extra debidamente comentados. No se consideró necesario realizar plantillas de los códigos para estos puntos extra.

3. Evaluar la implementación de las practicas a través de un protocolo de pruebas definida

3.1. Elaborar el protocolo de pruebas

Para ambas practicas se elaboraron en VSC códigos que generarán las topologías, y acorde con la practica tendrán las siguientes funciones para probar la efectividad del código realizado para el controlador:

- **Balanceo de carga por enlaces (`Topo_BC.py`):** Este script, a parte de generar la topología de la **Figura 3**, contará con las siguientes opciones a parte de generar la topología:
    - **Línea de comandos Mininet:** Abre la interfaz de mininet.
    - **Conectividad total (Pingall):** Realiza la operación pingall, que hace un ping global
    - **Flujos:** Enseña los flujos que tengan instalados los
    - **Abrir terminales:** Abre las terminales de los equipos pertenecientes a la topología. Para esta función se recomienda tener instalado Xterm.
    - **Ver flujos en x SW:** Enseña los flujos del switch deseado.
    - **PING entre dos hosts:** Realiza una prueba de conexión entre dos hosts específicos.
    - **IPERF entre dos hosts:** Comprueba la tasa de transferencia entre dos hosts.
    - **SALIR:** Cierra el script y ejecuta el comando `mn -c` que elimina todos los procesos y contenedoras relacionados con la red creada en mininet.

- **VLANs (`TopoVLAN.py`):** De igual forma, este script genera la topología de interés (**Figura 4**) y cuenta con las siguientes opciones:

    - **Línea de comandos Mininet:** Abre la interfaz de mininet.
    - **Conectividad total (Pingall):** Realiza la operación pingall, que hace un ping global.
    - **Flujos:** Enseña los flujos que instalados en los switches.
    - **Abrir terminales:** Abre las terminales de los equipos pertenecientes a la topología. Para esta función se recomienda tener instalado Xterm.
    - **Ver flujos en x SW:** Enseña los flujos del switch X deseado.
    - **PING entre dos hosts:** Realiza una prueba de conexión entre dos hosts específicos.
    - **IPERF entre dos hosts:** Comprueba la tasa de transferencia entre dos hosts.
    - **SALIR:** Cierra el script y ejecuta el comando `mn -c` que elimina todos los procesos y contenedoras relacionados con la red creada en Mininet.

Estos códigos se le facilitaran al estudiante para que realice las pruebas de su código.

En los anexos al final de este informe (Sección **VII**) se encuentran los códigos con los que el estudiante podrá generar las topologías de las practicas y poner a prueba sus códigos.

3.2. Ejecutar las pruebas

Por comodidad, se recomienda dividir la consola de la máquina virtual de Mininet en dos (**Figura 9**). En una parte ejecutar el script que genera la topología y en la otra poner el código del controlador. El código del controlador se debe ubicar en el directorio **`/home/pox/pox`**, y se ejecuta con la siguiente línea de código **`./pox.py log.level --DEBUG`** ***`nombredelcodigopox`*** desde el directorio **`/home/`*`nombredelamaquina`*`/pox`** (**Figura 9**).

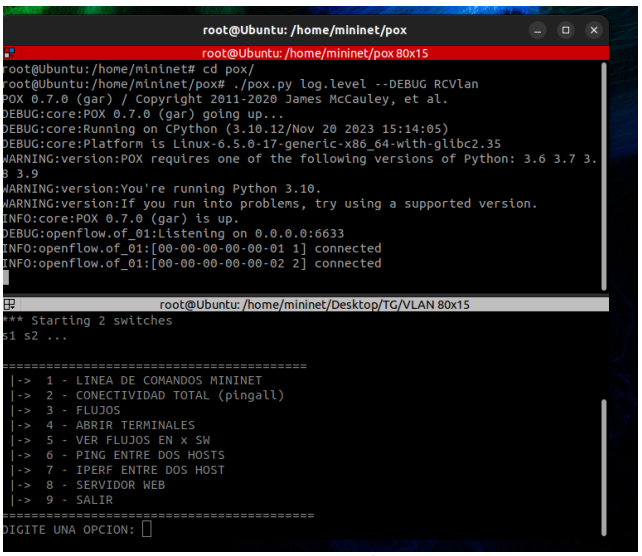

Figura 9: *Prueba en ejecución, practica VLAN.*

Gracias a las funciones de los códigos `Topo_BC.py` y `TopoVLAN.py` descritas en 3.1, es bastante sencillo ejecutar las pruebas para comprobar que los códigos funcionan correctamente. Por ejemplo, la **Figura 10** enseña una operación `Pingall` en la práctica VLAN demostrando que el código realizado fue el correcto.

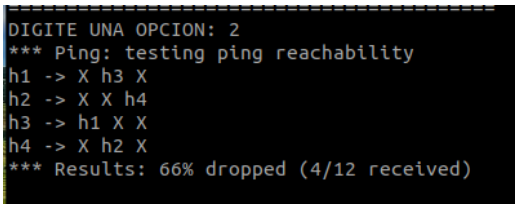

Figura 10: *Prueba en ejecución, practica VLAN, función Pingall.*

3.3. Realizar las guías correspondientes a cada una de las practicas

Al tratarse de temas un poco más avanzados en un curso inicial de SDN, se espera que el estudiante ya tengo ciertas bases y conocimientos relacionados.

Las guías, se desarrollaron explicando los temas nuevos requeridos para la realización de estas prácticas. También se tuvo en cuenta el explicar las sintaxis más importantes para la realización de los códigos. Lo anterior, haciendo especial énfasis en los nuevos conceptos. Se podría afirmar que el estudiante contará con recursos suficientes y no tendrá que hacer uso de otras fuentes.

En los anexos al final de este informe (Sección **VII**) se encuentran las guías realizadas que orientaran al estudiante con sus instrucciones detalladas.

3.4. Revisar documentación y realizar correcciones

Finalmente, se revisaron detalladamente toda la documentación y códigos realizados en las etapas anteriores, prestando especial atención a la claridad de los conceptos, la cohesión de las idas y la corrección gramatical. Se realizaron las correcciones necesarias para garantizar que el estudiante comprenda fácilmente que debe realizar y con que herramientas y ayudas cuenta para ello.

VI. Conclusiones

Se realizo una investigación de los temas mas relevantes para un curso de redes programables, de los cuales se escogieron los mas aptos para la realización de dos practicas a partir de los recursos disponibles. Los temas escogidos fueron Balanceo de carga en enlaces y redes VLAN. Esto, ya que, las redes VLANs son ampliamente implementadas en infraestructura de empresas, y las técnicas de Balanceo de carga en enlaces son necesarias para optimizar y hacer un mayor aprovechamiento de los

recursos de red que dan acceso a los servidores, altamente demandados en la actualidad.

Para la elaboración de las practicas, se diseñaron los escenarios y topologías teniendo en cuenta las herramientas de software disponibles, a saber: máquina virtual de Mininet, Visual Studio Code y POX controller. Haciendo uso de estas herramientas fue posible la realización y depuración de los códigos para las topologías, protocolo de pruebas, controladores y códigos plantillas que se le entregaran al estudiante junto con las guías, también desarrolladas desde cero, para que este se oriente y desarrolle la práctica.

Se debe resaltar la facilidad y versatilidad que ofrece SDN para desarrollar estos escenarios de VLAN y balanceo de carga en enlaces en comparación con la forma en que se hace con las redes convencionales. Ya que por ejemplo para las VLAN no es necesario hacer uso de puertos troncales y es mucho mas sencilla su configuración. Y, para el escenario de balanceo de carga en enlaces es posible realizarlo sin la necesidad de routers, solo instalando flujos en switchs OpenFlow se ahorran costos, tiempo y se tiene un control mas centralizado de la gestión del trafico de la red ofreciendo más posibilidades y reduciendo tareas de administración.

VII. Anexos

Se anexa a continuación para la práctica de Balanceo de carga por enlaces:

1. **Guía:** Contiene una breve explicación de los conceptos clave para desarrollar la práctica, el enunciado de la misma y algunas ayudas.
2. **Topología y ejecutor de pruebas:** Script que utilizara el estudiante para generar la topología de esta práctica y poner a prueba su código
3. **BCenlaces Reactivo:** Código que da solución a la problemática planteada para el desarrollo de la práctica.
4. **BCenlaces_plantilla:** Esquema del código, que se entregara como base al estudiante para que este lo complete y desarrolle la práctica.
5. **BCenlacesPorProtocolos:** Código solución al punto extra planteado en la guía de la práctica.

Guía:

El balanceo de carga es una técnica utilizada en redes informáticas para distribuir de manera equitativa la carga de trabajo o el tráfico entre múltiples recursos, como servidores, enlaces de red o dispositivos de almacenamiento. El objetivo principal del balanceo de carga es mejorar el rendimiento, la disponibilidad y la eficiencia de los sistemas al evitar la sobrecarga de un recurso específico y distribuir de manera uniforme el tráfico, tareas o solicitudes entre varios componentes. Esto puede lograrse mediante diversos algoritmos y estrategias que asignan las solicitudes a los recursos de manera equitativa, teniendo en cuenta factores como la capacidad, la carga actual y la disponibilidad de cada recurso.

El balanceo de carga se utiliza comúnmente en entornos de servidores web, aplicaciones empresariales, redes de entrega de contenido (CDN) y en cualquier situación donde la distribución uniforme de la carga mejore la eficiencia y la capacidad de respuesta del sistema y su capacidad para manejar mayores volúmenes de tráfico. También puede proporcionar tolerancia a fallos al redirigir automáticamente el tráfico lejos de los recursos que pueden estar experimentando problemas.

En esta práctica se implementa una forma de balanceo de carga conocida como "Equal-Cost Multipath (ECMP)" routing. En este escenario, un enrutador tiene múltiples rutas hacia un destino y distribuye el tráfico de manera equitativa entre esas rutas. Esto se hace para optimizar la utilización de la red y mejorar el rendimiento general. Al enviar tráfico a través de múltiples rutas, el enrutador busca distribuir la carga de manera uniforme, evitando la congestión en una sola ruta. Esto ayuda a utilizar de manera más eficiente los recursos de red disponibles.

Actividad:

En esta práctica, con la ayuda del controlador POX en modo reactivo, se diseñará un balanceador de carga que, analizando la dirección IP de origen y/o destino, decidirá en qué dirección se enviarán los paquetes.

En la **Figura 1** de la topología:

- Representa el flujo del tráfico IP con origen h1-h3 hacia el servidor h7 y viceversa.
- Representa el flujo del tráfico IP con origen h4-h6 hacia el servidor h7 y viceversa.

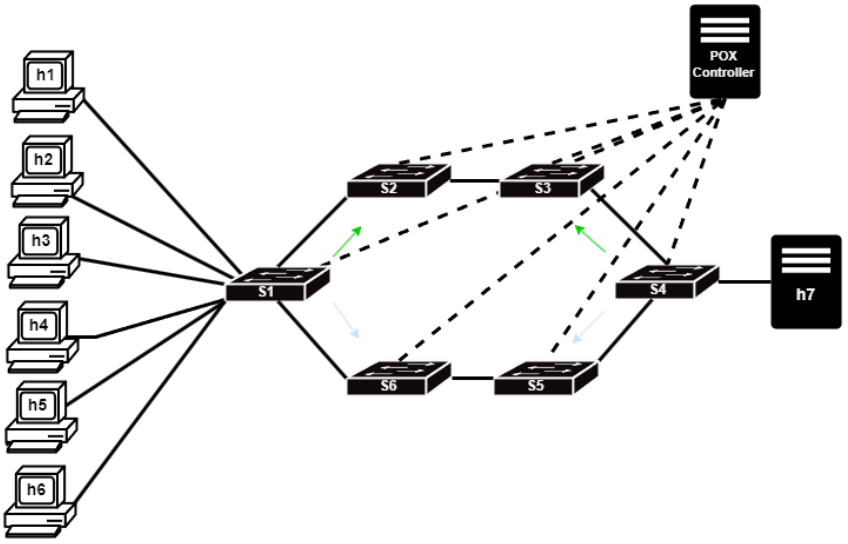

Figura 1: *Topología balanceador de carga*

De esta manera se deberá elaborar un código que le indique al controlador POX en modo reactivo cómo manejar el tráfico ARP e IP para que haya una conexión total. Lo anterior, con la condición de que el flujo del tráfico IP se divida así:

- Tráfico IP con origen h1-h3 hacia el servidor h7 y viceversa solo lo gestionarán los switches S1, S2, S3 y S4.
- Tráfico IP con origen h4-h6 hacia el servidor h7 y viceversa solo lo gestionarán los switches S1, S6, S5 y S4.

Recuerde que debe encontrar una solución para evitar una tormenta broadcast en las resoluciones ARP. Para ello, se recomienda que estos paquetes sigan una trayectoria en sentido horario.

Escenario de pruebas:

Para comprobar el correcto funcionamiento del código se cuenta con un script (`Topo_BC.py`) que genera la topología de la **Figura 1** y además cuenta con estas opciones:

1. **Línea de comandos Mininet:** Abre la interfaz de Mininet.
2. **Conectividad total (Pingall):** Realiza la operación Pingall, que hace un ping global.
3. **Flujos:** Muestra los flujos que estén instalados en todos los switch.
4. **Abrir terminales:** Abre las terminales de los equipos pertenecientes a la topología. Para esta función se recomienda tener instalado Xterm.
5. **Ver flujos en x SW:** Muestra los flujos del switch deseado.
6. **PING entre dos hosts:** Realiza una prueba de conexión entre dos hosts específicos.
7. **IPERF entre dos hosts:** Comprueba la tasa de transferencia entre dos hosts.
8. **SALIR:** Cierra el script y ejecuta el comando `mn -c` que elimina todos los procesos y contenedores relacionados con la red creada en Mininet.

Tip: Se recomienda dividir la consola de esta manera para mayor comodidad a la hora de realizar las pruebas.

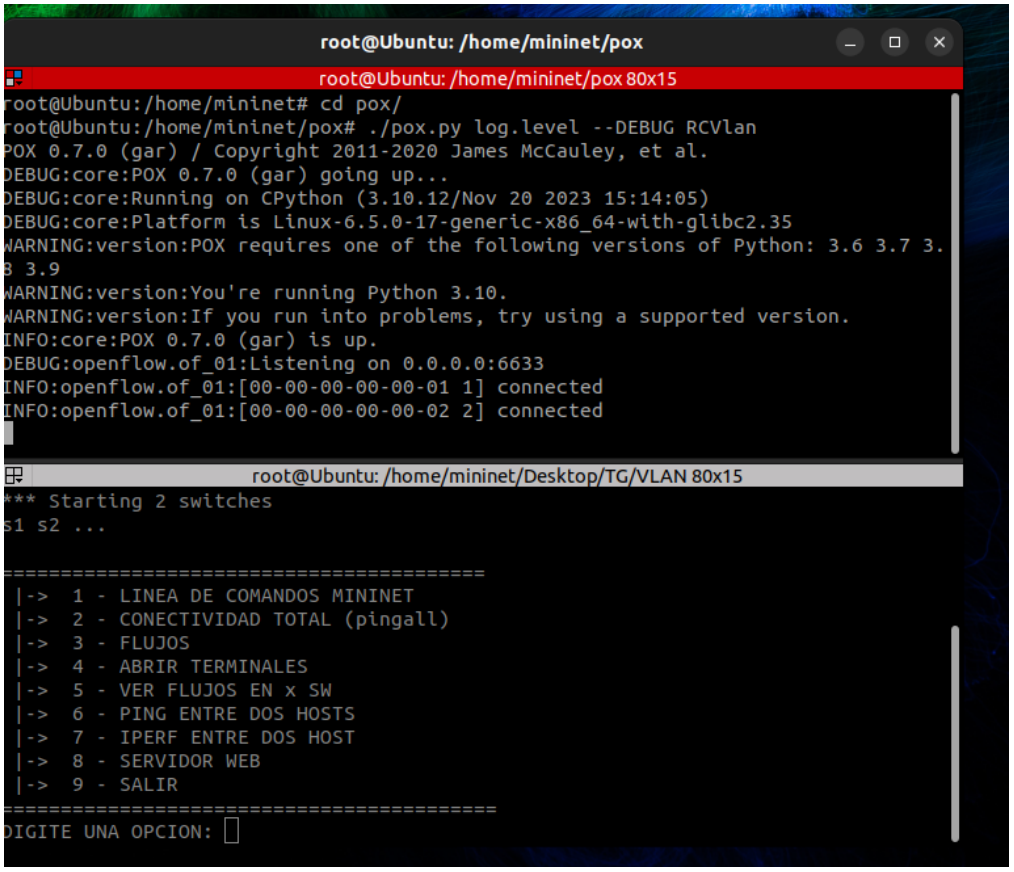

Figura 2: *Prueba en ejecución, practica VLAN.*

Algunos métodos y funciones útiles para la realización de su código:

Al tratarse de un controlador en modo reactivo, cuando los switches no encuentran una acción para un paquete que reciben, envían mensajes hacia el controlador para que les indique qué hacer con ellos. Para ello se implementa la siguiente función que recibe dichos mensajes event.

```
def _handle_PacketIn (event):
```

Podemos saber qué switch está enviando los mensajes extrayendo su dirección MAC. De esta manera, mediante un condicional, se puede indicar al switch 1 que realice ciertas acciones con los paquetes que este reciba:

```
if(dpidToStr(event.dpid)=='00-00-00-00-00-01'): #-- SW 1 --#
```

Así se analizan los paquetes de interés y se logra extraer información importante, como por ejemplo la dirección MAC destino de este paquete.

```
packet = event.parsed
mac_destino= packet.dst
```

Como solo nos interesan los paquetes IPv4, de esta manera se pueden omitir los que sean tipo IPv6:

```
if event.parsed.type != 34525: # Distingue de paquetes IPV6
```

Así se distinguen paquetes APRs de IPs:

```
if packet.type == 2054: #ARPs
if packet.type == 2048: # IPs
```

También se puede extraer las direcciones IP origen y destino:

```
ds_ip = packet.next.dstip
src_ip = packet.next.srcip
```

De esta manera se crea un mensaje openflow que se envía al switch indicando que acción realizar. En este caso se indica que realice una acción FLOOD, útil para el manejo de paquetes ARP.

```
# se le envía la acción a ejecutar
 msg = of.ofp_packet_out()
```

```
 msg.data = event.ofp
 msg.actions.append(of.ofp_action_output(port =of.OFPP_FLOOD))
 event.connection.send(msg)
```

Al tratarse de un controlador reactivo los switch siempre le preguntaran al controlador que deben hacer con cada paquete, si no hay una entrada en la tabla de flujo para ese tipo de tráfico. Si un determinado tipo de paquetes llegan muy frecuentemente al switch (por ejemplo solicitudes ARP) puede ser util instalar un flujo temporal que le indique al switch que hacer con ellos para asi optimizar tiempos y evitar retrasos.

```
# Instala el flujo temporalmente
 flow = of.ofp_flow_mod()
 flow.match = of.ofp_match(dl_type =0x0806)
 flow.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
 flow.hard_timeout = 100
 event.connection.send(flow)
```

Nótese que se puede saber que tipo de paquete ha llegado al controlador revisando el campo `dl_type` la cabecera ethernet:

```
dl_type =0x0806 Para ARP y
dl_type =0x0800 Para IP
```

De esta manera se le envía un mensaje openflow al switch que indica que hacer con paquetes IP que vayan hacia `'10.0.0.5'`. Luego se procede instalando el flujo:

```
if((str(ds_ip))=='10.0.0.5'):
msg = of.ofp_packet_out()
msg.data = event.ofp
msg.actions.append(of.ofp_action_output(port= 5))
event.connection.send(msg)

flow = of.ofp_flow_mod()
flow.match = of.ofp_match(dl_type =0x0800, nw_dst = IPAddr("10.0.0.5"))
flow.actions.append(of.ofp_action_output(port = 5))
flow.hard_timeout = 100
event.connection.send(flow)
```

Extra:

Hacer balanceo por protocolo ARP e IP de la siguiente manera:

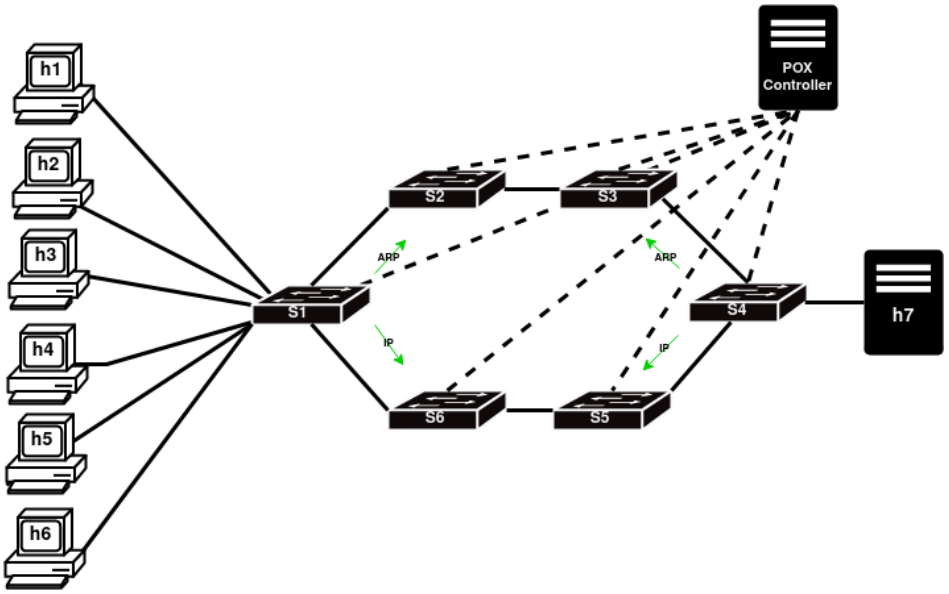

Figura 3: *Punto extra balanceo por protocolo ARP e IP..*

Se propone filtrar no por IP origen como se realizó anteriormente si no por protocolo. De esta manera, como se observa en la **Figura 3**, los paquetes ARP se conmutarán a través de los switches S1, S2, S3 y S4, y los paquetes IP a través de los switches S1, S6, S5 y S4.

Topología y ejecutor de pruebas:

```
#!/usr/bin/python
from mininet.net import Mininet
from mininet.cli import CLI
from mininet.node import OVSSwitch, RemoteController
from mininet.topo import Topo, SingleSwitchTopo
from mininet.log import setLogLevel, info
from mininet.util import dumpNodeConnections
from mininet.link import TCLink
from mininet.term import makeTerm
from time import sleep

# Utilizar este programa para generar la topología y probar su script para el controlador

import os

import sys
def startNetwork():
        net = Mininet(switch=OVSSwitch)

        h1=net.addHost('h1', mac='00:00:00:00:10:01', ip='10.0.0.1/24')
        h2=net.addHost('h2', mac='00:00:00:00:10:02', ip='10.0.0.2/24')
        h3=net.addHost('h3', mac='00:00:00:00:10:03', ip='10.0.0.3/24')
        h4=net.addHost('h4', mac='00:00:00:00:10:04', ip='10.0.0.4/24')
        h5=net.addHost('h5', mac='00:00:00:00:10:05', ip='10.0.0.5/24')
        h6=net.addHost('h6', mac='00:00:00:00:10:06', ip='10.0.0.6/24')

        # Server
        h7=net.addHost('h7', mac='00:00:00:00:10:07', ip='10.0.0.7/24')

        net.addSwitch('s1', mac='00-00-00-00-00-01')
        net.addSwitch('s2', mac='00-00-00-00-00-02')
        net.addSwitch('s3', mac='00-00-00-00-00-03')
        net.addSwitch('s4', mac='00-00-00-00-00-04')
        net.addSwitch('s5', mac='00-00-00-00-00-05')
        net.addSwitch('s6', mac='00-00-00-00-00-06')

        net.addLink('h1','s1',0,1) # net.addLink(Node1,Node2,portNode1,portNode2)
        net.addLink('h2','s1',0,2)
        net.addLink('h3','s1',0,3)
        net.addLink('h4','s1',0,4)
        net.addLink('h5','s1',0,5)
        net.addLink('h6','s1',0,6)

        net.addLink('h7','s4',0,1)

        net.addLink('s1','s2',7,1)
        net.addLink('s2','s3',2,1)
        net.addLink('s3','s4',2,2)
        net.addLink('s4','s5',3,1)
        net.addLink('s5','s6',2,2)
        net.addLink('s6','s1',1,8)

        net.addController('c0',controller=RemoteController, ip='127.0.0.1', port=6633)

        net.start()

        option=0
        while option !=1:
                info('\n=======================================')
                info('\n |->  1 - LINEA DE COMANDOS MININET')
                info('\n |->  2 - CONECTIVIDAD TOTAL (pingall)')
                info('\n |->  3 - VER FLUJOS')
                info('\n |->  4 - ABRIR TERMINALES')
                info('\n |->  5 - VER FLUJOS EN X SWITCH')
                info('\n |->  6 - PING ENTRE DOS HOST')
                info('\n |->  7 - IPERF ENTRE DOS HOST')
                info('\n |->  8 - SALIR')
```

```
                info('\n=========================================\n')
                opt =input("DIGITE UNA OPCION: ")
                if opt.isdigit():
                        option =int(opt)

                #================================================
                if option ==2:
                        net.pingAll()
                option =0
                #================================================

                elif option ==3:
                        for sw in net.switches:
                                info ("%s "% sw.name)
                                os.system ("sudo ovs-ofctl dump-flows %s "% sw.name)

        info('\n=========================================================================
==========================================================\n')
                option =0

        #================================================
                elif option ==4:
                        net.startTerms()
                option =0
        #================================================

                elif option ==5:
                        info("INGRESE EL NUMERO DE UN SW\r\n")
                        s = input("switch: ")
                        sw = net.get(s)
                        os.system("sudo ovs-ofctl dump-flows %s" % sw.name)
                option =0

        #================================================

                elif option == 6:
                        info("PING ENTRE DOS HOST \r\n")
                        h_ping_1 = input("HOST 1: ")
                        h_ping_2 = input("HOST 2: ")
                        host1,host2 = net.get(h_ping_1,h_ping_2)
                        net.ping((host1,host2))
                option =0

        #================================================

                elif option ==7:

                        info("**IPERF ENTRE DOS HOST** \r\n")
                        host_1_iperf = input("HOST1: ")
                        host_2_iperf = input("HOST2: ")
                        host1,host2 = net.get(host_1_iperf,host_2_iperf)
                        net.iperf((host1,host2),l4Type='TCP')
                option =0


                elif option ==8:
                        net.stop()
                        sys.exit()

        CLI ( net )
#////////////////////////////////////////////////////////
if __name__ =='__main__':
        setLogLevel('info')
        startNetwork()
```

BCenlaces Reactivo:

```
#/usr/bin/python
```

```
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpidToStr
import pox.lib.packet as pkt
from pox.lib.addresses import EthAddr, IPAddr
from pox.lib.revent import *
from pox.lib.util import dpid_to_str
from pox.lib.packet.ethernet import ethernet
from pox.lib.packet.ipv4 import ipv4


log = core.getLogger()


def _handle_PacketIn (event):

        packet = event.parsed  # Se usa para obtener los datos del paquete analizado.
pueden usarse para diversos propósitos, como extraer direcciones MAC de origen y destino,
direcciones IP u otros campos


                if event.parsed.type != 34525: # Distingue de paquetes IPV6

                        if(dpidToStr(event.dpid)=='00-00-00-00-00-01'): #---------------------------------- SW 1 ---
        ----------------------------------------------------------#


                                if packet.type == 2054: # Tipo de paquete ARP

                                # se le envia la accion a ejecutar

                                # Crea un mensaje OFPT_PACKET_OUT, que se utiliza para enviar un
                        paquete desde el controlador hacia un switch OpenFlow
                                msg = of.ofp_packet_out()

                                # Establece los datos del mensaje como los datos del paquete
                        recibido
                                msg.data = event.ofp

                                # Establecer la accion, en este caso, el puerto de salida del
switch
                                msg.actions.append(of.ofp_action_output(port
                                =of.OFPP_FLOOD))

                                # Envía el mensaje al switch OpenFlow para su
                                procesamiento
                                event.connection.send(msg)

                                # Instala el flujo temporalmente

                                flow = of.ofp_flow_mod()
                                flow.match = of.ofp_match(dl_type =0x0806)
                                flow.actions.append(of.ofp_action_output(port =
                        of.OFPP_FLOOD))
                                # Instala el flujo temporalmente
                                flow.hard_timeout = 50
                                event.connection.send(flow)


                                if packet.type == 2048: # IPs

                                        ds_ip = packet.next.dstip # Se obtiene la direccion IP destino
                                        del paquete   capturado

                                        src_ip = packet.next.srcip # Se obtiene la direccion IP origen
                                        del paquete   capturado

                                        # Paquetes IPs hacia h1
                                        if((str(ds_ip))=='10.0.0.1'):
```

```
                msg = of.ofp_packet_out()
                msg.data = event.ofp
                msg.actions.append(of.ofp_action_output(port= 1))
                event.connection.send(msg)

                # Instala el flujo temporalmente
                flow = of.ofp_flow_mod()
                flow.match = of.ofp_match(dl_type =0x0800, nw_dst =
IPAddr("10.0.0.1"))
                flow.actions.append(of.ofp_action_output(port = 1))
                flow.hard_timeout = 50
                event.connection.send(flow)

            #Paquetes IPs hacia h2
            if((str(ds_ip))=='10.0.0.2'):

                msg = of.ofp_packet_out()
                msg.data = event.ofp
                msg.actions.append(of.ofp_action_output(port= 2))
                event.connection.send(msg)

                # Instala el flujo temporalmente
                flow = of.ofp_flow_mod()
                flow.match = of.ofp_match(dl_type =0x0800, nw_dst =
IPAddr("10.0.0.2"))
                flow.actions.append(of.ofp_action_output(port = 2))
                flow.hard_timeout = 50
                event.connection.send(flow)

            #Paquetes IPs hacia h3
            if((str(ds_ip))=='10.0.0.3'):

                msg = of.ofp_packet_out()
                msg.data = event.ofp
                msg.actions.append(of.ofp_action_output(port= 3))
                event.connection.send(msg)

                # Instala el flujo temporalmente
                flow = of.ofp_flow_mod()
                flow.match = of.ofp_match(dl_type =0x0800, nw_dst =
IPAddr("10.0.0.3"))
                flow.actions.append(of.ofp_action_output(port = 3))
                flow.hard_timeout = 50
                event.connection.send(flow)

            #Paquetes IPs hacia h4
            if((str(ds_ip))=='10.0.0.4'):

                msg = of.ofp_packet_out()
                msg.data = event.ofp
                msg.actions.append(of.ofp_action_output(port= 4))
                event.connection.send(msg)

                flow = of.ofp_flow_mod()
                flow.match = of.ofp_match(dl_type =0x0800, nw_dst =
IPAddr("10.0.0.4"))
                flow.actions.append(of.ofp_action_output(port = 4))
                flow.hard_timeout = 50
                event.connection.send(flow)

            #Paquetes IPs hacia h5
            if((str(ds_ip))=='10.0.0.5'):

                msg = of.ofp_packet_out()
                msg.data = event.ofp
                msg.actions.append(of.ofp_action_output(port= 5))
                event.connection.send(msg)
```

```
                                        flow = of.ofp_flow_mod()
                                        flow.match = of.ofp_match(dl_type =0x0800, nw_dst =
IPAddr("10.0.0.5"))
                                        flow.actions.append(of.ofp_action_output(port = 5))
                                        flow.hard_timeout = 50
                                        event.connection.send(flow)

                                #Paquetes IPs hacia h6
                                if((str(ds_ip))=='10.0.0.6'):

                                        msg = of.ofp_packet_out()
                                        msg.data = event.ofp
                                        msg.actions.append(of.ofp_action_output(port= 6))
                                        event.connection.send(msg)

                                        flow = of.ofp_flow_mod()
                                        flow.match = of.ofp_match(dl_type =0x0800, nw_dst =
IPAddr("10.0.0.6"))
                                        flow.actions.append(of.ofp_action_output(port = 6))
                                        flow.hard_timeout = 50
                                        event.connection.send(flow)

                                #Paquetes IPs hacia h7 con origen h1-3

                                if((str(ds_ip))=='10.0.0.7' and ((str(src_ip))=='10.0.0.1' or
(str(src_ip))=='10.0.0.2' or (str(src_ip))=='10.0.0.3')):

                                        msg = of.ofp_packet_out()
                                        msg.data = event.ofp
                                        msg.actions.append(of.ofp_action_output(port= 7))
                                        event.connection.send(msg)

                                        for i in range (1,4):
                                                flow = of.ofp_flow_mod()
                                                flow.match = of.ofp_match(dl_type =0x0800,
                        nw_src = IPAddr("10.0.0.%s"%i),nw_dst = IPAddr("10.0.0.7"))
                                        flow.actions.append(of.ofp_action_output(port = 7))
                                                flow.hard_timeout = 50
                                                event.connection.send(flow)

                                #Paquetes IPs hacia h7 con origen h4-6

                                if((str(ds_ip))=='10.0.0.7' and ((str(src_ip))=='10.0.0.4' or
(str(src_ip))=='10.0.0.5' or (str(src_ip))=='10.0.0.6')):

                                        msg = of.ofp_packet_out()
                                        msg.data = event.ofp
                                        msg.actions.append(of.ofp_action_output(port= 8))
                                        event.connection.send(msg)

                                        for i in range (4,7):
                                                flow = of.ofp_flow_mod()
                                                flow.match = of.ofp_match(dl_type =0x0800,
                        nw_src = IPAddr("10.0.0.%s"%i),nw_dst = IPAddr("10.0.0.7"))
                                        flow.actions.append(of.ofp_action_output(port = 8))
                                                flow.hard_timeout = 50
                                                event.connection.send(flow)

                        if(dpidToStr(event.dpid)=='00-00-00-00-00-02'): #---------------------------------- SW 2 ---
-----------------------------------------------------------#

                                if packet.type == 2054:   # ARPs
                                        msg = of.ofp_packet_out()
                                        msg.data = event.ofp
                                        msg.actions.append(of.ofp_action_output(port = 2))
                                        event.connection.send(msg)

                                        flow = of.ofp_flow_mod()
                                        flow.match = of.ofp_match(dl_type =0x0806)
```

```
                flow.actions.append(of.ofp_action_output(port = 2))
                flow.hard_timeout = 50
                event.connection.send(flow)

        if packet.type == 2048:   # IPs

                ds_ip = packet.next.dstip
                src_ip = packet.next.srcip

                #Paquetes IPs hacia h7 con origen h1-3

                if((str(ds_ip))=='10.0.0.7'):

                        msg = of.ofp_packet_out()
                        msg.data = event.ofp
                        msg.actions.append(of.ofp_action_output(port= 2))
                        event.connection.send(msg)


                        flow = of.ofp_flow_mod()
                        flow.match = of.ofp_match(dl_type =0x0800,nw_dst =
IPAddr("10.0.0.7"))
                        flow.actions.append(of.ofp_action_output(port = 2))
                        flow.hard_timeout = 50
                        event.connection.send(flow)


                #Paquetes IPs hacia h1-3 con origen h7

                if((str(src_ip))=='10.0.0.7'):

                        msg = of.ofp_packet_out()
                        msg.data = event.ofp
                        msg.actions.append(of.ofp_action_output(port= 1))
                        event.connection.send(msg)


                        flow = of.ofp_flow_mod()
                        flow.match = of.ofp_match(dl_type =0x0800, nw_src =
IPAddr("10.0.0.7"))
                        flow.actions.append(of.ofp_action_output(port = 1))
                        flow.hard_timeout = 50
                        event.connection.send(flow)


        if(dpidToStr(event.dpid)=='00-00-00-00-00-03'): #---------------------------------- SW 3 ---
----------------------------------------------------------#

        if packet.type == 2054:   # ARPs

                msg = of.ofp_packet_out()
                msg.data = event.ofp
                msg.actions.append(of.ofp_action_output(port = 2))
                event.connection.send(msg)

                flow = of.ofp_flow_mod()
                flow.match = of.ofp_match(dl_type =0x0806)
                flow.actions.append(of.ofp_action_output(port = 2))
                flow.hard_timeout = 50
                event.connection.send(flow)

        if packet.type == 2048:   # IPs

                ds_ip = packet.next.dstip
                src_ip = packet.next.srcip

                #Paquetes IPs hacia h7 con origen h1-3

                if((str(ds_ip))=='10.0.0.7'):
```

```
                                        msg = of.ofp_packet_out()
                                        msg.data = event.ofp
                                        msg.actions.append(of.ofp_action_output(port= 2))
                                        event.connection.send(msg)


                                        flow = of.ofp_flow_mod()
                                        flow.match = of.ofp_match(dl_type =0x0800, nw_dst =
IPAddr("10.0.0.7"))
                                        flow.actions.append(of.ofp_action_output(port = 2))
                                        flow.hard_timeout = 50
                                        event.connection.send(flow)

                                #Paquetes IPs hacia h1-3 con origen h7

                                if((str(src_ip))=='10.0.0.7'):

                                        msg = of.ofp_packet_out()
                                        msg.data = event.ofp
                                        msg.actions.append(of.ofp_action_output(port= 1))
                                        event.connection.send(msg)


                                        flow = of.ofp_flow_mod()
                                        flow.match = of.ofp_match(dl_type =0x0800, nw_src =
IPAddr("10.0.0.7"))
                                        flow.actions.append(of.ofp_action_output(port = 1))
                                        flow.hard_timeout = 50
                                        event.connection.send(flow)

        if(dpidToStr(event.dpid)=='00-00-00-00-00-04'): #---------------------------------- SW 4 ---
----------------------------------------------------------#

                if packet.type == 2054:   # ARPs
                        msg = of.ofp_packet_out()
                        msg.data = event.ofp
                        msg.actions.append(of.ofp_action_output(port =
                of.OFPP_FLOOD))
                        event.connection.send(msg)

                        flow = of.ofp_flow_mod()
                        flow.match = of.ofp_match(dl_type =0x0806)
                        flow.actions.append(of.ofp_action_output(port =
of.OFPP_FLOOD))
                        flow.hard_timeout = 50
                        event.connection.send(flow)

                if packet.type == 2048:   # IPs

                        ds_ip = packet.next.dstip
                        src_ip = packet.next.srcip

                        #Paquetes IPs hacia h7
                        if((str(ds_ip))=='10.0.0.7'):

                                msg = of.ofp_packet_out()
                                msg.data = event.ofp
                                msg.actions.append(of.ofp_action_output(port= 1))
                                event.connection.send(msg)

                                flow = of.ofp_flow_mod()
                                flow.match = of.ofp_match(dl_type =0x0800, nw_dst =
IPAddr("10.0.0.7"))
                                flow.actions.append(of.ofp_action_output(port = 1))
                                flow.hard_timeout = 50
                                event.connection.send(flow)

                        #Paquetes IPs con origen h7 hacia h1-3
```

```
                                        if((str(src_ip))=='10.0.0.7' and ((str(ds_ip))=='10.0.0.1' or
                        (str(ds_ip))=='10.0.0.2' or (str(ds_ip))=='10.0.0.3')):

                                                msg = of.ofp_packet_out()
                                                msg.data = event.ofp
                                                msg.actions.append(of.ofp_action_output(port= 2))
                                                event.connection.send(msg)

                                                for i in range (1,4):
                                                        flow = of.ofp_flow_mod()
                                                        flow.match = of.ofp_match(dl_type =0x0800,
                                        nw_src = IPAddr("10.0.0.7"),nw_dst = IPAddr("10.0.0.%s"%i))
                                                flow.actions.append(of.ofp_action_output(port = 2))
                                                        flow.hard_timeout = 50
                                                        event.connection.send(flow)

                                        #Paquetes IPs con orgien h7 hacia h4-6

                                        if((str(src_ip))=='10.0.0.7' and ((str(ds_ip))=='10.0.0.4' or
(str(ds_ip))=='10.0.0.5' or (str(ds_ip))=='10.0.0.6')):

                                                msg = of.ofp_packet_out()
                                                msg.data = event.ofp
                                                msg.actions.append(of.ofp_action_output(port= 3))
                                                event.connection.send(msg)

                                                for i in range (4,7):
                                                        flow = of.ofp_flow_mod()
                                                        flow.match = of.ofp_match(dl_type =0x0800,
                                        nw_src = IPAddr("10.0.0.7"),nw_dst = IPAddr("10.0.0.%s"%i))
                                                flow.actions.append(of.ofp_action_output(port = 3))
                                                        flow.hard_timeout = 50
                                                        event.connection.send(flow)


                        if(dpidToStr(event.dpid)=='00-00-00-00-00-05'): #---------------------------------- SW 5 ---
-------------------------------------------------------------#

                                        if packet.type == 2054:   # ARPs
                                                msg = of.ofp_packet_out()
                                                msg.data = event.ofp
                                                msg.actions.append(of.ofp_action_output(port = 2))
                                                event.connection.send(msg)

                                                flow = of.ofp_flow_mod()
                                                flow.match = of.ofp_match(dl_type =0x0806)
                                                flow.actions.append(of.ofp_action_output(port = 2))
                                                flow.hard_timeout = 50
                                                event.connection.send(flow)

                                        if packet.type == 2048:   # IPs

                                                ds_ip = packet.next.dstip
                                                src_ip = packet.next.srcip

                                        # Paquetes IPs hacia h7 con origen h1-3

                                                if((str(ds_ip))=='10.0.0.7'):

                                                        msg = of.ofp_packet_out()
                                                        msg.data = event.ofp
                                                        msg.actions.append(of.ofp_action_output(port= 1))
                                                        event.connection.send(msg)

                                                        flow = of.ofp_flow_mod()
                                                        flow.match = of.ofp_match(dl_type =0x0800, nw_dst =
                                IPAddr("10.0.0.7"))
                                                        flow.actions.append(of.ofp_action_output(port = 1))
```

```
                                        flow.hard_timeout = 50
                                        event.connection.send(flow)

                                #Paquetes IPs hacia h4-6 con origen h7

                                if((str(src_ip))=='10.0.0.7'):

                                        msg = of.ofp_packet_out()
                                        msg.data = event.ofp
                                        msg.actions.append(of.ofp_action_output(port= 2))
                                        event.connection.send(msg)

                                        flow = of.ofp_flow_mod()
                                        flow.match = of.ofp_match(dl_type =0x0800, nw_src =
IPAddr("10.0.0.7"))
                                        flow.actions.append(of.ofp_action_output(port = 2))
                                        flow.hard_timeout = 50
                                        event.connection.send(flow)



                        if(dpidToStr(event.dpid)=='00-00-00-00-00-06'): #---------------------------------- SW 6 ---
----------------------------------------------------------#

                                if packet.type == 2054:   # ARPs
                                        msg = of.ofp_packet_out()
                                        msg.data = event.ofp
                                        msg.actions.append(of.ofp_action_output(port = 1))
                                        event.connection.send(msg)

                                        flow = of.ofp_flow_mod()
                                        flow.match = of.ofp_match(dl_type =0x0806)
                                        flow.actions.append(of.ofp_action_output(port = 1))
                                        flow.hard_timeout = 50
                                        event.connection.send(flow)

                                if packet.type == 2048:   # IPs

                                        ds_ip = packet.next.dstip
                                        src_ip = packet.next.srcip

                                        #Paquetes IPs con destino h7 y origen h4-6

                                        if((str(ds_ip))=='10.0.0.7'):

                                                msg = of.ofp_packet_out()
                                                msg.data = event.ofp
                                                msg.actions.append(of.ofp_action_output(port= 2))
                                                event.connection.send(msg)

                                                flow = of.ofp_flow_mod()
                                                flow.match = of.ofp_match(dl_type =0x0800, nw_dst =
IPAddr("10.0.0.7"))
                                                flow.actions.append(of.ofp_action_output(port = 2))
                                                flow.hard_timeout = 50
                                                event.connection.send(flow)

                                        #Paquetes IPs con destino h4-6 y origen h7

                                                if((str(src_ip))=='10.0.0.7'):

                                                        msg = of.ofp_packet_out()
                                                        msg.data = event.ofp
                                                        msg.actions.append(of.ofp_action_output(port= 1))
                                                        event.connection.send(msg)

                                                        flow = of.ofp_flow_mod()
```

```
                                        flow.match = of.ofp_match(dl_type =0x0800, nw_src =
                                IPAddr("10.0.0.7"))
                                        flow.actions.append(of.ofp_action_output(port = 1))
                                        flow.hard_timeout = 50
                                        event.connection.send(flow)


def launch ():
        core.openflow.addListenerByName("PacketIn", _handle_PacketIn)
```

BCenlaces_plantilla:

```
# Complemente con su código en los espacios:

        # .
        # ..
        # ...


#/usr/bin/python
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpidToStr
import pox.lib.packet as pkt
from pox.lib.addresses import EthAddr, IPAddr
from pox.lib.revent import *
from pox.lib.util import dpid_to_str
from pox.lib.packet.ethernet import ethernet
from pox.lib.packet.ipv4 import ipv4


log = core.getLogger()


def _handle_PacketIn (event):

        packet = event.parsed   # Se usa para obtener los datos del paquete analizado.
pueden usarse para diversos propósitos, como extraer direcciones MAC de origen y destino,
direcciones IP u otros campos


        if event.parsed.type != 34525: # Distingue de paquetes IPV6

                # Extrae la dirección Mac del Switch
                if(dpidToStr(event.dpid)=='00-00-00-00-00-01'): #---------------------------------- SW 1 ----------------
-------------------------------------------#


                        if packet.type == 2054: #ARPs

                                # se le envia la accion a ejecutar
                                # Crea un mensaje OFPT_PACKET_OUT, que se utiliza para enviar un
                        paquete desde el controlador hacia un switch OpenFlow
                                msg = of.ofp_packet_out()

                                # Establece los datos del mensaje como los datos del paquete
                        recibido
                                msg.data = event.ofp

                                # Establecer la accion, en este caso, el puerto de salida del switch
                                msg.actions.append(of.ofp_action_output(port =of.OFPP_FLOOD))

                                # Envía el mensaje al switch OpenFlow para su procesamiento
                                event.connection.send(msg)

                                # Instala el flujo temporalmente
```

```
                flow = of.ofp_flow_mod()
                flow.match = of.ofp_match(dl_type =0x0806)
                flow.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
                # Instala el flujo temporalmente
                flow.hard_timeout = 50
                event.connection.send(flow)



        if packet.type == 2048: # IPs

                ds_ip = packet.next.dstip # Se obtiene la direccion IP destino del
        paquete   capturado

                src_ip = packet.next.srcip # Se obtiene la direccion IP origen del
        paquete   capturado

                # Paquetes IPs hacia h1
                if((str(ds_ip))=='10.0.0.1'):

                        msg = of.ofp_packet_out()
                        msg.data = event.ofp
                        msg.actions.append(of.ofp_action_output(port= 1))
                        event.connection.send(msg)

                        # Instala el flujo temporalmente
                        flow = of.ofp_flow_mod()
                        flow.match = of.ofp_match(dl_type =0x0800, nw_dst =
IPAddr("10.0.0.1"))
                        flow.actions.append(of.ofp_action_output(port = 1))
                        flow.hard_timeout = 50
                        event.connection.send(flow)

                #Paquetes IPs hacia h2
                if((str(ds_ip))=='10.0.0.2'):

                        # .
                        # ..
                        # ...

                        # Instala el flujo temporalmente

                        # .
                        # ..
                        # ...

                #Paquetes IPs hacia h3

                        # .
                        # ..
                        # ...

                        # Instala el flujo temporalmente

                        # .
                        # ..
                        # ...

                #Paquetes IPs hacia h4

                        # .
                        # ..
                        # ...

                        # Instala el flujo temporalmente

                        # .
                        # ..
```

```
                                        # ...

                                    #Paquetes IPs hacia h5

                                        # .
                                        # ..
                                        # ...

                                        # Instala el flujo temporalmente

                                        # .
                                        # ..
                                        # ...

                                    #Paquetes IPs hacia h6

                                        # .
                                        # ..
                                        # ...

                                        # Instala el flujo temporalmente

                                        # .
                                        # ..
                                        # ...

                                    #Paquetes IPs hacia h7 con origen h1-3

                                    if((str(ds_ip))=='10.0.0.7' and ((str(src_ip))=='10.0.0.1' or
(str(src_ip))=='10.0.0.2' or (str(src_ip))=='10.0.0.3')):

                                        msg = of.ofp_packet_out()
                                        msg.data = event.ofp
                                        msg.actions.append(of.ofp_action_output(port= 7))
                                        event.connection.send(msg)

                                        for i in range (1,4):
                                            flow = of.ofp_flow_mod()
                                            flow.match = of.ofp_match(dl_type =0x0800, nw_src =
                    IPAddr("10.0.0.%s"%i),nw_dst = IPAddr("10.0.0.7"))
                flow.actions.append(of.ofp_action_output(port = 7))
                                            flow.hard_timeout = 50
                                            event.connection.send(flow)

                                    #Paquetes IPs hacia h7 con origen h4-6

                                        # .
                                        # ..
                                        # ...

                                        # Instala el flujo temporalmente

                                        # .
                                        # ..
                                        # ...

                    if(dpidToStr(event.dpid)=='00-00-00-00-00-02'): #--------------------------------- SW 2 ---------------
-------------------------------------------#

                        if packet.type == 2054:   # ARPs

                            # .
                            # ..
                            # ...

                            # Instala el flujo temporalmente

                            # .
                            # ..
```

```
                # ...

            if packet.type == 2048:   # IPs

                ds_ip = packet.next.dstip
                src_ip = packet.next.srcip

                # Paquetes IPs hacia h7 con origen h1-3

                if((str(ds_ip))=='10.0.0.7'):

                    msg = of.ofp_packet_out()
                    msg.data = event.ofp
                    msg.actions.append(of.ofp_action_output(port= 2))
                    event.connection.send(msg)


                    flow = of.ofp_flow_mod()
                    flow.match = of.ofp_match(dl_type =0x0800,nw_dst =
IPAddr("10.0.0.7"))
                    flow.actions.append(of.ofp_action_output(port = 2))
                    flow.hard_timeout = 50
                    event.connection.send(flow)


                #Paquetes IPs hacia h1-3 con origen h7

                if((str(src_ip))=='10.0.0.7'):

                    msg = of.ofp_packet_out()
                    msg.data = event.ofp
                    msg.actions.append(of.ofp_action_output(port= 1))
                    event.connection.send(msg)


                    flow = of.ofp_flow_mod()
                    flow.match = of.ofp_match(dl_type =0x0800, nw_src =
IPAddr("10.0.0.7"))
                    flow.actions.append(of.ofp_action_output(port = 1))
                    flow.hard_timeout = 50
                    event.connection.send(flow)


            if(dpidToStr(event.dpid)=='00-00-00-00-00-03'): #---------------------------------- SW 3 ----------------
----------------------------------------#

                # .
                # ..
                # ...
    z

            if(dpidToStr(event.dpid)=='00-00-00-00-00-04'): #---------------------------------- SW 4 ----------------
----------------------------------------#

                if packet.type == 2054:   # ARPs


                    # .
                    # ..
                    # ...
    z
                if packet.type == 2048:   # IPs


                    # .
                    # ..
                    # ...
    z
```

```
		if(dpidToStr(event.dpid)=='00-00-00-00-00-05'): #---------------------------------- SW 5 ----------
----------------------------------------#

				# .
				# ..
				# ...

	z

		if(dpidToStr(event.dpid)=='00-00-00-00-00-06'): #---------------------------------- SW 6 ----------
	----------------------------------------#

				# .
				# ..
				# ...
	z

def launch ():
	core.openflow.addListenerByName("PacketIn", _handle_PacketIn)
```

BCenlaces por protocolo:

```
#/usr/bin/python
from datetime import datetime
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpidToStr
import pox.lib.packet as pkt
from pox.lib.addresses import EthAddr, IPAddr
from pox.lib.revent import *
from pox.lib.util import dpid_to_str
from pox.lib.packet.ethernet import ethernet
from pox.lib.packet.ipv4 import ipv4


log = core.getLogger()


def _handle_PacketIn (event):


	packet = event.parsed
	mac_destino= packet.dst

	if event.parsed.type != 34525:

		if(dpidToStr(event.dpid)=='00-00-00-00-00-01'): #---------------------------------- SW 1 ----------
--------------------------------#

			if packet.type == 2054: #ARPs

				# se le envia la accion a ejecutar

				msg = of.ofp_packet_out()
				msg.data = event.ofp
				msg.actions.append(of.ofp_action_output(port =of.OFPP_FLOOD))
				event.connection.send(msg)

				# Instala el flujo temporalmente

				flow = of.ofp_flow_mod()
				flow.match = of.ofp_match(dl_type =0x0806)
				flow.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
				flow.hard_timeout = 50
				event.connection.send(flow)

			if packet.type == 2048: # IPs

				ds_ip = packet.next.dstip
```

```
                        src_ip = packet.next.srcip

                        #Paquetes IPs hacia h1
                        if((str(ds_ip))=='10.0.0.1'):

                                msg = of.ofp_packet_out()
                                msg.data = event.ofp
                                msg.actions.append(of.ofp_action_output(port= 1))
                                event.connection.send(msg)

                                # Instala el flujo temporalmente
                                flow = of.ofp_flow_mod()
                                flow.match = of.ofp_match(dl_type =0x0800, nw_dst =
IPAddr("10.0.0.1"))
                                flow.actions.append(of.ofp_action_output(port = 1))
                                flow.hard_timeout = 50
                                event.connection.send(flow)

                        #Paquetes IPs hacia h2
                        if((str(ds_ip))=='10.0.0.2'):

                                msg = of.ofp_packet_out()
                                msg.data = event.ofp
                                msg.actions.append(of.ofp_action_output(port= 2))
                                event.connection.send(msg)

                                # Instala el flujo temporalmente
                                flow = of.ofp_flow_mod()
                                flow.match = of.ofp_match(dl_type =0x0800, nw_dst =
        IPAddr("10.0.0.2"))
                                flow.actions.append(of.ofp_action_output(port = 2))
                                flow.hard_timeout = 50
                                event.connection.send(flow)

                        #Paquetes IPs hacia h3
                        if((str(ds_ip))=='10.0.0.3'):

                                msg = of.ofp_packet_out()
                                msg.data = event.ofp
                                msg.actions.append(of.ofp_action_output(port= 3))
                                event.connection.send(msg)

                                # Instala el flujo temporalmente
                                flow = of.ofp_flow_mod()
                                flow.match = of.ofp_match(dl_type =0x0800, nw_dst =
        IPAddr("10.0.0.3"))
                                flow.actions.append(of.ofp_action_output(port = 3))
                                flow.hard_timeout = 50
                                event.connection.send(flow)

                        #Paquetes IPs hacia h4
                        if((str(ds_ip))=='10.0.0.4'):

                                msg = of.ofp_packet_out()
                                msg.data = event.ofp
                                msg.actions.append(of.ofp_action_output(port= 4))
                                event.connection.send(msg)

                                # Instala el flujo temporalmente
                                flow = of.ofp_flow_mod()
                                flow.match = of.ofp_match(dl_type =0x0800, nw_dst =
        IPAddr("10.0.0.4"))
                                flow.actions.append(of.ofp_action_output(port = 4))
                                flow.hard_timeout = 50
                                event.connection.send(flow)

                        #Paquetes IPs hacia h5
                        if((str(ds_ip))=='10.0.0.5'):
```

```
                                        msg = of.ofp_packet_out()
                                        msg.data = event.ofp
                                        msg.actions.append(of.ofp_action_output(port= 5))
                                        event.connection.send(msg)

                                        # Instala el flujo temporalmente
                                        flow = of.ofp_flow_mod()
                                        flow.match = of.ofp_match(dl_type =0x0800, nw_dst =
                        IPAddr("10.0.0.5"))
                                        flow.actions.append(of.ofp_action_output(port = 5))
                                        flow.hard_timeout = 50
                                        event.connection.send(flow)

                                #Paquetes IPs hacia h6
                                if((str(ds_ip))=='10.0.0.6'):

                                        msg = of.ofp_packet_out()
                                        msg.data = event.ofp
                                        msg.actions.append(of.ofp_action_output(port= 6))
                                        event.connection.send(msg)

                                        # Instala el flujo temporalmente
                                        flow = of.ofp_flow_mod()
                                        flow.match = of.ofp_match(dl_type =0x0800, nw_dst =
                        IPAddr("10.0.0.6"))
                                        flow.actions.append(of.ofp_action_output(port = 6))
                                        flow.hard_timeout = 50
                                        event.connection.send(flow)

                                #Paquetes IPs hacia h7
                                if((str(ds_ip))=='10.0.0.7'):

                                        msg = of.ofp_packet_out()
                                        msg.data = event.ofp
                                        msg.actions.append(of.ofp_action_output(port= 8))
                                        event.connection.send(msg)

                                        # Instala el flujo temporalmente
                                        flow = of.ofp_flow_mod()
                                        flow.match = of.ofp_match(dl_type =0x0800, nw_dst =
                        IPAddr("10.0.0.7"))
                                        flow.actions.append(of.ofp_action_output(port = 8))
                                        flow.hard_timeout = 50
                                        event.connection.send(flow)

                        if(dpidToStr(event.dpid)=='00-00-00-00-00-02'): #---------------------------------- SW 2 ----------------
--------------------------------#

                                if packet.type == 2054: # ARPs

                                        # se le envia la accion a ejecutar
                                        msg = of.ofp_packet_out()
                                        msg.data = event.ofp
                                        msg.actions.append(of.ofp_action_output(port =of.OFPP_FLOOD))
                                        event.connection.send(msg)

                                        # Instala el flujo temporalmente
                                        flow = of.ofp_flow_mod()
                                        flow.match = of.ofp_match(dl_type =0x0806)
                                        flow.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
                                        flow.hard_timeout = 50
                                        event.connection.send(flow)

                        if(dpidToStr(event.dpid)=='00-00-00-00-00-03'): #---------------------------------- SW 3 ----------------
--------------------------------#

                                if packet.type == 2054: # ARPs

                                        # se le envia la accion a ejecutar
```

```
                                msg = of.ofp_packet_out()
                                msg.data = event.ofp
                                msg.actions.append(of.ofp_action_output(port =of.OFPP_FLOOD))
                                event.connection.send(msg)

                                # Instala el flujo temporalmente
                                flow = of.ofp_flow_mod()
                                flow.match = of.ofp_match(dl_type =0x0806)
                                flow.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
                                flow.hard_timeout = 50
                                event.connection.send(flow)

        if(dpidToStr(event.dpid)=='00-00-00-00-00-04'): #---------------------------------- SW 4 ----------------
------------------------------#

                if packet.type == 2054: #ARPs

                                # se le envia la accion a ejecutar
                                msg = of.ofp_packet_out()
                                msg.data = event.ofp
                                msg.actions.append(of.ofp_action_output(port =of.OFPP_FLOOD))
                                event.connection.send(msg)

                                # Instala el flujo temporalmente

                                flow = of.ofp_flow_mod()
                                flow.match = of.ofp_match(dl_type =0x0806)
                                flow.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
                                flow.hard_timeout = 50
                                event.connection.send(flow)

                if packet.type == 2048: # IPs

                                ds_ip = packet.next.dstip
                                src_ip = packet.next.srcip

                                #Paquetes IPs hacia h1
                                if((str(ds_ip))=='10.0.0.7'):

                                                msg = of.ofp_packet_out()
                                                msg.data = event.ofp
                                                msg.actions.append(of.ofp_action_output(port= 1))
                                                event.connection.send(msg)

                                                # Instala el flujo temporalmente
                                                flow = of.ofp_flow_mod()
                                                flow.match = of.ofp_match(dl_type =0x0800, nw_dst =
IPAddr("10.0.0.7"))
                                                flow.actions.append(of.ofp_action_output(port = 1))
                                                flow.hard_timeout = 50
                                                event.connection.send(flow)

                                if((str(ds_ip))!='10.0.0.7'):

                                                msg = of.ofp_packet_out()
                                                msg.data = event.ofp
                                                msg.actions.append(of.ofp_action_output(port= 3))
                                                event.connection.send(msg)

                                                # Instala el flujo temporalmente
                                                flow = of.ofp_flow_mod()
                                                flow.match = of.ofp_match(dl_type =0x0800, nw_src =
IPAddr("10.0.0.7"))
                                                flow.actions.append(of.ofp_action_output(port = 3))
                                                flow.hard_timeout = 50
                                                event.connection.send(flow)

        if(dpidToStr(event.dpid)=='00-00-00-00-00-05'): #---------------------------------- SW 5 ----------------
------------------------------#
```

```
                if packet.type == 2048: #IPs

                        ds_ip = packet.next.dstip
                        src_ip = packet.next.srcip

                        if((str(ds_ip))=='10.0.0.7'):

                                msg = of.ofp_packet_out()
                                msg.data = event.ofp
                                msg.actions.append(of.ofp_action_output(port= 1))
                                event.connection.send(msg)

                                # Instala el flujo temporalmente
                                flow = of.ofp_flow_mod()
                                flow.match = of.ofp_match(dl_type =0x0800, nw_dst =
        IPAddr("10.0.0.7"))
                                flow.actions.append(of.ofp_action_output(port = 1))
        flow.hard_timeout = 50
                                event.connection.send(flow)

                        if((str(ds_ip))!='10.0.0.7'):

                                msg = of.ofp_packet_out()
                                msg.data = event.ofp
                                msg.actions.append(of.ofp_action_output(port= 2))
                                event.connection.send(msg)

                                # Instala el flujo temporalmente
                                flow = of.ofp_flow_mod()
                                flow.match = of.ofp_match(dl_type =0x0800, nw_src =
        IPAddr("10.0.0.7"))
                                flow.actions.append(of.ofp_action_output(port = 2))
                                flow.hard_timeout = 50
                                event.connection.send(flow)

        if(dpidToStr(event.dpid)=='00-00-00-00-00-06'): #---------------------------------- SW 6 ----------------
--------------------------------#

                if packet.type == 2048: #IPs

                        ds_ip = packet.next.dstip
                        src_ip = packet.next.srcip

                        if((str(ds_ip))=='10.0.0.7'):

                                msg = of.ofp_packet_out()
                                msg.data = event.ofp
                                msg.actions.append(of.ofp_action_output(port= 2))
                                event.connection.send(msg)

                                # Instala el flujo temporalmente
                                flow = of.ofp_flow_mod()
                                flow.match = of.ofp_match(dl_type =0x0800, nw_dst =
        IPAddr("10.0.0.7"))
                                flow.actions.append(of.ofp_action_output(port = 2))
                                flow.hard_timeout = 50
                                event.connection.send(flow)

                        if((str(ds_ip))!='10.0.0.7'):

                                msg = of.ofp_packet_out()
                                msg.data = event.ofp
                                msg.actions.append(of.ofp_action_output(port= 1))
                                event.connection.send(msg)

                                # Instala el flujo temporalmente
                                flow = of.ofp_flow_mod()
```

```
                                        flow.match = of.ofp_match(dl_type =0x0800, nw_src =
                IPAddr("10.0.0.7"))
                                        flow.actions.append(of.ofp_action_output(port = 1))
                                        flow.hard_timeout = 50
                                        event.connection.send(flow)

def launch ():
        core.openflow.addListenerByName("PacketIn", _handle_PacketIn)
```

A continuación, se presentan los anexos para la práctica de VLANs:

1. **Guía:** Contiene una breve explicación de los conceptos clave para desarrollar la práctica, el enunciado de esta y algunas ayudas.
2. **Topología y ejecutor de pruebas:** Script que utilizara el estudiante para generar la topología de esta práctica y poner a prueba su código
3. **VLAN Practivo:** Código que da solución a la problemática planteada para el desarrollo de la práctica.
4. **VLAN Proactivo plantilla:** Esquema del código, que se entregara como base al estudiante para que este lo complete y desarrolle la práctica.
5. **VLAN Reactivo:** Código solución a uno de los puntos extra planteados en la guía de la práctica.

Guía:

Las VLAN, o Redes de Área Local Virtuales por sus siglas en inglés (Virtual Local Area Networks), son una técnica de segmentación de redes que permite dividir una red física en múltiples segmentos lógicos o subredes, aunque estén conectados al mismo switch o red física. Es una forma de virtualización que facilita la utilización de recursos físicos por diferentes usuarios, aislándolos entre sí. Para comunicar nodos de diferentes VLAN se necesita un enrutador.

Las VLAN permiten la administración flexible de recursos de red al agrupar dispositivos lógicamente en lugar de depender de la ubicación física. Esto es especialmente útil en entornos donde los usuarios o dispositivos se mueven con frecuencia. También, al aislar grupos de dispositivos en VLAN separadas, se reduce la cantidad de equipos vulnerables ante un ataque. Las políticas de seguridad pueden aplicarse a nivel de VLAN para controlar el tráfico entre segmentos lógicos. También facilitan la optimización del ancho de banda al limitar los broadcasts únicamente a los dispositivos en la misma VLAN. Esto ayuda a reducir la congestión y mejorar el rendimiento general de la red.

Las VLAN se pueden extender a través de diferentes switches utilizando enlaces que permiten tráfico con etiquetas que identifican la pertenencia de un paquete a una VLAN específica. Una de las especificaciones más usadas para etiquetar el tráfico es la conocida IEEE 802.1q. Este define la inclusión de un encabezado que contiene información de la VLAN de origen del paquete.

En esta práctica con la ayuda del controlador POX se explorará una solución para implementar VLAN en una red. Para ello se hará uso de los encabezados del protocolo 802.1Q. Estos encabezados se insertan a cada paquete perteneciente a una VLAN y ayudan a distinguirlos y dirigirlos correctamente.

Actividad

El objetivo consiste en instalar flujos en los switches S1 y S2, a través del controlador POX, que habiliten el tráfico VLAN mediante el protocolo 802.1Q. De esta manera, los hosts **h1** y **h3** pertenecientes a la VLAN A estarán aislados de los hosts **h2** y **h4** pertenecientes a la VLAN B. Para comprobar el correcto funcionamiento se ejecutará un ping global que demuestre los requisitos pedidos.

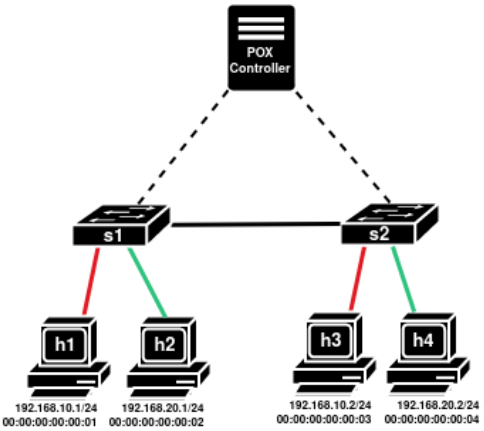

Figura 1: *Topología: Rojo VLAN A, Verde: VLAN B*

Una de las ventajas de solucionar este escenario con SDN es, por ejemplo, el hecho de que no es necesario asignar puertos troncales en los switches. Simplemente se debe indicar el camino por el que se desee que viajen los paquetes, asignándoles un ID que indica a qué VLAN pertenece.

Las funciones y clases sugeridas para el desarrollo de esta práctica son las siguientes:

- Estas indican que a los paquetes que lleguen por el puerto 3 se les asigne un VLAN ID. En este caso, por el puerto 2 hay un host conectado perteneciente a la VLAN 20.

```
flow.match = of.ofp_match(in_port = 2)
flow.actions.append(of.ofp_action_vlan_vid(vlan_vid = 20))
```

- De esta manera se indica al switch que elimine el encabezado 802.1q de los paquetes que llegan por el puerto 1 con VLAN 10 y los dirige al puerto 3.

```
flow.match = of.ofp_match(in_port = 1, dl_vlan = 10)
flow.actions.append(of.ofp_action_strip_vlan ())
flow.actions.append(of.ofp_action_output(port = 3))
```

Escenario de pruebas:

De esta manera se indica al switch que elimine el encabezado 802.1q de los paquetes que llegan por el puerto 1 con VLAN 10 y los dirija al puerto 3:

1. **Línea de comandos Mininet:** Abre la interfaz de Mininet.
2. **Conectividad total (pingall):** Realiza la operación pingall, que hace un ping global.
3. **Flujos:** Muestra los flujos que estén instalados.
4. **Abrir terminales:** Abre las terminales de los equipos pertenecientes a la topología. Para esta función se recomienda tener instalado Xterm.
5. **Ver flujos en x SW:** Muestra los flujos del switch deseado.
6. **Ping entre dos hosts:** Realiza una prueba de conexión entre dos hosts específicos.
7. **IPERF entre dos hosts:** Comprueba la tasa de transferencia entre dos hosts.
8. **Servidor web:** Genera un escenario cliente-servidor con dos hosts de la red.
9. **SALIR:** Cierra el script y ejecuta el comando `mn -c` que elimina todos los procesos y contenedores relacionados con la red creada en Mininet.

Tip: Se recomienda dividir la consola de esta manera para mayor comodidad a la hora de realizar las pruebas.

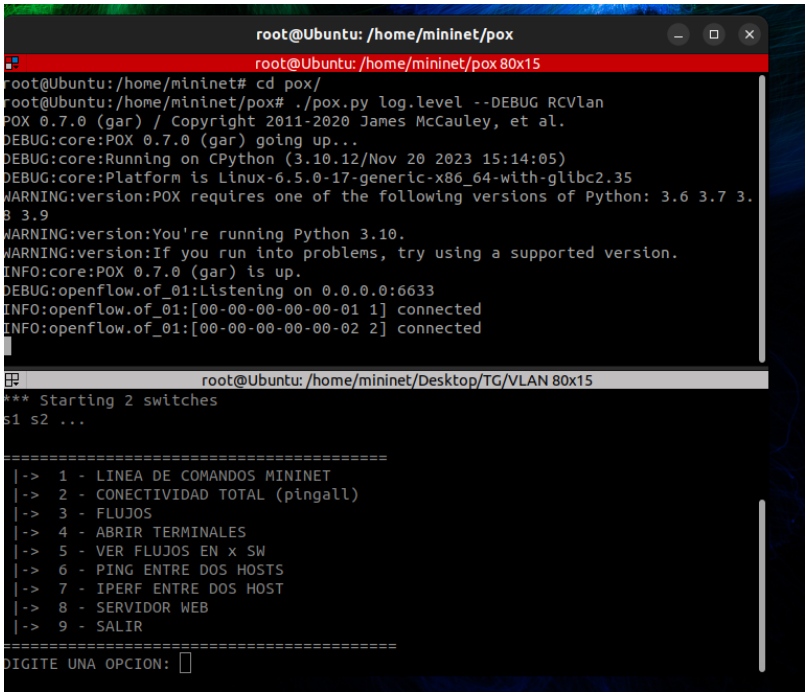

Figura 2: *Prueba en ejecución, practica VLAN.*

Extra:

1. Agregue un switch S3 entre S1 y S2 y cree una nueva VLAN.
2. Realice el código para un controlador POX reactivo.

Tips de ayuda para el punto 2:

- La siguiente línea de código sirve para identificar tráfico etiquetado con encabezados del protocolo 802.1Q:

```
if packet.type == ethernet.VLAN_TYPE: # Si es un paquete VLAN
```

- De esta manera se accede a la etiqueta (Id) del encabezado de un paquete capturado:

```
packet.next.id
```

Topología y ejecutor de pruebas:

```
#!/usr/bin/python
from mininet.net import Mininet
from mininet.cli import CLI
from mininet.node import OVSSwitch, RemoteController
from mininet.topo import Topo, SingleSwitchTopo
from mininet.log import setLogLevel, info
from mininet.util import dumpNodeConnections
from mininet.link import TCLink
from mininet.term import makeTerm
from time import sleep

import os

import sys
def startNetwork():
        net = Mininet(switch=OVSSwitch)

        # Vlan 10: h1 y h3
        # Vlan 20: h2 y h4

        h1=net.addHost('h1', mac='00:00:00:00:10:01', ip='10.0.0.1/24')
        h2=net.addHost('h2', mac='00:00:00:00:10:02', ip='10.0.0.2/24')
```

```
        h3=net.addHost('h3', mac='00:00:00:00:20:03', ip='10.0.0.3/24')
        h4=net.addHost('h4', mac='00:00:00:00:20:04', ip='10.0.0.4/24')

        #h1=net.addHost('h1', mac='00:00:00:00:10:01', ip='192.168.10.1/24')
        #h2=net.addHost('h2', mac='00:00:00:00:10:02', ip='192.168.20.1/24')
        #h3=net.addHost('h3', mac='00:00:00:00:20:03', ip='192.168.10.2/24')
        #h4=net.addHost('h4', mac='00:00:00:00:20:04', ip='192.168.20.2/24')

        # SW
        net.addSwitch('s1', mac='00-00-00-00-00-01')
        net.addSwitch('s2', mac='00-00-00-00-00-02')

        net.addLink('h1','s1',0,1) # net.addLink(Node1,Node2,portNode1,portNode2)
        net.addLink('h2','s1',0,2)
        net.addLink('s1','s2',3,1)
        net.addLink('h3','s2',0,3)
        net.addLink('h4','s2',0,4)

        net.addController('c0',controller=RemoteController, ip='127.0.0.1', port=6633)

        net.start()

        option=0
        while option !=1:
                info('\n=========================================')
                info('\n |->  1 - LINEA DE COMANDOS MININET')
                info('\n |->  2 - CONECTIVIDAD TOTAL (pingall)')
                info('\n |->  3 - FLUJOS')
                info('\n |->  4 - ABRIR TERMINALES')
                info('\n |->  5 - VER FLUJOS EN x SW')
                info('\n |->  6 - PING ENTRE DOS HOSTS')
                info('\n |->  7 - IPERF ENTRE DOS HOST')
                info('\n |->  8 - SERVIDOR WEB')
                info('\n |->  9 - SALIR')

        info('\n=========================================\n')
        opt =input("DIGITE UNA OPCION: ")
                if opt.isdigit():
                        option =int(opt)
        #================================================
                if option ==2:
                        net.pingAll()
                option =0
        #================================================
                elif option ==3:
                        for sw in net.switches:
                                info ("%s "% sw.name)
                                os.system ("sudo ovs-ofctl dump-flows %s "% sw.name)

        info('\n=========================================================================
==========================================================================\n')
                option =0

                elif option ==4:
                        net.startTerms()
                option =0

                elif option ==5:
                        info("CUAL SW ?\r\n")
                        s = input("SWITCH: ")
                        sw = net.get(s)
                        os.system("sudo ovs-ofctl dump-flows %s" % sw.name)

                elif option ==6:
                        info("PING ENTRE DOS HOST \r\n")
                        h_ping_1 = input("PRIMER HOST: ")
                        h_ping_2 = input("SEGUNDO HOST: ")
                        host1,host2 = net.get(h_ping_1,h_ping_2)
                        net.ping((host1,host2))
```

```
                elif option ==7:

                        info("**IPERF entre dos hosts** \r\n")
                        host_1_iperf = input("host1: ")
                        host_2_iperf = input("host2: ")
                        host1,host2 = net.get(host_1_iperf,host_2_iperf)
                        net.iperf((host1,host2),l4Type='TCP')

                elif option ==8:

                        info("**Servidor WEB en un host** \r\n")
                        server_web = input("server WEB: ")
                        cliente_web = input("cliente WEB: ")
                        s_net_web = net.get(server_web)
                        c_net_web = net.get(cliente_web)
                        makeTerm(s_net_web, title='Server WEB', term='xterm', cmd="bash -c 'exec
        sudo python2 -m SimpleHTTPServer 80;"')
        makeTerm(c_net_web)

                elif option ==9:
                        net.stop()
                sys.exit()

        CLI ( net )
#////////////////////////////////////////////////////////
if __name__ =='__main__':
        setLogLevel('info')
        startNetwork()
```

VLAN Proactivo plantilla:

```
#!/usr/bin/python
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpidToStr
from pox.lib.addresses import EthAddr, IPAddr

log = core.getLogger()

def _handle_ConnectionUp (event):

  if dpidToStr(event.dpid)=='00-00-00-00-00-01': ##---------------------- SW 1 -------------------------------

        # Paquetes de h1 vlan10 a h3 vlan10

        flow = of.ofp_flow_mod()
        flow.match = of.ofp_match(in_port = 1)
        # Asigna un Id a la VLAN en cuestión, en este caso 10
        flow.actions.append(of.ofp_action_vlan_vid(vlan_vid = 10))
        flow.actions.append(of.ofp_action_output(port = 3))
        event.connection.send(flow)

        # Paquetes de h2 vlan20 a h4 vlan20

        flow = of.ofp_flow_mod()
        flow.match = of.ofp_match(in_port = 2)
        # Asigna un Id a la VLAN en cuestión, en este caso 20
        flow.actions.append(of.ofp_action_vlan_vid(vlan_vid = 20))
        flow.actions.append(of.ofp_action_output(port = 3))
        event.connection.send(flow)

        # Paquetes de h3 vlan10 a h1 vlan10

        flow = of.ofp_flow_mod()
        flow.match = of.ofp_match(in_port = 3, dl_vlan = 10)
        # se indica al switch que elimine el encabezado 802.1q
        flow.actions.append(of.ofp_action_strip_vlan ())
        flow.actions.append(of.ofp_action_output(port = 1))
```

```
        event.connection.send(flow)

        # Paquetes de h4 vlan20 a h2 vlan20
        flow = of.ofp_flow_mod()
        flow.match = of.ofp_match(in_port = 3, dl_vlan = 20)
        # indica al switch que elimine el encabezado 802.1q
        flow.actions.append(of.ofp_action_strip_vlan ())
        flow.actions.append(of.ofp_action_output(port = 2))
        event.connection.send(flow)


 elif dpidToStr(event.dpid)=='00-00-00-00-00-02':   ##---------------------- SW 2 ------------------------------| |

        # Paquetes de h3 VLAN 10 a h1 VLAN 10

        flow = of.ofp_flow_mod()
        flow.match = of.ofp_match(in_port = 3)
        # Asigna un Id a la VLAN en cuestión, en este caso 10
        flow.actions.append(of.ofp_action_vlan_vid(vlan_vid = 10))
        flow.actions.append(of.ofp_action_output(port = 1))
        event.connection.send(flow)

        # Paquetes de h4 VLAN 20 a h2 VLAN 20

        flow = of.ofp_flow_mod()
        flow.match = of.ofp_match(in_port = 4)
        # Asigna un Id a la VLAN en cuestión, en este caso 20
        flow.actions.append(of.ofp_action_vlan_vid(vlan_vid = 20))
        flow.actions.append(of.ofp_action_output(port = 1))
        event.connection.send(flow)

        # Paquetes de h1 VLAN 10 a h3 VLAN 10

        flow = of.ofp_flow_mod()
        flow.match = of.ofp_match(in_port = 1, dl_vlan = 10)
        # se indica al switch que elimine el encabezado 802.1q
        flow.actions.append(of.ofp_action_strip_vlan ())
        flow.actions.append(of.ofp_action_output(port = 3))
        event.connection.send(flow)

        # Paquetes de h2 VLAN 20 a h4 VLAN 20

        flow = of.ofp_flow_mod()
        flow.match = of.ofp_match(in_port = 1, dl_vlan = 20)
        # se indica al switch que elimine el encabezado 802.1q
        flow.actions.append(of.ofp_action_strip_vlan ())
        flow.actions.append(of.ofp_action_output(port = 4))
        event.connection.send(flow)


def launch ():
        core.openflow.addListenerByName("ConnectionUp", _handle_ConnectionUp)
        log.info("Proactivecontroller is running")
```

VLAN Proactivo plantilla:

```
# Complemente con su código en los espacios:

        # .
        # ..
        # ...


#!/usr/bin/python
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpidToStr
from pox.lib.addresses import EthAddr, IPAddr
```

```
log = core.getLogger()

def _handle_ConnectionUp (event):

  if dpidToStr(event.dpid)=='00-00-00-00-00-01': ##---------------------- SW 1 -------------------------------

        # Paquetes de h1 vlan10 a h3 vlan10

        flow = of.ofp_flow_mod()
        flow.match = of.ofp_match(in_port = 1)
        # Asigna un Id a la VLAN en cuestión, en este caso 10
        flow.actions.append(of.ofp_action_vlan_vid(vlan_vid = 10))
        flow.actions.append(of.ofp_action_output(port = 3))
        event.connection.send(flow)

        # Paquetes de h2 vlan20 a h4 vlan20

        # .
        # ..
        # ...


        # Paquetes de h3 vlan10 a h1 vlan10

        flow = of.ofp_flow_mod()
        flow.match = of.ofp_match(in_port = 3, dl_vlan = 10)
        # se indica al switch que elimine el encabezado 802.1q
        flow.actions.append(of.ofp_action_strip_vlan ())
        flow.actions.append(of.ofp_action_output(port = 1))
        event.connection.send(flow)

        # Paquetes de h4 vlan20 a h2 vlan20
        # .
        # ..
        # ...


  elif dpidToStr(event.dpid)=='00-00-00-00-00-02':   ##---------------------- SW 2 -------------------------------| |

        # Paquetes de h3 VLAN 10 a h1 VLAN 10

                # .
                # ..
                # ...


        # Paquetes de h4 VLAN 20 a h2 VLAN 20

                # .
                # ..
                # ...

        # Paquetes de h1 VLAN 10 a h3 VLAN 10

                # .
                # ..
                # ...

        # Paquetes de h2 VLAN 20 a h4 VLAN 20

                # .
                # ..
                # ...


def launch ():
        core.openflow.addListenerByName("ConnectionUp", _handle_ConnectionUp)
        log.info("Proactivecontroller is running")
```

VLAN Reactivo:

```
#/usr/bin/python
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpidToStr
import pox.lib.packet as pkt
from pox.lib.addresses import EthAddr, IPAddr
from pox.lib.revent import *
from pox.lib.util import dpid_to_str
from pox.lib.packet.ethernet import ethernet
from pox.lib.packet.ipv4 import ipv4


log = core.getLogger()


def _handle_PacketIn (event):

        packet = event.parsed
        mac_destino= packet.dst

        if event.parsed.type != 34525: # Distingue de paquetes IPV6

                if(dpidToStr(event.dpid)=='00-00-00-00-00-01'): #---------------------------------- SW 1 ----------------
-------------------------------------------#


                        if event.port == 1:

                                # se le envia la accion a ejecutar
                                msg = of.ofp_packet_out()
                                msg.data = event.ofp
                                # se le asigna un Id a la VLAN en cuestión en este caso 10
                                msg.actions.append(of.ofp_action_vlan_vid(vlan_vid = 10))
                                msg.actions.append(of.ofp_action_output(port = 3))
                                event.connection.send(msg)

                        if event.port == 2:

                                # se le envia la accion a ejecutar
                                msg = of.ofp_packet_out()
                                msg.data = event.ofp
                                # se le asigna un Id a la VLAN en cuestión en este caso 20
                                msg.actions.append(of.ofp_action_vlan_vid(vlan_vid = 20))
                                msg.actions.append(of.ofp_action_output(port = 3))
                                event.connection.send(msg)


                        if event.port == 3:

                                if packet.type == ethernet.VLAN_TYPE: # Si es un paquete VLAN

                                        if packet.next.id == 10: #VLAN 10

                                                # se le envia la accion a ejecutar
                                                msg = of.ofp_packet_out()
                                                msg.data = event.ofp
                                        # se indica al switch que elimine el encabezado 802.1q
                                                msg.actions.append(of.ofp_action_strip_vlan ())
                                                msg.actions.append(of.ofp_action_output(port = 1))
                                                event.connection.send(msg)

                                        if packet.next.id == 20: #VLAN 20

                                                # se le envia la accion a ejecutar
                                                msg = of.ofp_packet_out()
                                                msg.data = event.ofp
```

```
                                        # se indica al switch que elimine el encabezado 802.1q
                                                msg.actions.append(of.ofp_action_strip_vlan ())
                                                msg.actions.append(of.ofp_action_output(port = 2))
                                                event.connection.send(msg)


                        if(dpidToStr(event.dpid)=='00-00-00-00-00-02'): #---------------------------------- SW 2 ----------------
-------------------------------------------#

                                if event.port == 3:

                                        # se le envia la accion a ejecutar
                                        msg = of.ofp_packet_out()
                                        msg.data = event.ofp
                                        # se le asigna un Id a la VLAN en cuestión en este caso 10
                                        msg.actions.append(of.ofp_action_vlan_vid(vlan_vid = 10))
                                        msg.actions.append(of.ofp_action_output(port = 1))
                                        event.connection.send(msg)


                                if event.port == 4:

                                        # se le envia la accion a ejecutar
                                        msg = of.ofp_packet_out()
                                        msg.data = event.ofp
                                        # se le asigna un Id a la VLAN en cuestión en este caso 20
                                        msg.actions.append(of.ofp_action_vlan_vid(vlan_vid = 20))
                                        msg.actions.append(of.ofp_action_output(port = 1))
                                        event.connection.send(msg)


                                if event.port == 1:

                                        if packet.type == ethernet.VLAN_TYPE: # Si es un paquete VLAN

                                                if packet.next.id == 10: #VLAN 10

                                                        # se le envia la accion a ejecutar
                                                        msg = of.ofp_packet_out()
                                                        msg.data = event.ofp
                                                # se indica al switch que elimine el encabezado 802.1q
                                                        msg.actions.append(of.ofp_action_strip_vlan ())
                                                        msg.actions.append(of.ofp_action_output(port = 3))
                                                        event.connection.send(msg)

                                                if packet.next.id == 20: #VLAN 20

                                                        # se le envia la accion a ejecutar
                                                        msg = of.ofp_packet_out()
                                                        msg.data = event.ofp
                                                        # # se indica al switch que elimine el encabezado
                                                802.1q
                                                        msg.actions.append(of.ofp_action_strip_vlan ())
                                                        msg.actions.append(of.ofp_action_output(port = 4))
                                                        event.connection.send(msg)

def launch ():
        core.openflow.addListenerByName("PacketIn", _handle_PacketIn)
```

VIII. Referencias bibliográficas

[1] Cisco. Intent-based-networking.html. Retrieved 06 14, 2021, from https://www.cisco.com/c/en_uk/solutions/enterprise-networks/intent-based-networking.html

[2] Benzekki, K., El Fergougui, A., & Elbelrhiti Elalaoui, A. (2016). Software-defined networking (SDN): a survey. Security and Communication Networks, 9(18), 5803–5833. doi:10.1002/sec.1737.

[3] Bwalya, B., & Zimba, A. (2021). An SDN approach to mitigating network management challenges in traditional networks. International Journal on Information Technologies and Security, 13(4), 3-14.

[4] Rana, D. S., Dhondiyal, S. A., & Chamoli, S. K. (2019). Software defined networking (SDN) challenges, issues and solution. International journal of computer sciences and engineering, 7(1), 884-889.

[5] Kreutz, Diego & Ramos, Fernando & Veríssimo, Paulo & Esteve Rothenberg, Christian & Azodolmolky, Siamak & Uhlig, Steve. (2014). Software-Defined Networking: A Comprehensive Survey. ArXiv e-prints. 103. 10.1109/JPROC.2014.2371999.

[6] Jadhav, Neha. (2018). Cross Layer Design in Software Defined Networking (SDN): Integration of two technologies. 10.13140/RG.2.2.34256.15361.

[7] Rana, Deepak & Dhondiyal, Shiv & Chamoli, Sushil. (2019). Software Defined Networking (SDN) Challenges, issues and Solution. INTERNATIONAL JOURNAL OF COMPUTER SCIENCES AND ENGINEERING. 7. 884-889. 10.26438/ijcse/v7i1.884889.

[8] Khorsandroo, S., Sánchez, A. G., Tosun, A. S., Arco, J. M., & Doriguzzi-Corin, R. (2021). Hybrid SDN evolution: A comprehensive survey of the state-of-the-art. Computer Networks, 192, 107981.

[9] Kaur, Sukhveer & Singh, Japinder & Ghumman, Navtej. (2014). Network Programmability Using POX Controller. 10.13140/RG.2.1.1950.6961.

[10] Li, W., Meng, W., & Kwok, L. F. (2016). A survey on OpenFlow-based Software Defined Networks: Security challenges and countermeasures. Journal of Network and Computer Applications, 68, 126–139. doi:10.1016/j.jnca.2016.04.011

[11] Noman, H. M., & Jasim, M. N. (2020, July). Pox controller and open flow performance evaluation in software defined networks (sdn) using mininet emulator. In IOP conference series: materials science and engineering (Vol. 881, No. 1, p. 012102). IOP Publishing.

[12] Fernandez, M. P. (2013). [IEEE 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA) - Barcelona (2013.3.25-2013.3.28)] 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA) - Comparing OpenFlow Controller Paradigms Scalability: Reactive and Proactive. , (), 1009–1016. doi:10.1109/AINA.2013.113

[13] Poulton, N. (2020). Docker Deep Dive: Zero to Docker in a single book. NIGEL POULTON LTD.