Escuela Politécnica Superior de Linares

Grado en Ingeniería Telemática

Universidad de Jaén
Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

SIMULACIÓN DE REDES DEFINIDAS POR SOFTWARE: EJEMPLOS DE USO.

Alumno: Juan Francisco Caballero Herrera

Tutor: Dr. Antonio Jesús Yuste Delgado
Depto.: Ingeniería de Telecomunicación

Noviembre, 2020

1. Introducción

Internet ha sufrido una gran transformación desde sus inicios en la década de 1960 hasta nuestros días. Sus protocolos y capacidades han ido avanzando rápidamente, así como su penetración en la sociedad tanto civil como militar. Toda esta evolución no había sido acompañada por un avance real en las técnicas de gestión de las redes hasta que, a mediados de la década de 1990, nacen las primeras iniciativas para separación del plano de control y datos.

Como en toda innovación, sus primeros pasos no fueron muy exitosos y se necesitó de un periodo de pruebas, fracasos y nuevos enfoques para llegar a un paradigma más consolidado y que goza de una mayor reputación. Gracias al importante avance de esta tecnología se pueden obtener redes adaptables, fácilmente escalables y administradas de forma centralizada de manera eficiente. Esto hace idóneo su uso en el desarrollo de la red 5G, la cual buscan la flexibilidad y la eficiencia como características clave.

A lo largo de este Trabajo Fin de Grado se intentará aportar una visión general de las redes definidas por software (SDN por sus siglas del inglés Software-Defined Networking), desde su nacimiento hasta el estado actual del arte, conociendo su arquitectura y realizando varios ejemplos de uso para conocer las ventajas y funcionamiento de este paradigma.

2. Antecedentes

2.1. Orígenes y evolución de Internet

La red Internet ha evolucionado en gran medida desde que en agosto de 1962 Joseph Carl Robnett Licklider, profesor e investigador del MIT (siglas del inglés Massachusetts Institute of Technology), redactara una serie de memorandos describiendo el concepto de una "Red Galáctica" [1]. Esta idea consistía en una serie de ordenadores interconectados mediante los cuales se podía acceder a datos y programas de forma rápida y global, muy similar al Internet actual. En octubre de ese mismo año, Licklider se puso a la cabeza de la Agencia de Proyectos de Investigación Avanzados de Defensa (DARPA por sus siglas del inglés Defense Advanced Research Projects Agency) en el programa de investigación informática, trabajando junto a Ivan Sutherland, Bob Taylor, y Lawrence G. Roberts, investigador del MIT, los cuales le sucederían en la agencia.

El primer documento sobre la teoría de conmutación de paquetes lo publicó Leonard Kleinrock en julio de 1961. No sería hasta 1964 cuando Kleinrock convenció a Roberts sobre la viabilidad de hacer uso de esta tecnología en vez del uso de circuitos dedicados. En 1965, gracias el trabajo conjunto de Roberts y Thomas Merrill, se conectaron ordenadores dos ordenadores entre Massachusetts y California, mediante línea telefónica conmutada de baja velocidad, estableciéndose la primera red de área amplia (WAN por sus siglas del inglés Wide Area Network). El resultado del experimento fue positivo y evidenció la posibilidad de interconectar equipos para trabajar y compartir información sin que ello supusiera un perjuicio para su rendimiento. Sin embargo, el uso del sistema telefónico de conmutación de circuitos era altamente ineficiente y dejaba de manifiesto la necesidad de la conmutación de paquetes ideado por Kleinrock.

A finales de 1966 Roberts entró a formar parte de DARPA para desarrollar el concepto de redes de telecomunicaciones, dando origen al concepto de Red de la Agencia de Proyectos de Investigación Avanzada (ARPANET por sus siglas del inglés Advanced Research Projects Agency Network). El concepto fue publicado en 1967, pero no sería hasta septiembre de 1969 que no se realizaría la primera conexión entre en campus de Los Ángeles de la Universidad de California (UCLA) y el Stanford Research Institute (SRI). Un mes más tarde se enviaría el primer mensaje host-to-host, y se unirían otros dos equipos, uno en el campus en Santa Bárbara de la Universidad de California (UCSB) y en la Universidad de Utah.

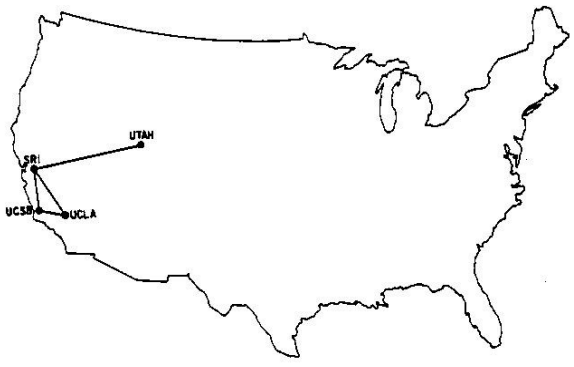


Imagen 2.1: Mapa de Arpanet en 1969 [2]

Se comienzan a desarrollar protocolos para el correcto funcionamiento de la red. El primero de ellos fue Network Control Protocol (NCP) y detallaron todas sus especificaciones técnicas en las Request For Comment (RFC), así como las de todos los protocolos que lo siguieron. Durante 1971 se desarrollaron los primeros protocolos a nivel de aplicación, el Protocolo de transferencia de archivos (FTP por sus siglas del inglés File Transfer Protocol) y Red de teletipo (TELNET por sus siglas del inglés Teletype Network), ampliamente usados hoy en día, y el extinto Mail Box Protocol (MBP).

Los dispositivos encargados de interconectar las redes en esta red primitiva se conocían como Procesadores de mensajes de interfaz (IMP por sus siglas del inglés Interface Message Processors). El número de equipos fue creciendo rápidamente, sumando un total de 65 equipos para 1976. Pero había un problema, debido al gran control y estandarización presentes en ARPANET no podía intercomunicarse con otras redes coetáneas como SATNET, red satelital, o ALOHANET, red de conmutación de paquetes de radio creada en Hawaii [3]. Debido a esta problemática, Robert Elliot Kahn, al que más tarde se le uniría Vinton Gray "Vint" Cerf, comenzó a desarrollar el protocolo de control de transmisión (TCP por sus siglas del inglés Transmission Control Protocol), cuyo concepto fue documentado por primera vez en la RFC 675 en diciembre de 1974. El concepto se basaba en la posibilidad de la interconexión de redes de diferente naturaleza, con una administración descentralizada, la inclusión de la protección ante errores mediante la retransmisión de paquetes y que no fuera necesario el cambio en una red para conectarse a otra.

Se realizaron varias pruebas antes de dar el salto definitivo a TCP/IP. El primero se realizó en 1975 entre Stanford y el London University College. En 1977 se realizaría un segundo test a mayor escala, uniendo diversos sitios de EE UU, Reino Unido y Noruega. A partir de este momento se desarrollaron diversas versiones de TCP/IP hasta que en 1983 se adoptó de forma definitiva la tecnología mediante la migración de ARPANET a TCP/IP.

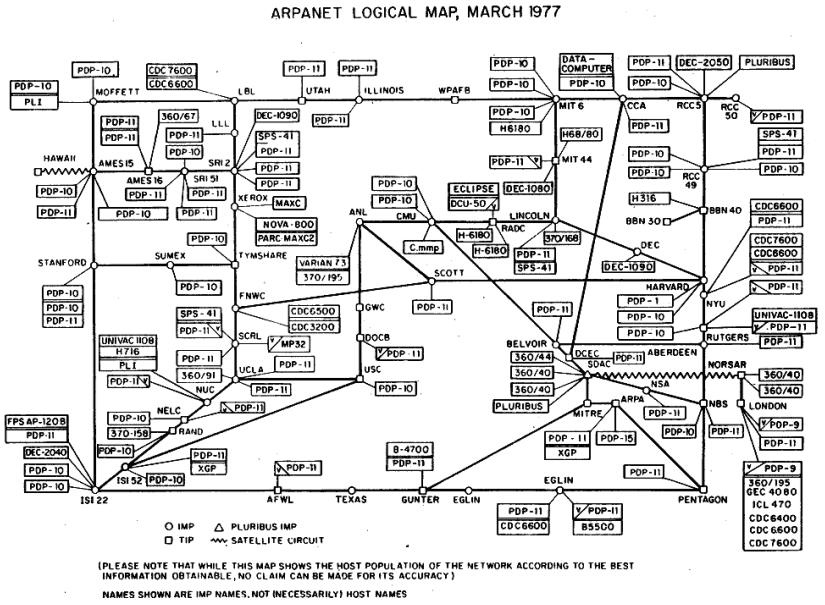


Imagen 2.2: Mapa lógico de ARPANET en marzo de 1977 [4]

De forma paralela, durante el año 1981 se desarrolla el protocolo para la transferencia simple de correo (SMPT por sus siglas del inglés Simple Mail Transfer Protocol), documentado en la RFC 821, el cual emula el correo tradicional de forma sencilla. En 1983 surge el protocolo de espacio de nombres de dominio (DNS por sus siglas del inglés Domain Name Space) ayudando a la jerarquización de la red, aunque no sería hasta 1984 cuando se liberarán los dominios .org, .net, .com, .edu, .gov, .mil y los respectivos para cada país, como .es para el caso de España o .uk para Reino Unido.

Aunque se considere a ARPANET la predecesora de Internet, la primera red en hacer de backbone de Internet fue NFSNET, desarrollada por la Fundación Nacional de Ciencia (NSF por sus siglas del inglés National Science Foundation), dependiente del gobierno de los Estados Unidos. Esta red comenzó a operar en 1985 haciendo uso de las mismas tecnologías y protocolos que ARPANET y tuvo un éxito inmediato. La realidad en Europa era bastante diferente ya que veían TCP/IP como un experimento, y esperaban a un desarrollo más completo del modelo y protocolos OSI para un despegue e interconexión con EEUU. Debido a un retraso notable en el desarrollo de dicho paradigma, y a la progresiva implantación en entornos académicos de equipos con sistemas operativos basados en Unix, los cuales incluían los protocolos de TCP/IP de forma nativa, potenciaron la adopción de los mismos y la interconexión entre países.

En paralelo al despliegue de la NFSNET, en 1989 en la Organización Europea para la Investigación Nuclear (CERN por sus siglas de frances Conseil Européenne pour la Recherche Nucléaire) se inició el desarrollo de un nuevo proyecto de gestión de la información. No sería hasta tres años más tarde, en 1992, cuando nacería la red informática mundial (WWW por sus siglas del inglés World Wide Web) de manos de Tim Berners-Lee, que contenía el protocolo de transferencia de hipertexto (HTTP por sus siglas del inglés Hypertext Transfer Protocol) que permitía la inclusión de enlaces a otras páginas diferentes o incluso de contenido multimedia. Esto supuso el despegue definitivo de Internet y el surgimiento de los primeros buscadores con entorno gráfico. En 1993 nace el buscador gratuito Mosaic, que posteriormente pasaría a denominarse Netscape y llegaría hasta nuestros días como Mozilla Firefox. Un año más tarde aparece en escena Internet Explorer, el buscador gratuito de Microsoft. Para aquel entonces ya se encontraban conectados a la red más de 600.000 hosts repartidos en más de 5.000 redes localizadas en más de 100 países.

En 1995 la NFS deja el control de Internet en mano de cuatro empresas americanas (MFS Datanet, Sprint, Ameritech y Pacific Bell) que constituirán los llamados puntos de acceso a la red (NAP por sus siglas del inglés Network Access Point) y que se encargaría de proporcionar acceso a la red a las empresas que ofertan los servicios de conexión a Internet, conocidas como ISP (por sus siglas del inglés Internet Service Provider). Este hecho constituyó el primer paso hacia la neutralidad de la red (no depende de ningún país) y a una estructura descentralizada de la misma. A partir de este momento, multitud de empresas empiezan a desplegar sus propias redes, migrando del par trenzado a la fibra óptica como medio de transmisión.

En cuanto al volumen de tráfico generado en la red, su evolución ha sido exponencial. Esto se puede ver perfectamente en el punto de interconexión DE-CIX de Frankfurt, que ha multiplicado su tráfico por 225.000 desde los 40 Mbps en 1995 al pico de 9,1 Tbps registrado durante la pandemia causada por el Covid-19. Algunos de los hitos más relevantes de este crecimiento fueron el primer auge de los juegos de PC, haciendo

que en el año 2002 se alcanzara un pico de 8,5 Gbps, el nacimiento de redes sociales como Facebook y Twitter, así como de Youtube, dieron lugar a un nuevo pico histórico de 90 Gbps. En 2007 se lanza el primer iPhone, provocando que en 2008 se alcanzaran los 500 Gbps, y surge en 2010 la red social de fotografía Instagram, observándose un pico de 2 Tbps en el tráfico. En 2018 la mitad de la población mundial se encontraba conectada a Internet y se obtuvo un pico de 6,7 Tbps que se superó un año después llegando a los 8 Tbps. Durante los meses de marzo y abril de 2020 se alcanzan los 9,1 Tbps debido al confinamiento generalizado que ha sufrido Europa, aunque actualmente el volumen de tráfico se ha visto reducido de nuevo [5].

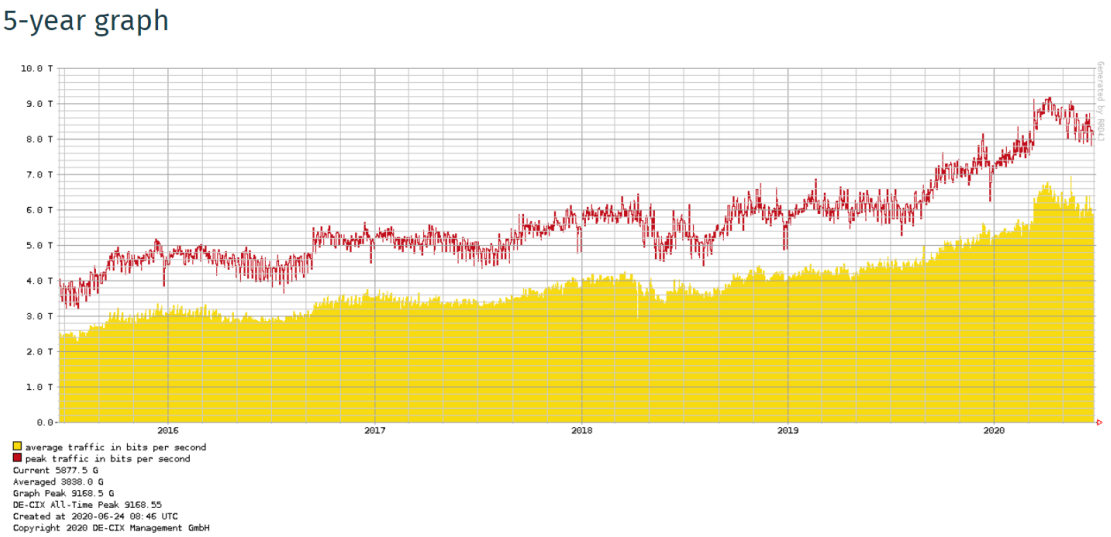


Imagen 2.3: Media y pico de tráfico durante los últimos cinco años en DE-CIX [5]

En cuanto al nacimiento de Internet en España,vino de la mano de la Compañía Telefónica Nacional de España (CTNE), siendo las primeras redes de datos se desarrolladas en la década de los 60. Las primeras empresas en contar con conmutación de circuitos fueron Renfe, Iberia y La Caixa, entre otros, a finales de la década. En la misma época, el gobierno de EE.UU solicitó "la instalación de los primeros circuitos dedicados de larga distancia que permitiesen comunicar la base de Rota con el Pentágono" [6] y Banesto plantea la primera gran red entre sus oficinas mediante el uso de conmutación de paquetes.

En 1971, la CTNE crea la Red Especial de Transmisión de Datos (RETD) tomando ARPANET como base. Dicha red se queda obsoleta en 1978 debido al gran crecimiento del tráfico y surge el sistema TESYS, donde Telefónica, en colaboración con Secoinsa y Sitre, se encargarían de la manufactura de los nuevos equipos que conformarían la red y entrarían en fase de producción en 1984. A principio de la década de los 80 llegan a España varias redes redes de datos como EUnet, para usuarios de UNIX, y EARN, financiada por IBM y cuyo propósito era conectar los equipos de dicha marca. Ya en 1988 se crea la red RedIRIS (Interconexión de los Recursos Informáticos de las Universidades y Centros de Investigación), pública y de vocación académica, que suponía la conexión de España a Internet, y cuyo desarrollo duró hasta 1990.

No sería hasta 1992 cuando llegaría el acceso a Internet de mano de compañías privadas. Su rápida aceptación llevó a que en 1995 España contara con 10 compañías que ofrecieran Internet. Este crecimiento se vio impulsado enormemente gracias a la liberalización de las telecomunicaciones en 1997.

Actualmente, según los datos ofrecidos por [7], en 2017 el 85% de la población española contaba con acceso a Internet, situándose España en el top 20 de los países con mejor conectividad.

2.2. Orígenes de SDN

Ante el gran crecimiento experimentado en las redes, la administración y configuración de equipos intermedios, como switches o routers, de forma tradicional se hace insostenible. Esto se debe a la dificultad de llevar a cabo cambios en las redes puesto que cada equipo se debe configurar manualmente y de forma independiente. Como podemos ver en [8], existen varios intentos previos de llevar la programación a las redes de una forma más o menos exitosa. Algunos de los casos y su aproximación son los siguientes:

- Open Signaling (1994) [9]: separación del plano hardware de comunicación y software de control mediante programación de puertos de switches y routers.

- Active Networking (1995) [10]: infraestructura de red programable para servicios concretos, mediante switches programables que diferencian tipos de tráfico y ejecutan pseudocódigo incluido en los paquetes.

- DCAN (1995) [11]: pensado para conseguir de forma escalable el control y la gestión de redes asíncronas.

- 4D Project (2004) [12]: busca un nuevo diseño que enfatice la separación entre la parte lógica, encargada de las decisiones de enrutamiento, y los protocolos que manejan el funcionamiento de la red.

- NETCONF (2006) [13] : propone un protocolo de gestión para gestionar los equipos de la red, en un intento de solucionar los problemas del protocolo SNMP [14] como la falta de seguridad

- Ethane (2006) [15] : predecesor inmediato de OpenFlow que define una nueva arquitectura de red y hace uso de un controlador centralizado que gestiona las políticas de red y seguridad en la red.

Ya en 2011 surge la Open Network Foundation [16] con el objetivo de desarrollar, entre otros proyectos, SDN, al que definen como “la separación física del plano de control de la red del plano de reenvío, y donde un plano de control controla varios dispositivos” [17] e impulsarlo más allá del ámbito universitario en el que nace.

3. Objetivos

El concepto SDN se trata de un gran desconocido en el ámbito universitario. Sin embargo, se trata de un paradigma que está ganando protagonismo en el ámbito empresarial debido a su gran cantidad de ventajas.

De ese modo, un estudio de los elementos que componen una red SDN junto con un manual para

Los objetivos a tratar en el presente trabajo son:

- Estudio de la arquitectura SDN más usual.
- Estudio de los diferentes componentes hardware presentes en la red.
- Estudio de los diferentes protocolos existentes en el paradigma.
- Elección de los protocolos a utilizar en la simulación.
- Estudio de los simuladores existentes para el despliegue de redes SDN.
- Elección de simulador/es, despliegue de una red y realización de test de rendimiento.
- Realización de un manual de instalación de los elementos necesarios con explicaciones detalladas paso a paso.

4. Descripción de las soluciones adoptadas

4.1. Análisis de la arquitectura y componentes SDN

Debido a que no existe una única aproximación de SDN ni un estándar de facto, se pueden encontrar varias arquitecturas

4.1.1. Arquitectura

Las diferencias entre la arquitectura tradicional y la arquitectura propuesta por SDN son notables, así como los beneficios que el cambio de paradigma aporta. Actualmente existen dos arquitecturas que destacan sobre el resto, la propuesta por la IETF, ForCES [18], y OpenFlow [19]. La principal diferencia entre ambas es el modelo de reenvío usado, ForCES hace uso de bloques de funciones lógicas mientras que OpenFlow se basa en tablas. Debido a su mayor penetración tanto en el mundo académico como industrial, nuestro estudio lo basaremos sobre la arquitectura OpenFlow.

Como podemos ver en la imagen 4.1, la arquitectura tradicional de la red se compone de equipos de usuarios finales conectados a equipos intermedios como puede ser routers o switches. Estos equipos intermedios se interconectan mediante diversos medios de transmisión como puede ser la fibra óptica, par trenzado o radioenlaces, creando redundancia para poder ser robustos ante los posibles fallos que se pudieran ocasionar.

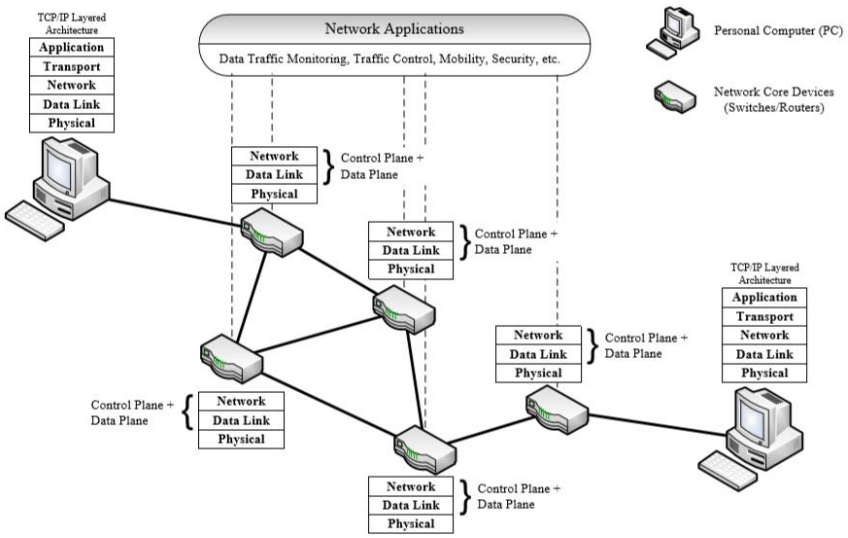


Imagen 4.1: Arquitectura de red tradicional [20]

La gestión de estos equipos se hace de forma distribuida ya que cada uno incluye su propio firmware con unas funciones concretas, que toma decisiones con respecto al tráfico recibido, apoyándose en más de 8000 RFCs [21]. A ello hay que añadirle la dificultad para innovar, ya que cualquier cambio de equipos que conforman la red es muy costoso y podría suponer un desastre, así como la inclusión de nuevos protocolos en equipos preexistentes, que necesitarían de una actualización del firmware que tendría que realizarse equipo a equipo.

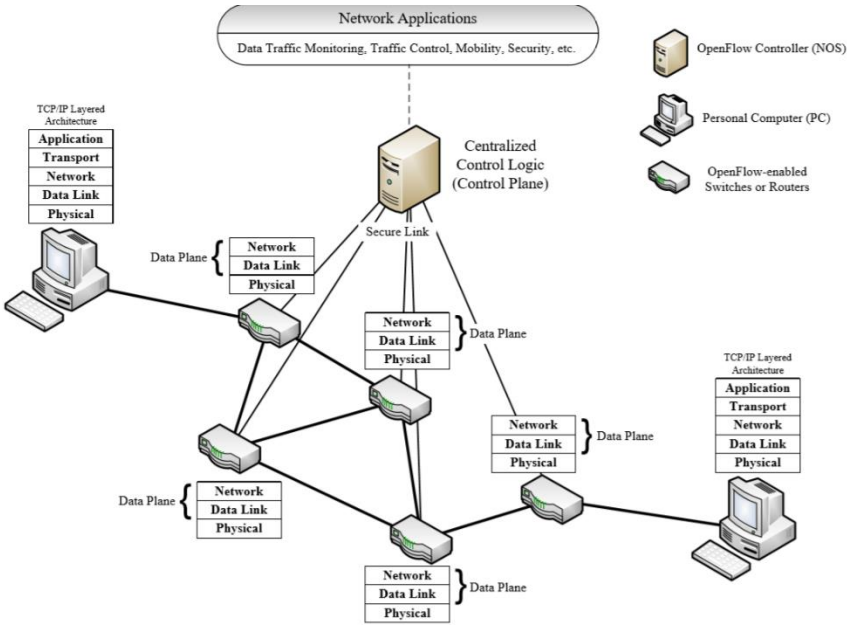


Imagen 4.2: Arquitectura paradigma SDN OpenFlow [20]

Por otro lado, la arquitectura en SDN separa el plano de datos (hardware) del de control (software) y se hallan tres capas claramente diferenciadas en esta arquitectura. La superior se conoce como capa de aplicación y la componen las conocidas como NorthBound APIs. Estas aplicaciones cuentan con los patrones de uso de la red de cada aplicación y son las encargadas de comunicarlo al nivel intermedio. El nivel intermedio, también conocido como nivel de control, cuenta con un controlador SDN, encargado de tomar las decisiones de reenvío según los datos aportados por el nivel superior, y de comunicar estas decisiones a los equipos del nivel inferior a través de las SouthBound APIs. El nivel inferior se trata del nivel de infraestructura y se compone de hosts, switches, routers y los medios de transmisión utilizados, que ejecutan las decisiones de encaminamiento según las tablas de encaminamiento proporcionadas por el nivel de control.

4.1.2. Dispositivos de reenvío

Los dispositivos que se utilizan para el acceso a la red y el reenvío de paquetes pueden ser diversos, tanto en fabricante como en funcionamiento, y entre ellos se encuentran routers, switches, switches virtuales y puntos de acceso inalámbricos entre otros. Para simplificar estas diferencias, en SDN se toman todos estos dispositivos como switches y que pueden ser de dos tipos, puros (OpenFlow-only) o híbridos (OpenFlow-hybrid).

Los switches OpenFlow-only, como su propio nombre indica, solo pueden funcionar haciendo uso de SDN, mientras que los OpenFlow-hybrid pueden funcionar indistintamente en entornos SDN y en los modos de funcionamiento normales, pero tienen que poder distinguir el tipo de tráfico que están cursando para poder procesarlo. La mayoría de switches comerciales vendidos hoy en día, como los Junos MX-Series de Juniper o los modelos PF5240 y PF5820 de NEC, soportan SDN.

4.1.2.1 Esquema de reenvío

A la hora de realizar el encaminamiento se sigue un esquema predeterminado que ha sido revisado conforme avanzaban las versiones de OpenFlow para mejorar la compatibilidad con los diferentes protocolos. Cada switch contiene una o más tablas de flujo por las que tienen que pasar los paquetes entrantes hasta lograr encontrar una coincidencia que les seleccione una acción a realizar. Estas acciones pueden ser su reenvío o mandarlo a unas tablas de flujo de salida para realizar alguna acción sobre el paquete antes de su salida del dispositivo, aunque esto último es opcional. En caso de que el paquete que entra no verifique ninguna de las entradas en ninguna tabla de flujo se marca como "table miss" y se realiza una acción por defecto sobre él.

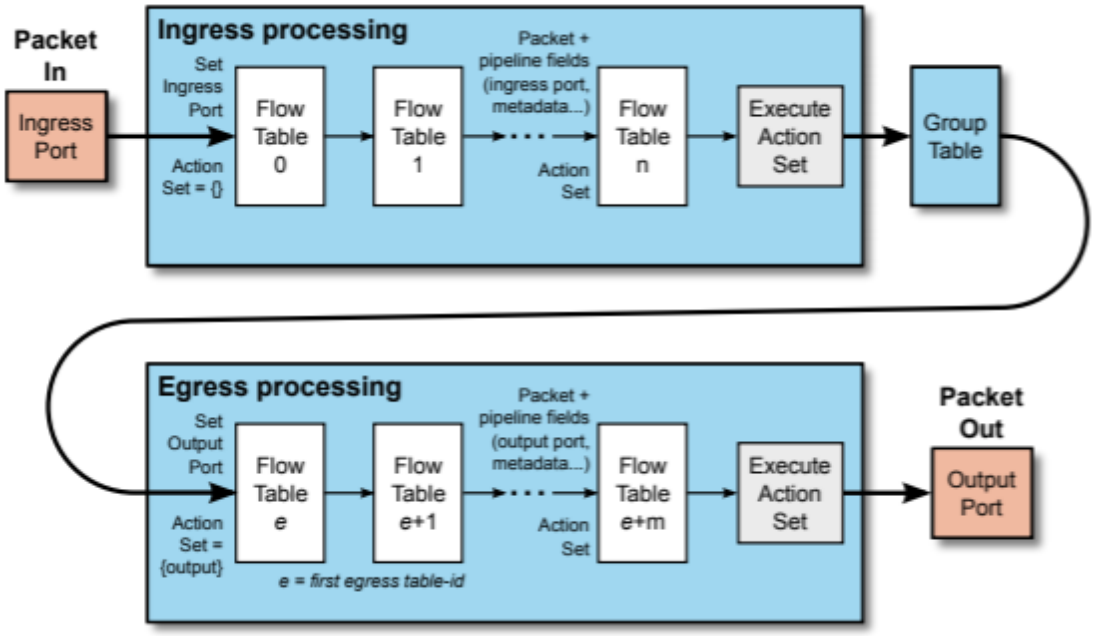


Imagen 4.3: Pipeline OpenFlow [22]

4.1.2.2 Acciones

Existe una gran cantidad de acciones posibles a realizar entre los paquetes que entran en los equipos, pero se puede clasificar en:

- acciones de reenvío: el paquete se marca para su reenvío por inundación, por un puerto concreto, hacia el controlador dentro de un mensaje OpenFlow o se marca para realizar un reenvío tradicional (esta opción solo está disponible para switches híbridos).

- acciones de modificación: se modifica algún campo de la cabecera el del paquete, lo que supone una modificación de su tratamiento a la salida.

- acción por defecto: para paquetes que no hacen match en ninguna tabla de ingreso se pueden definir acciones por defecto. Esta acción se conoce como "table miss".

- acciones de descarte: en caso de no cumplir ningún criterio y no definirse una acción por defecto, o que el paquete sea corrupto, se marca para su descarte.

- pasar a una tabla posterior, nunca a una anterior.

Todas estas acciones siguen un orden de prioridades descrito en [22] para su ejecución. Debido a la gran cantidad de operaciones que se pueden realizar sobre los paquetes OpenFlow distingue entre acciones requeridas, que suponen la base del protocolo, y opcionales, cuya implementación depende del fabricante y de la calidad del equipo final. Las acciones requeridas, y por tanto presentes en todos los equipos, son:

- *output port_no:* reenvía un paquete por un puerto determinado.
- *group group_no*: marca el paquete como parte de un grupo para su tratamiento en conjunto.
- *drop:* descarta el paquete.

Todas ellas se pueden consultar en [22].

4.1.2.3 Tablas de flujo

Las entradas en las tablas de flujo se componen de siete campos claramente diferenciados como son:

- Match fields (campos de coincidencia): es usado para hacer match con los paquetes que entran al equipo. Consiste en las cabeceras de los paquetes entrantes y, de forma opcional, otros campos del pipeline OpenFlow como pueden ser metadatos especificados por la tabla anterior.

- Priority (prioridad): en caso de verificarse varias entradas, se seleccionará la que cuente con una mayor prioridad.

- Counters (contadores): se realiza un conteo de coincidencias de paquetes dependiendo de varios criterios, tanto obligatorios como opcionales, y que se actualiza cada vez que un paquete hace match con la entrada en la tabla. Existe una gran cantidad de ellos que puede ser consultada en [22].

- Instructions (instrucciones): sirven para modificar la acción seleccionada o el camino a seguir por el paquete en el conmutador.

- Timeouts (temporizador): contador con la máxima cantidad de tiempo que puede permanecer un paquete en el equipo antes de proceder a su descarte.

- Cookie: se trata de un valor usado por el conmutador para filtrar paquetes en caso de que fuera necesario, el switch omite ese valor.

- Flags (banderas): se trata de marcas para el procesado de entradas en la tabla, como puede ser su eliminación o la desactivación de contadores.

Las entradas en las tablas de flujo se identifican por sus match fields y priority, que a su vez identifican de forma única los flujos de entrada de cada tabla. Como se puede

observar en la siguiente imagen, este es el proceso completo de procesador de cada paquete dentro del equipo.

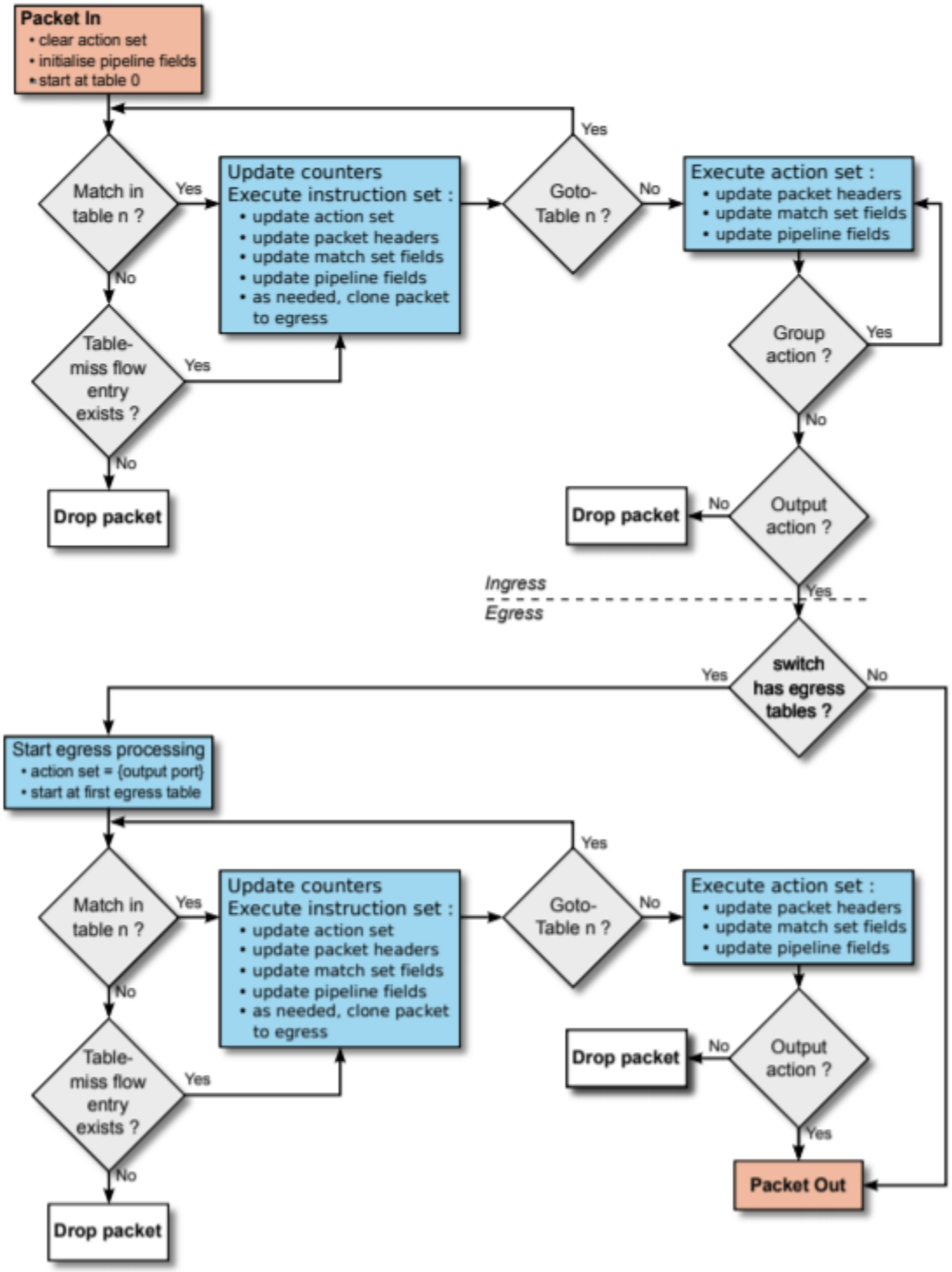


Imagen 4.4: Esquema de flujo de paquetes en switch OpenFlow [22]

Un switch normal puede soportar desde pocos miles a varias decenas de miles de reglas de reenvío [23]. El almacenamiento de tantas reglas supone un cuello de botella a la escalabilidad de OpenFlow y su correcto funcionamiento respetando las políticas de la red.

4.1.3. Southbound APIs

Una parte importante del paradigma SDN es la comunicación entre el controlador y los dispositivos de reenvío, ya que comunica el plano de control del plano de datos. Esta comunicación se lleva a cabo mediante las conocidas como southbound APIs.

Actualmente se cuenta con estándares de facto como el CLI de Cisco y el protocolo SNMP, aunque OpenFlow, como alternativa open source, representan una estandarización y madurez en el paradigma. Existen otros protocolos como OF-Config, del propio OpenFlow, y Open vSwitch Database Management Protocol (OVSDB) de Nicira, pero se limitan a roles de configuración.

4.1.3.1. OpenFlow

OpenFlow es el protocolo de comunicación más utilizado en el paradigma SDN y se usa para comunicar el switch y el controlador. El puerto empleado para la comunicación es el 6653/TCP y se recomienda el uso de TLS para añadir seguridad en los mensajes y evitar su manipulación. Existen varias versiones del protocolo, siendo la 1.0 (31 de diciembre de 2009 [24]) la más extendida y la 1.5.1 (26 de marzo de 2015 [22]) la más reciente y la cual vamos a estudiar.

El protocolo cuenta con tres tipos de mensajes diferenciados según sea el que los envía:

- Controller-to-switch (del controlador al switch):

La comunicación es iniciada por el controlador y puede requerir, o no, respuesta por parte del switch. Se usan para conocer las capacidades del dispositivo de reenvío, para conocer la configuración y poder realizar cambios en la misma si fuera necesario, aunque también pueden ser usados para enviar paquetes desde el controlador a equipos finales a través del switch. Los tipos de mensajes se engloban en:

- modify-state: tienen el propósito de modificar, añadir o borrar entradas del flujo o grupo, insertar o eliminar acciones de las tablas OpenFlow y establecer propiedades en los puertos.

- read-state: estos mensajes son usados por el controlador para recoger información del switch tal como configuración actual, estadísticas o capacidades.

- packet-out: estos paquetes tienen el propósito de ser usados por el controlador para enviar paquetes por un puerto concreto del switch para su reenvío por otros equipos como si fuera un paquete normal.

- barrier: los mensajes de solicitud o respuesta de barrera son utilizados por el controlador para obtener información sobre operaciones completadas.

- role-request: los mensajes de solicitud de rol son usados por el controlador para establecer el rol de su canal OpenFlow y fijar o consultar su ID de controlador.

- asynchronous-configuration: son utilizados para establecer un filtro adicional en los mensajes asíncronos que quiere recibir en su canal OpenFlow,o para consultar ese filtro. Suelen utilizarse al establecer el canal OpenFlow.

    - Asynchronous (asíncronos)

En este tipo de mensajes son los switches los que inician la comunicación con el controlador sin necesidad de un mensaje previo por parte de este. Son usados para notificar la llegada de paquetes o de cambios en el estado del equipo. Los principales tipos de mensajes asíncronos son los siguientes:

- packet-in: transfieren el control sobre el paquete directamente al controlador. Para todos los paquetes reenviados al puerto reservado del controlador se reserva una entrada en la tabla de flujo o la entrada table-miss.

- flow-removed: son usados para informar al controlador del borrado de una entrada en una tabla de flujo.

- port-status: informa del cambio en un puerto, ya sea de estado o configuración.

- role-status: se usan para informar del cambio de rol del equipo. Suelen enviarse al establecer conexión con un nuevo controlador.

- controller-status: son enviados al detectar cambios en un canal de comunicación OpenFlow

- flow-monitor: informa al controlador cuando se realiza algún cambio en una tabla de flujo según los criterios establecidos previamente por el controlador para monitorización.

    - Symmetric (simétricos)

Son mensajes que pueden ser enviados indistintamente por el controlador o el switch sin tener una solicitud previa por parte de ninguno de ellos. Existen cuatro tipos de mensajes simétricos, que son:

- hello: este mensajes es usado al inicio de la conexión para establecer la versión de OpenFlow a usar y negociar todos los parámetros a usar.

- echo: cualquiera de los dos puede iniciar el intercambio de mensajes y el otro debe mandar una respuesta de echo. Sirven para verificar que la conexión sigue levantada y para medir latencias en la misma.

- error: son utilizados para notificar errores a la otra parte, mayoritariamente son usados por el switch para notificar incidencias al controlador.

- experimenter: ofrecen un estándar a los switches OpenFlow para añadir nuevas funcionalidades, la mayoría de ellas se encuentran en revisión.

4.1.4 Controlador

Se trata del elemento principal para cualquier arquitectura SDN, ya que contiene una vista completa de la red, implementa las políticas a aplicar en la red, está al cargo de todos los dispositivos SDN que la componen y ofrece una API norte para las aplicaciones. Suelen incluir una serie de módulos como un pequeño switch y router, un firewall básico y un balanceador de carga. Además, cuenta con dos tipos de APIs, una orientada al sur (southbound), explicada en 4.1.3, y otra hacia el norte (northbound), explicada en 4.1.5.

El controlador abstrae los detalles de la comunicación con los equipos y permite a las aplicaciones de red interactuar con ellos. Todos los controladores se proveen de módulos para ofrecer unas funcionalidades básicas, aunque se pueden añadir más en función de las necesidades concretas de la red que se vaya a administrar. Dichas funcionalidades básicas son las siguientes:

- Descubrimiento de equipos de usuario (end-user device discovery).
- Descubrimiento de equipos de red (network device discovery).
- Administración de la topología de equipos de red (network device topology management): tanto para equipos de red como de los equipos de usuario directamente conectados a ellos.
- Administración de flujos: mantiene una base de datos de los flujos que se están manejando y ejecuta las acciones necesarias para mantenerla actualizada

Actualmente, existen en el mercado controladores SDN open source y propietarios, cada uno con sus características y ventajas. Entre los controladores de código abierto, predominan:

- NOX [25]: uno de los pioneros, fue diseñado en C++, y una gran parte de los controladores que surgen más tarde heredan de él.
- POX [25] hereda directamente de NOX, surge como una actualización del mismo, y fue desarrollado en Python.
- Beacon [26]: desarrollado en Java por la universidad de Stanford
- OpenDaylight [27]: también está desarrollado en Java y nace con la intención de ser un controlador para todo propósito. Actualmente es uno de los más usados y cuenta con una amplia documentación, ya que está gestionado por la Linux Fundation [28].

Algunos fabricantes de equipos de red cuentan con sus soluciones propietarias, como pueden ser NEC [29], IBM [30] o HP [31]

4.1.5. Northbound APIs

Se trata de la interfaz que proporciona el controlador a las aplicaciones de red para la interacción con la misma. Al contrario que su homólogo hacia el sur, no existe una API de uso generalizado o aceptada como estándar de facto. Algo que puede explicar este escenario podría ser que esta interfaz estaría completamente definida en software y no necesita de una implementación hardware que sí que es necesaria en la interfaz sur.

Existen varias aproximaciones para el desarrollo de estas APIs, aunque las más comunes son las siguientes:

- Definición de interfaces de aplicación portables de bajo nivel.
- Creación de APIs ad-hoc para un controlador en específico, con definiciones y propósitos específicos (mayor personalización).
- Traducción de los requisitos de la aplicación en solicitudes de servicio de nivel inferior.
- Proposición de una plataforma general de control, basada en Linux y abstracciones como el sistema de archivos virtuales.
- Dar permiso a los administradores de red para definir las asignaciones a módulos específicos y políticas de control de acceso.
- Usar lenguajes de programación SDN.

Las APIs que se desarrollan se pueden englobar en tres categorías diferentes:

- API REST (Representational State Transfer): se desarrollan siguiendo un paradigma cliente-servidor sin estados, en el que se puede incluir el uso de una caché, y cuyos mensajes pueden incluir trozos de código ejecutable en forma de applets o scripts. Esto ofrece una serie de ventajas como son la administración descentralizada, la posibilidad de comunicarse con cualquier equipo independientemente de su naturaleza, facilidad en la migración y en la escalabilidad [32].
- Lenguajes de programación: se hace uso de lenguajes de programación, abstrayendo los detalles del controlador para los desarrolladores de aplicaciones y proporcionando bloques básicos de código para una mayor facilidad en el trabajo. Existen dos tipos de lenguajes, uno llamado de programación lógica (programación a bajo nivel), basado en sentencias lógicas, y el otro programación reactiva funcional (programación a alto nivel), pensado para actuar según cambios en la red. Algunos de los lenguajes de programación SDN más usados son Frenetic, Procera, Nettle, NetCore, Pyretic y NetKAT.
- Otras APIs: los propios controladores son los que definen sus propias APIs para la comunicación. Algunos ejemplos son Onix con NVP NBAPI, MobileFlow con SDMN API y PANE con PANE API

4.1.6 Aplicaciones de red

Se ejecutan sobre el controlador y son las responsables de la administración de las entradas en las tablas de flujo en la red. Son capaces de configurar las rutas a seguir por los paquetes encontrando el mejor camino entre dos equipos terminales, balancear el tráfico existente entre múltiples caminos, reaccionar a cambios en la topología que ocurran en la red, como caída de links o adhesión de nuevos nodos, y redirección de tráfico con motivos de seguridad. Algunas de las aplicaciones estándar que pueden estar incluidas en el controlador son una interfaz gráfica de usuario (GUI), un learning switch y una aplicación para el enrutamiento.

Las aplicaciones se desarrollan de forma diferente según el entorno en el que se requiera su uso. Se pueden distinguir cinco grupos principales como los más usuales para el desarrollo de las aplicaciones, como son:

- Redes corporativas: pueden ser tanto redes internas de empresas como la red de una universidad cualquiera, se caracterizan por su demanda de características avanzadas en materias seguridad y rendimiento debido al manejo de datos sensibles, control de acceso de usuarios y a la necesidad de compartimentar la información accesible por cada usuario o grupo. Un ejemplo de aplicación SDN diseñada para empresas es Ethane [33], que nace en la Universidad de Stanford y es capaz de manejar hasta 19 switches, más de 300 dispositivos conectados por cable y aún más inalámbricos con un solo controlador. Actualmente está en desarrollo la segunda versión.

- Data Centers: debido a su evolución en los últimos años se ha hecho imprescindible un mejor manejo del tráfico y una implementación de nuevas políticas más rápida. La necesidad de la adaptabilidad de la red en tiempo real y la inclusión del soporte de nuevas tecnologías de forma rápida y eficaz lo hacen el espacio perfecto para la implementación de SDN. Un buen ejemplo de aplicación que gestiona los flujos es DevoFlow [34], la cual diferencia entre los que se denominan como “flujos ratón”, pequeños y para los que no es necesario el uso del controlador, y “flujos elefante”, grandes y para los que se hace uso de OpenFlow.

- Infrastructure-based Wireless Access Networks: el despliegue de redes celulares, como el actual despliegue de la infraestructura 5G, el WiFi o el WiMAX, suponen un reto, debido al roaming de los dispositivos entre puntos de acceso, al difícil control del acceso a la red y la planificación radioeléctrica de los equipos. Debido a esto, nacen Odin [35], el cual se enfoca en la abstracción de los puntos de acceso para mejorar el balanceo de carga y la movilidad de los clientes, y OpenRadio [36], enfocado en añadir flexibilidad en la capa física y MAC, siendo capaz de manejar un conjunto de redes de diferente tecnología como si fuera una sola de ellas.

- Redes ópticas: la separación del plano físico y lógico de SDN permite la fácil integración de redes de conmutación de circuitos y paquetes como si fueran una sola. Y esto se potencia en redes ópticas, ya que potencia la inclusión de varios flujos en la misma fibra, facilita la administración y control por parte de terceros

y añade virtualización a la red. Esto ha propiciado el avance hacia un SDN más específico para redes ópticas conocido como SDON

- Redes domésticas y de pequeños comercios: debido al incremento de pequeños aparatos conectados a la red, impulsado por el IoT, la administración de estas redes se ha vuelto cada vez más compleja. A lo anterior hay que añadirle la falta de seguridad existente en dichos entornos y al excesivo coste de contar con un administrador de red. Se han hecho varias propuestas para intentar implementar SDN en estos ámbitos, como implementar logs de lo que sucede en la red o intentar descubrir comportamientos anómalos de equipos, pero ninguno ha tenido éxito

4.2. Análisis de los entornos de virtualización

Para poder llevar a cabo pruebas en entornos SDN lo más usual y fácil es recurrir a entornos de virtualización de redes. Actualmente en el mercado existen varias opciones para la emulación entre las que se encuentran:

- GNS3 [37]: Se trata de uno de los emuladores más conocidos y usados ya que se trata de una opción Open Source, sin limitación de equipos a usar y que cuenta con soporte para que una gran cantidad de equipos de diferentes fabricantes puedan ser simulados. Además, cuenta con un entorno gráfico que ayuda a la creación de topologías de forma sencilla y herramientas para la fácil interacción con los equipos. Se puede instalar tanto en entornos Windows como Linux o MAC.

- EVE-NG [38]: Se presente como la alternativa a GNS3 ya que también es gratuito. Su principal diferencia es que se distribuye como una imagen para máquina virtual (.ova) y se accede a través del navegador web. Se trata de un emulador que requiere instalación previa de las imágenes de los equipos que se desean simular, por lo que ocupa un menor espacio y a su vez dificulta el inicio de las pruebas.

- Mininet [39]: Al contrario que los previos, Mininet no cuenta con una interfaz gráfica que facilite la interacción, pero su fácil instalación como imagen de máquina virtual (.ova) que cuenta con imágenes de los equipos SDN a virtualizar, OpenFlow y el controlador POX incorporados facilita en gran medida la familiarización con la herramienta.

4.3. Solución adoptada

Para la realización de las pruebas se va a utilizar una máquina virtual Ubuntu 14.04, en la que se integran los diferentes componentes, ya que es mucho más fácil la instalación de los mismos y la comunicación entre las partes.

Como entorno de virtualización se va a utilizar Mininet que, si bien no es el que más ayuda brinda gráficamente, se trata de la más fácil de poner a punto ya que cuenta con todos los componentes necesarios. Además, el despliegue de la primera topología (1 switch, 2 hosts), se realiza con "`sudo mn`".

En el apartado del controlador, se va a utilizar POX, desarrollado en Python y con una curva de aprendizaje más rápida que los demás controladores. Como contraparte, este controlador no cuenta con una interfaz gráfica que ayude con la visualización de la topología, por lo que vamos a añadirle una extensión llamada POXDesk, que, aprovechando la potencia del controlador, descubre y muestra la topología y diferente información de los switches.

Para la correcta instalación de todos los componentes se adjunta una guía de instalación en el Anexo I.

5. Simulación de distintas topologías

El desarrollo de las pruebas va a consistir en la simulación y análisis de cuatro topologías diferentes en Mininet:

- Topología por defecto, consistente en dos equipos finales y un switch intermedio.

- Topología compleja tomando como base las topologías predefinidas de Mininet.

- Topología creada con Miniedit, editor gráfico básico incluido en Mininet.

- Topología personalizada descrita en Python e interpretada por Mininet.

Para cada topología se mostrarán los pasos a seguir para su despliegue en Mininet o Miniedit, cómo iniciar el controlador POX junto a la extensión POXDesk (GUI) para su correcto funcionamiento y una mayor facilidad para la visualización de los resultados, los comandos utilizados en Mininet para el análisis de la red creada y un pequeño análisis de los paquetes capturados con Wireshark.

5.1. Topología base

Se abren tres terminales y se consigue acceso super usuario en cada una de ellas. Iniciamos POX y POSDesk, Mininet y Wireshark.

En el primero se va a lanzar POX y POSDesk con el siguiente comando

```
./pox/pox.py samples.pretty_log web messenger messenger.log_service messenger.ajax_transport openflow.of_service openflow.webservice poxdesk openflow.discovery poxdesk.tinytopo forwarding.l2_pairs
```

En el segundo se lanza Mininet con su topología por defecto mediante el siguiente comando

```
mn --controller remote -x –mac
```

y que asigna las MAC 00:00:00:00:00:01 y 00:00:00:00:00:02 y las IPs 9.0.0.1 y 9.0.0.2 respectivamente a los hosts h1 y h2.

Y en el tercero se inicia Wireshark con

```
wireshark
```

La topología desplegada es la siguiente:

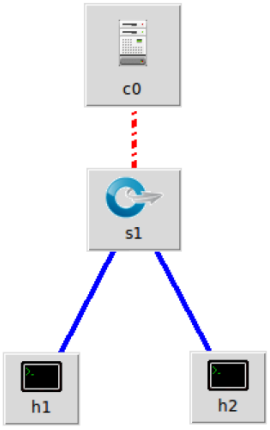


Imagen 5.1.1: topología Mininet por defecto.

Si capturamos paquetes desde el inicio de la conexión podemos comprobar que existe un intercambio de paquetes OpenFlow entre s1 y c0 para establecer los parámetros de la conexión e identificar los posibles datos iniciales con los que cuentan ambos

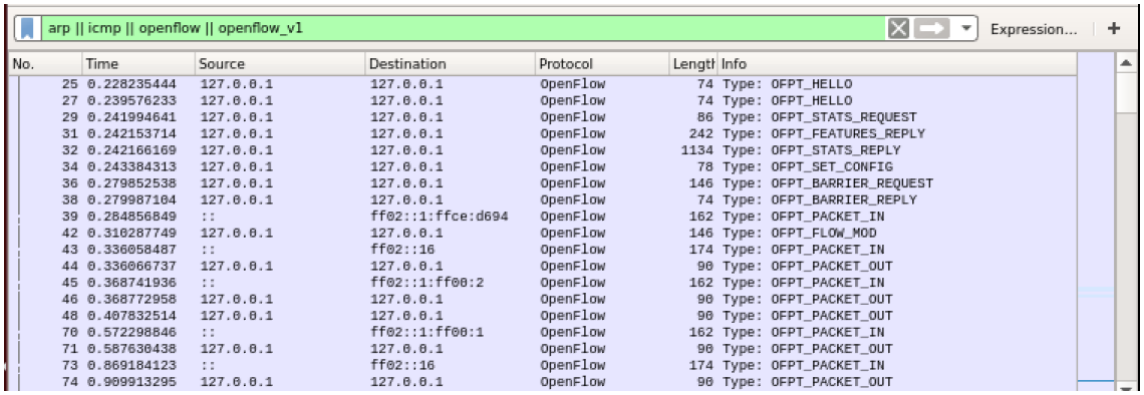


Imagen 5.1.2: establecimiento conexión entre switch y controlador.

y realiza un chequeo constante del estado de la red para descubrir los posibles cambios que ocurren en la red.

En el momento del despliegue de la red la tabla de flujo del switch s1 se encuentra vacía como se puede consultar en el TableViewer de s1 (00-00-00-00-00-01)

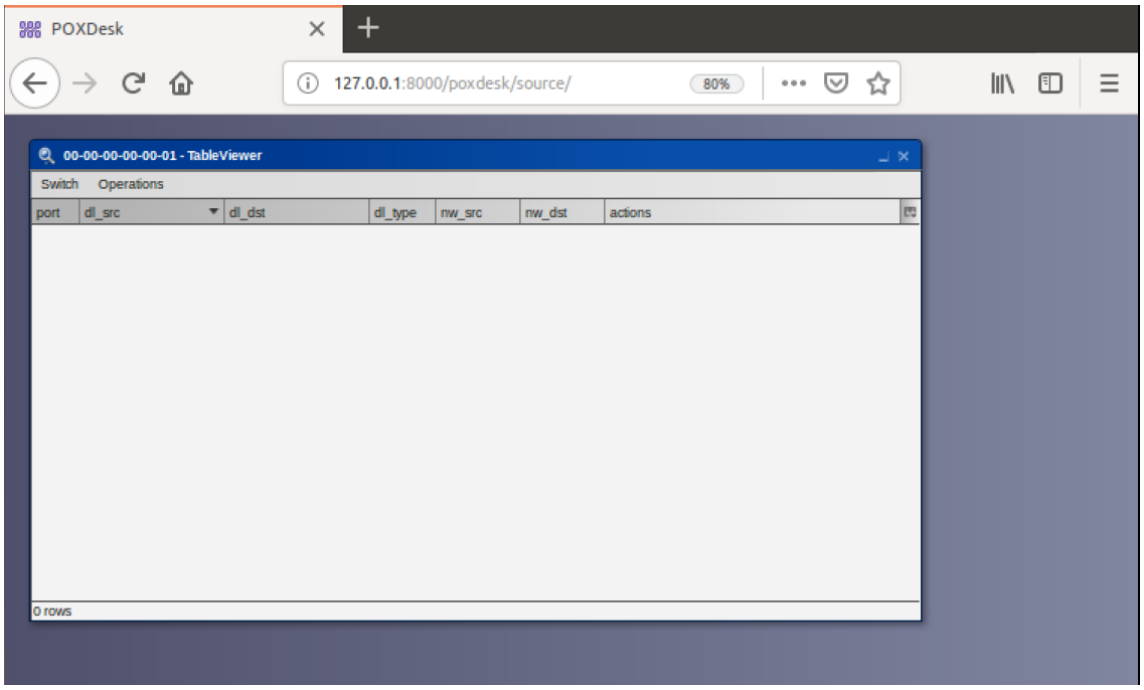


Imagen 5.1.3: tabla de flujo al inicio de la simulación.

En la sección TopoViewer se puede ver que se cuenta con un switch, aunque los hosts no se muestren como sí pasa en Mininet

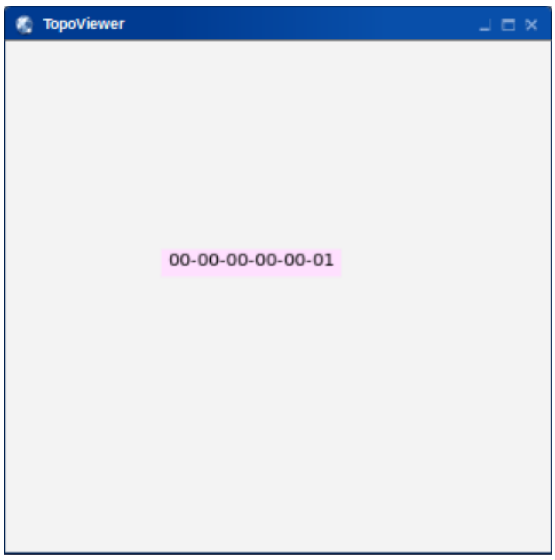


Imagen 5.1.4: topología mostrada en POXDesk.

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

Imagen 5.1.5: salida al comando net en Mininet.

donde se ve que existen dos hosts, h1 y h2, conectados en los puertos eth1 y eth2 respectivamente de s1 y un controlador c0.

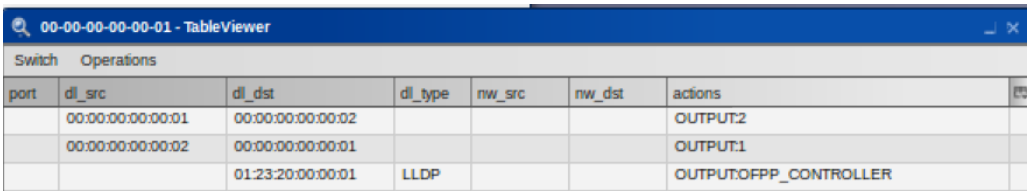
Al realizar un ping desde h2 a h1 se ve que el tiempo de respuesta del primer paquete es muy superior al resto, cosa que al realizar después el ping de manera inversa esto no ocurre.

```
root@ubuntu:/home/sdn# ping -c 3 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=62.0 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.040 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.034 ms

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000m
s
rtt min/avg/max/mdev = 0.034/20.692/62.002/29.210 ms
```

Imagen 5.1.6: ping de h2 a h1 desde consola x de Ubuntu.

Una vez realizado este intercambio de mensajes, la tabla de s1 cuenta con las entradas necesarias para futuras comunicaciones.



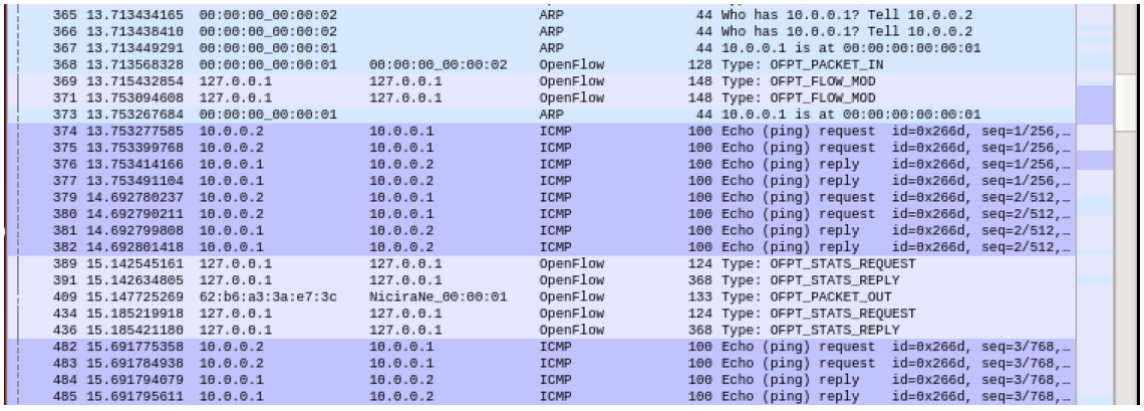
00-00-00-00-00-01 - TableViewer

Switch Operations

port	dl_src	dl_dst	dl_type	nw_src	nw_dst	actions
	00:00:00:00:00:01	00:00:00:00:00:02				OUTPUT:2
	00:00:00:00:00:02	00:00:00:00:00:01				OUTPUT:1
		01:23:20:00:00:01	LLDP			OUTPUT:OFPP_CONTROLLER

Imagen 5.1.7: tabla de flujo de s1 actualizada.

Como podemos observar en Wireshark, a fin de obtener la dirección ethernet correspondiente a la dirección IP requerida, en este caso 9.0.0.1, se lanza un ARP en la que h1 responde que a la IP 9.0.0.1 le corresponde la dirección 00:00:00:00:00:01. En ese momento s1 solicita saber si hay modificaciones en las tablas de flujo mediante un mensaje OFPT_FLOW_MOD, al cual el controlador envía una actualización y ya se produce el intercambio de mensajes ICMP (ping).



```
365 13.713434165  00:00:00_00:00:02                    ARP          44 Who has 10.0.0.1? Tell 10.0.0.2
366 13.713438410  00:00:00_00:00:02                    ARP          44 Who has 10.0.0.1? Tell 10.0.0.2
367 13.713449291  00:00:00_00:00:01                    ARP          44 10.0.0.1 is at 00:00:00:00:00:01
368 13.713568328  00:00:00_00:00:01  00:00:00_00:00:02 OpenFlow    128 Type: OFPT_PACKET_IN
369 13.715432854  127.0.0.1          127.0.0.1         OpenFlow    148 Type: OFPT_FLOW_MOD
371 13.753094608  127.0.0.1          127.0.0.1         OpenFlow    148 Type: OFPT_FLOW_MOD
373 13.753267684  00:00:00_00:00:01                    ARP          44 10.0.0.1 is at 00:00:00:00:00:01
374 13.753277585  10.0.0.2           10.0.0.1          ICMP        100 Echo (ping) request  id=0x266d, seq=1/256,…
375 13.753399768  10.0.0.2           10.0.0.1          ICMP        100 Echo (ping) request  id=0x266d, seq=1/256,…
376 13.753414166  10.0.0.1           10.0.0.2          ICMP        100 Echo (ping) reply    id=0x266d, seq=1/256,…
377 13.753491104  10.0.0.1           10.0.0.2          ICMP        100 Echo (ping) reply    id=0x266d, seq=1/256,…
379 14.692780237  10.0.0.2           10.0.0.1          ICMP        100 Echo (ping) request  id=0x266d, seq=2/512,…
380 14.692790211  10.0.0.2           10.0.0.1          ICMP        100 Echo (ping) request  id=0x266d, seq=2/512,…
381 14.692799808  10.0.0.1           10.0.0.2          ICMP        100 Echo (ping) reply    id=0x266d, seq=2/512,…
382 14.692801418  10.0.0.1           10.0.0.2          ICMP        100 Echo (ping) reply    id=0x266d, seq=2/512,…
389 15.142545161  127.0.0.1          127.0.0.1         OpenFlow    124 Type: OFPT_STATS_REQUEST
391 15.142634805  127.0.0.1          127.0.0.1         OpenFlow    368 Type: OFPT_STATS_REPLY
409 15.147725269  62:b6:a3:3a:e7:3c  NiciraNe_00:00:01 OpenFlow    133 Type: OFPT_PACKET_OUT
434 15.185219918  127.0.0.1          127.0.0.1         OpenFlow    124 Type: OFPT_STATS_REQUEST
436 15.185421180  127.0.0.1          127.0.0.1         OpenFlow    368 Type: OFPT_STATS_REPLY
482 15.691775358  10.0.0.2           10.0.0.1          ICMP        100 Echo (ping) request  id=0x266d, seq=3/768,…
483 15.691784938  10.0.0.2           10.0.0.1          ICMP        100 Echo (ping) request  id=0x266d, seq=3/768,…
484 15.691794679  10.0.0.1           10.0.0.2          ICMP        100 Echo (ping) reply    id=0x266d, seq=3/768,…
485 15.691795611  10.0.0.1           10.0.0.2          ICMP        100 Echo (ping) reply    id=0x266d, seq=3/768,…
```

Imagen 5.1.8: intercambio mensajes primer ping.

Si se repite el proceso se puede observar que la duración del retardo del primer mensaje es mucho menor a la anterior, ya que la consulta se la realiza a s1

```
root@ubuntu:/home/sdn# ping -c 3 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.551 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.063 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.055 ms

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000m
s
rtt min/avg/max/mdev = 0.055/0.223/0.551/0.231 ms
```

Imagen 5.1.9: retardos en segunda tanta del ping.

que ya cuenta con las tablas correspondientes y no necesita del intercambio de mensajes con c0.

4998	194.2698111…	Vmware_c0:00:01		ARP	62	192.168.18.1 is at 00:50:56:c0:00:01
5003	194.5692170…	62:b6:a3:3a:e7:3c	NiciraNe_00:00:01	OpenFlow	133	Type: OFPT_PACKET_OUT
5031	197.0986106…	ee:75:ef:7d:ad:dd	NiciraNe_00:00:01	OpenFlow	133	Type: OFPT_PACKET_OUT
5045	197.4995412…	127.0.0.1	127.0.0.1	OpenFlow	124	Type: OFPT_STATS_REQUEST
5047	197.4996234…	127.0.0.1	127.0.0.1	OpenFlow	368	Type: OFPT_STATS_REPLY
5071	197.5085234…	127.0.0.1	127.0.0.1	OpenFlow	124	Type: OFPT_STATS_REQUEST
5072	197.5086035…	127.0.0.1	127.0.0.1	OpenFlow	368	Type: OFPT_STATS_REPLY
5114	197.7843129…	10.0.0.2	10.0.0.1	ICMP	100	Echo (ping) request id=0x28c2, seq=1/256, ttl…
5115	197.7845393…	10.0.0.2	10.0.0.1	ICMP	100	Echo (ping) request id=0x28c2, seq=1/256, ttl…
5116	197.7845561…	10.0.0.1	10.0.0.2	ICMP	100	Echo (ping) reply id=0x28c2, seq=1/256, ttl…
5117	197.7847157…	10.0.0.1	10.0.0.2	ICMP	100	Echo (ping) reply id=0x28c2, seq=1/256, ttl…
5124	198.7833202…	10.0.0.2	10.0.0.1	ICMP	100	Echo (ping) request id=0x28c2, seq=2/512, ttl…
5125	198.7833349…	10.0.0.2	10.0.0.1	ICMP	100	Echo (ping) request id=0x28c2, seq=2/512, ttl…
5126	198.7833501…	10.0.0.1	10.0.0.2	ICMP	100	Echo (ping) reply id=0x28c2, seq=2/512, ttl…
5127	198.7833534…	10.0.0.1	10.0.0.2	ICMP	100	Echo (ping) reply id=0x28c2, seq=2/512, ttl…
5128	198.8451424…	Vmware_c0:00:01		ARP	62	Who has 192.168.0.1? Tell 192.168.18.1

Imagen 5.1.10: intercambio de mensajes en segunda tanda del ping.

En cuanto al ancho de banda entre los dispositivos podemos comprobar que es muy alto mediante el comando iperf, superior a 50Gbps.

"host: h1"

```
root@ubuntu:/home/sdn# ifconfig
h1-eth0   Link encap:Ethernet  HWaddr 00:00:00:00:00:01
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.
0.0.0
          inet6 addr: fe80::200:ff:fe00:1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:472 errors:0 dropped:186 overruns:0 fram
e:0
          TX packets:32 errors:0 dropped:0 overruns:0 carrier
:0
          collisions:0 txqueuelen:1000
          RX bytes:58368 (58.3 KB)  TX bytes:2824 (2.8 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:
0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@ubuntu:/home/sdn# iperf -c 10.0.0.2
------------------------------------------------------------
Client connecting to 10.0.0.2, TCP port 5001
TCP window size:  340 KByte (default)
------------------------------------------------------------
[ 12] local 10.0.0.1 port 36030 connected with 10.0.0.2 port
5001
[ ID] Interval       Transfer     Bandwidth
[ 12]  0.0-10.0 sec  64.5 GBytes  55.4 Gbits/sec
root@ubuntu:/home/sdn#
```

"host: h2"

```
root@ubuntu:/home/sdn# ifconfig
h2-eth0   Link encap:Ethernet  HWaddr 00:00:00:00:00:02
          inet addr:10.0.0.2  Bcast:10.255.255.255  Mask:255.
0.0.0
          inet6 addr: fe80::200:ff:fe00:2/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:471 errors:0 dropped:184 overruns:0 fram
e:0
          TX packets:32 errors:0 dropped:0 overruns:0 carrier
:0
          collisions:0 txqueuelen:1000
          RX bytes:58640 (58.6 KB)  TX bytes:2824 (2.8 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:
0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@ubuntu:/home/sdn# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 13] local 10.0.0.2 port 5001 connected with 10.0.0.1 port 3
6030
[ ID] Interval       Transfer     Bandwidth
[ 13]  0.0-10.0 sec  64.5 GBytes  55.4 Gbits/sec
```

Imagen 5.1.11: test de ancho de banda entre h1 y h2 haciendo uso de iperf.

5.2. Topología compleja proporcionada por Mininet

Se abren tres terminales y se consigue acceso super usuario en cada una de ellas. Iniciamos POX y POSDesk, Mininet y Wireshark.

En el primero se va a lanzar POX y POSDesk con el siguiente comando

```
./pox/pox.py samples.pretty_log web messenger
messenger.log_service messenger.ajax_transport openflow.of_service
openflow.webservice poxdesk openflow.discovery poxdesk.tinytopo
forwarding.l2_pairs
```

En el segundo se lanza Mininet con una de las topologías predefinidas, en este caso una topología en árbol, con tres niveles de profundidad, tres switches por subnivel y tres hosts en cada switch final mediante el comando

```
mn -c //limpiar posibles datos de ejec. anteriores
mn --topo tree,3,3 --controller remote -x -mac
```

y que asigna las MAC 00:00:00:00:00:01 a 00:00:00:00:00:1b y las IPs 9.0.0.1 a 9.0.0.27 respectivamente a los hosts entre h1 y h27.

Y en el tercero se inicia Wireshark con

```
wireshark
```

La topología desplegada es al siguiente:

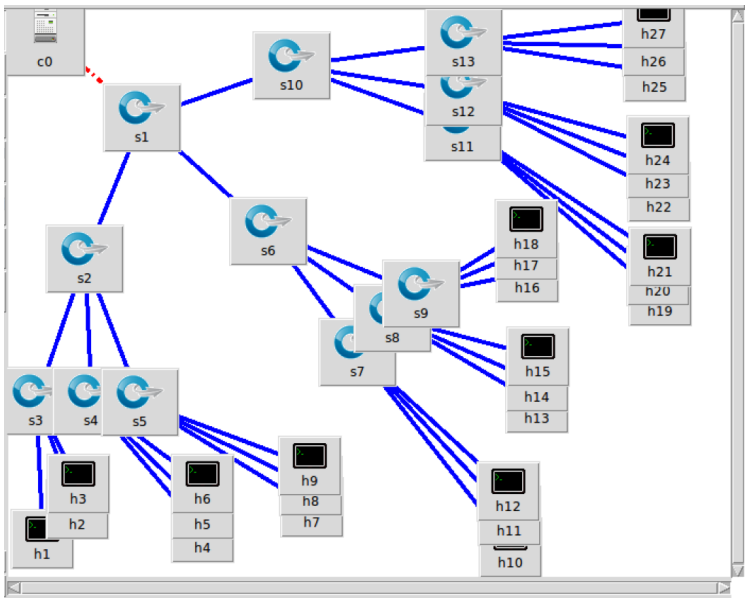


Imagen 5.2.1: topología tree,3,3

Si capturamos paquetes desde el inicio de la conexión podemos comprobar que existe un intercambio de paquetes OpenFlow entre los 13 switches y c0 para establecer los parámetros de la conexión e identificar los posibles datos iniciales con los que cuentan ambos.

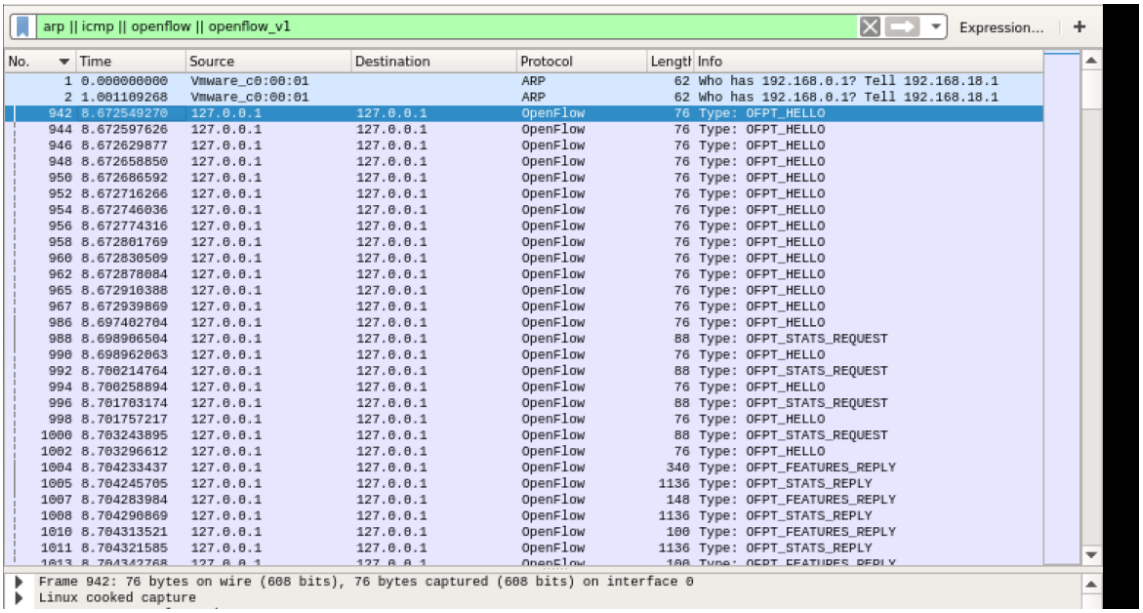


Imagen 5.2.2: establecimiento conexión entre switches y controlador.

y realiza un chequeo constante del estado de la red para descubrir los posibles cambios que ocurren en la red.

En el momento del despliegue de la red la tabla de flujo de los switches de la red se encuentra vacía. Lo podemos comprobar eligiendo el switch s1 (00-00-00-00-00-01) en TableViewer y vemos que se encuentra vacía. Lo único que contiene es la entrada de los mensajes LLDP que son los usados para el descubrimiento de la topología.

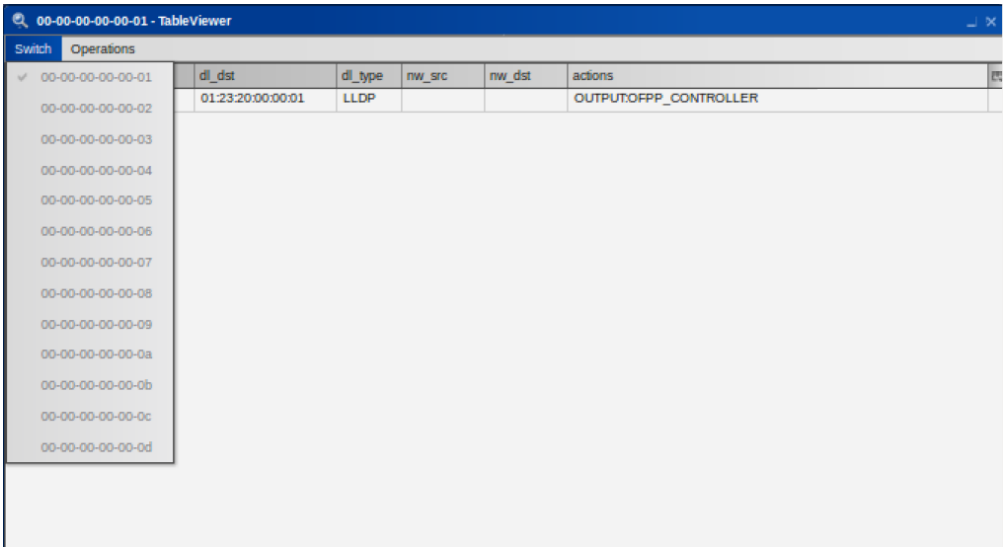


Imagen 5.2.3: tabla de flujo al inicio de la simulación.

En la sección TopoViewer se puede ver que se cuenta con 13 switches, aunque los hosts no se muestren como sí pasa en Mininet. Además, el enlace s1-s10 no se muestra en POXDesk,

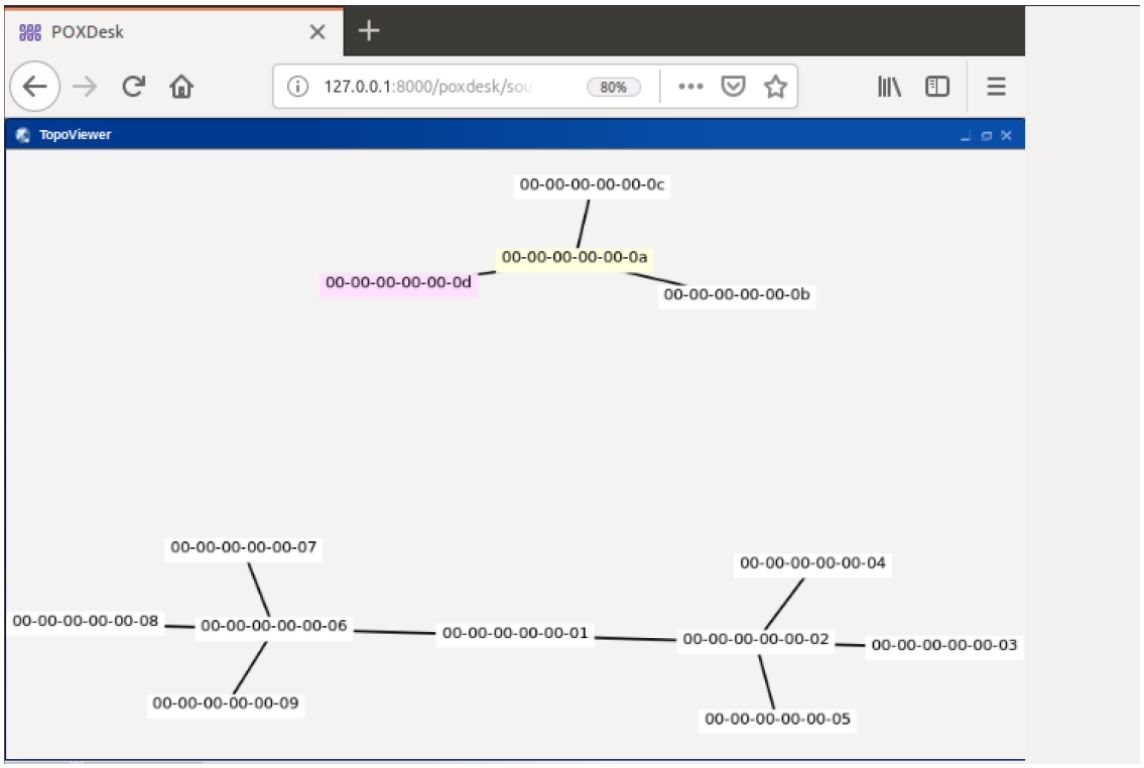


Imagen 5.2.4: topología mostrada en POXDesk.

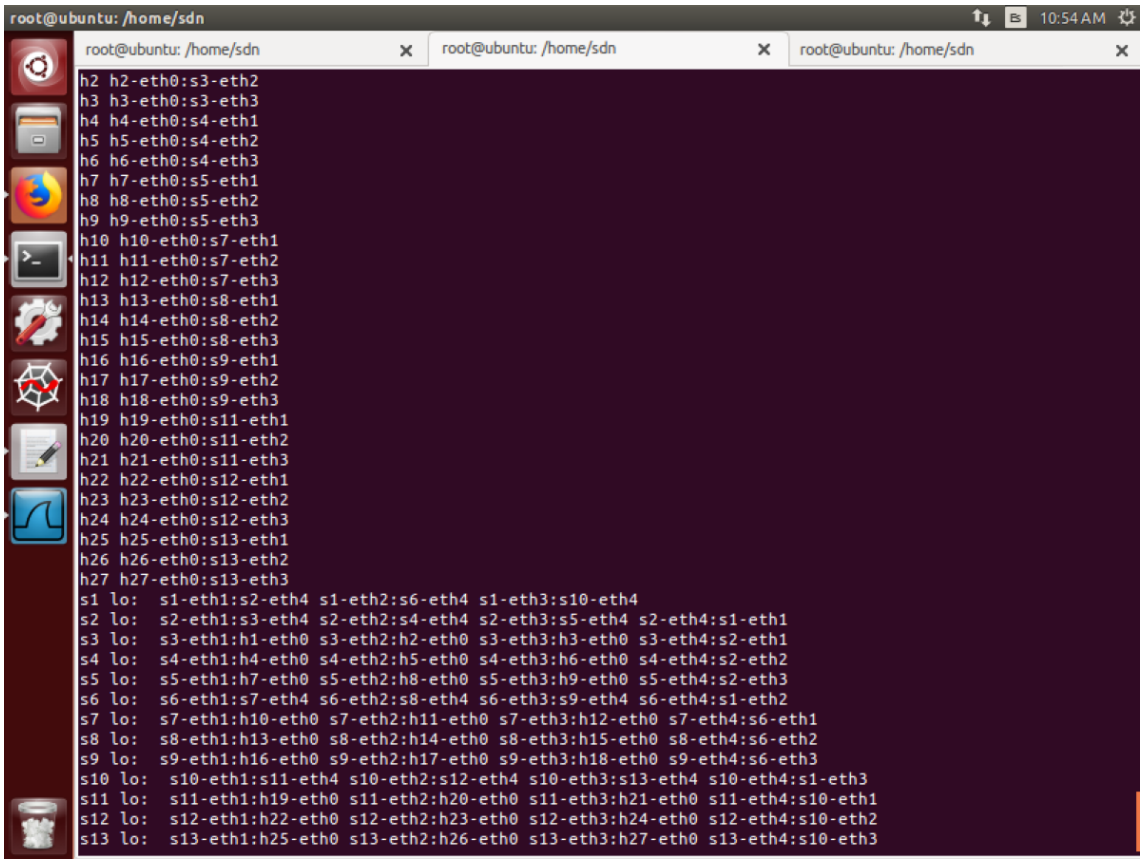


Imagen 5.2.5: salida al comando net en Mininet.

mientras que en Mininet aparecen 27 hosts, h1 - h27, conectados con los switches s1 a s13 según la siguiente tabla

host	h1	h2	h3	h4	h5	h6	h7	h8	h9	h10	h11
switch	s3	s3	s3	s4	s4	s4	s5	s5	s5	s7	s7
puerto host	eth0	eth0	eth0	eth0	eth0	eth0	eth0	eth0	eth0	eth0	eth0
puerto switch	eth1	eth2	eth3	eth1	eth2	eth3	eth1	eth2	eth3	eth1	eth2

	h12	h13	h14	h15	h16	h17	h18	h19	h20	h21	h22
	s7	s8	s8	s8	s9	s9	s9	s11	s11	s11	s12
	eth0	eth0	eth0	eth0	eth0	eth0	eth0	eth0	eth0	eth0	eth0
	eth3	eth1	eth2	eth3	eth1	eth2	eth3	eth1	eth2	eth3	eth1

	h23	h24	h25	h26	h27
	s12	s12	s13	s13	s13
	eth0	eth0	eth0	eth0	eth0
	eth2	eth3	eth1	eth2	eth3

Imagen 5.2.6: tabla de conexión entre hosts y switches

y un controlador c0.

Al realizar un ping desde h1 a h17 se ve que el tiempo de respuesta del primer paquete es muy superior al resto, cosa que al realizar después el ping de manera inversa esto no ocurre.

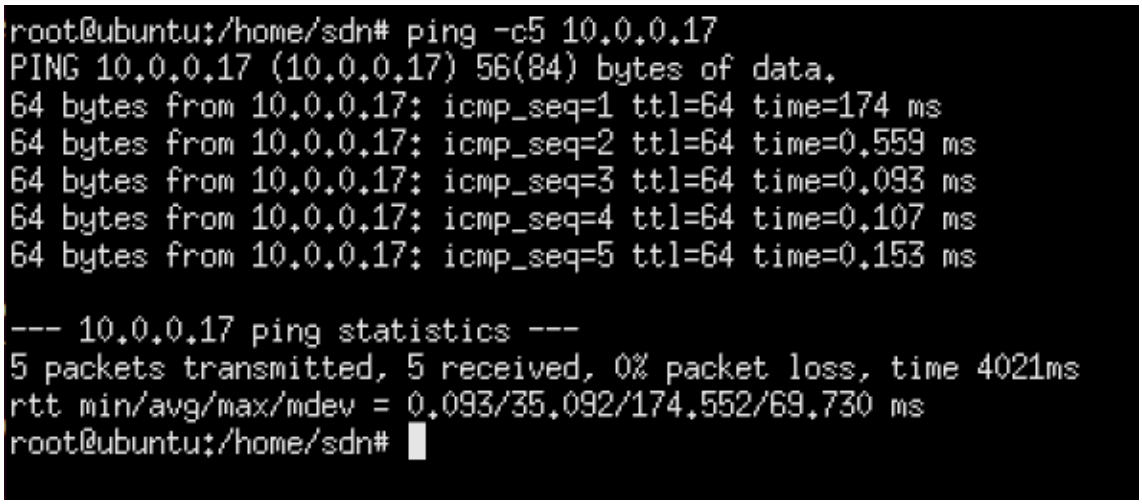


Imagen 5.2.7: primera prueba de ping de h1 a h17.

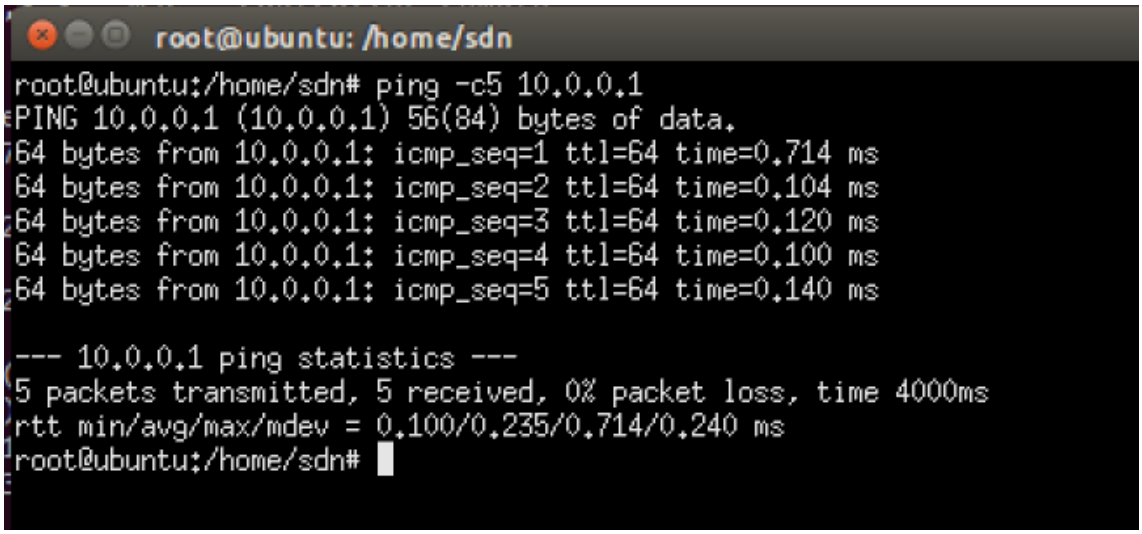


Imagen 5.2.8: ping posterior de h17 a h1.

Como podemos comprobar, la latencia de primer mensaje la primera vez que se realiza un ping es muy superior, del orden de unas 300 veces (174 ms frente a 0,559 ms), y el segundo mensaje sigue teniendo una latencia superior al resto (0,559 ms frente a una media de 118 ms).

Realizamos una comprobación de la conectividad entre los hosts mediante un

```
pingall
```

, lo que arroja el siguiente resultado

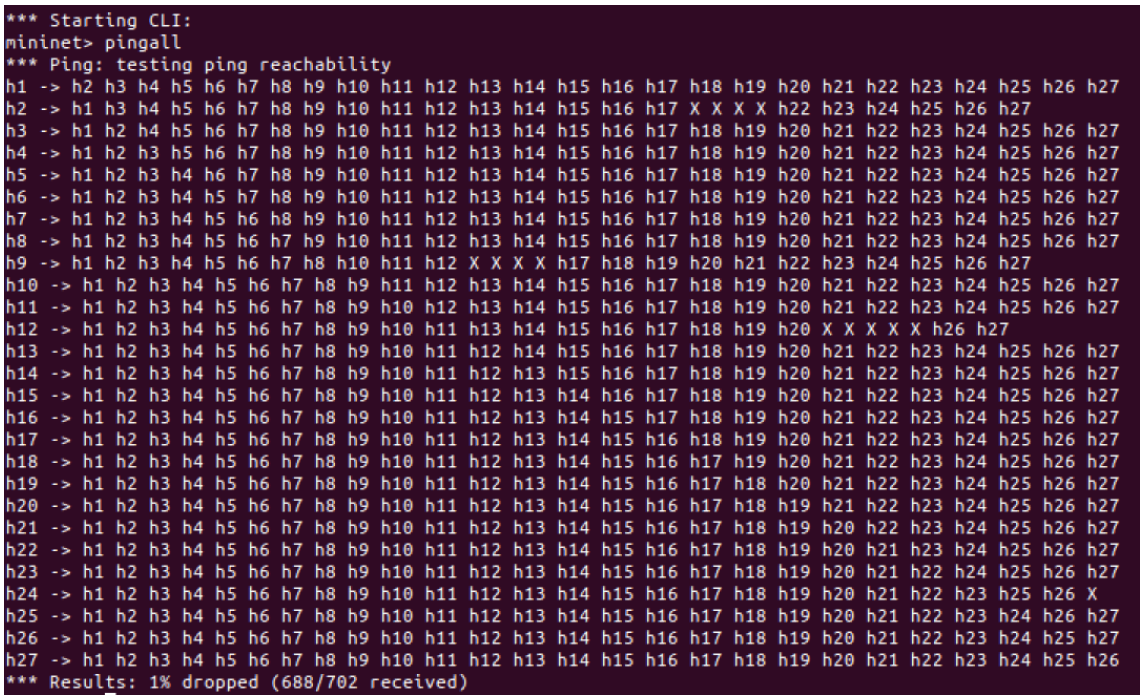


Imagen 5.2.9: salida al comando pingall.

Si realizamos un segundo test, con las tablas de flujo actualizadas en cada uno de los switches, el resultado es bastante diferente.

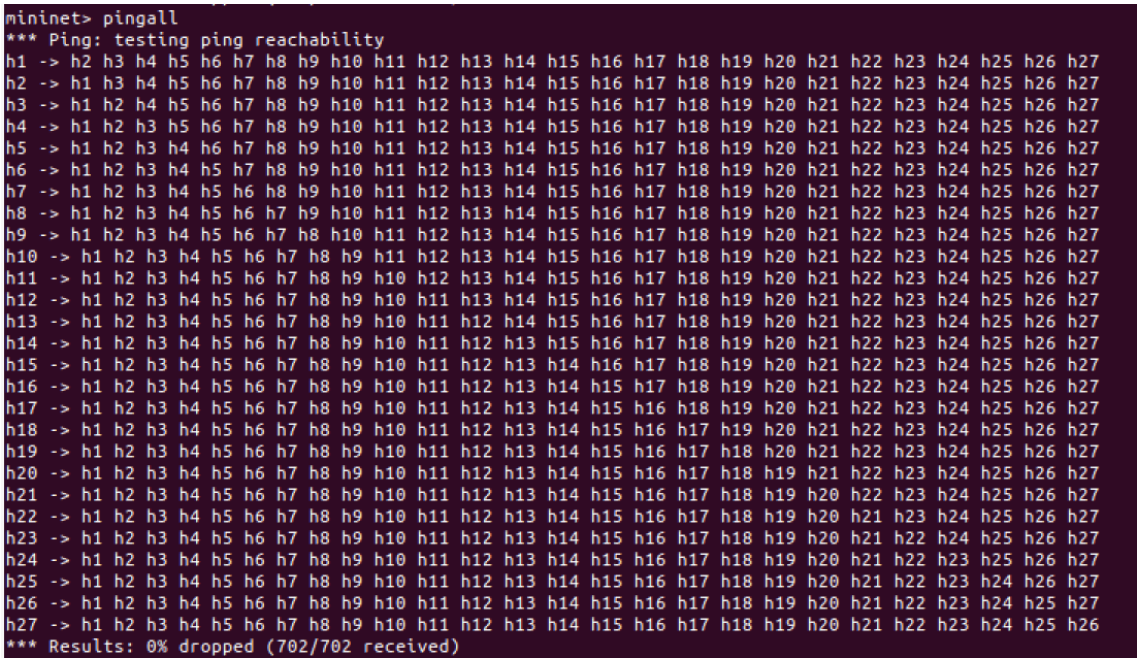


Imagen 5.2.10: salida comando pingall lanzado por segunda vez.

Como podemos observar, la primera vez se reciben solo 688 de 702 paquetes enviados, mientas que las segunda hay conectividad completa en toda la red.

Si comprobamos la tabla de flujo de s1, descubrimos que cuenta con un total de 487 entradas en las que se indica la acción a realizar con los paquetes según su dirección origen y destino.

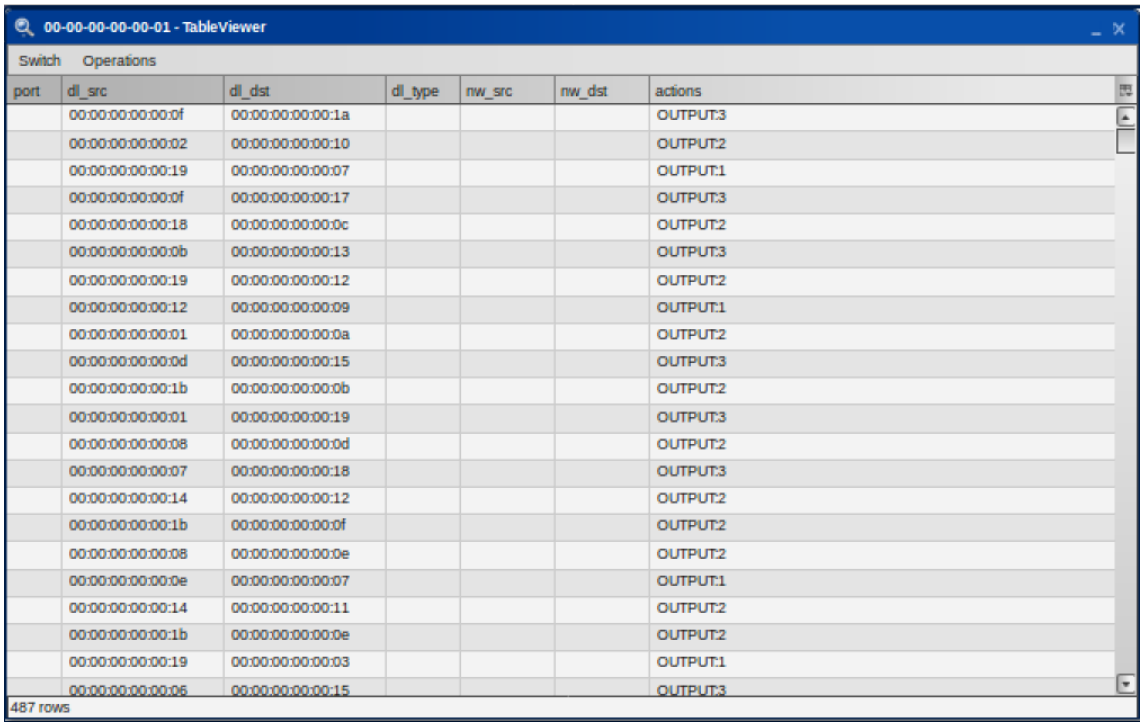


Imagen 5.2.11: TableViewer s1.

Si observamos Wireshark, podemos observar que se realiza uma inundación de la red con paquetes ARP e ICMP iniciada por h1 para obtener la dirección ethernet de h17 y poder establecer la comunicación. Debido a la gran cantidad de equipos en la red y la topología tan compleja que se presenta, existen una gran cantidad de paquetes que se mueven en la red para conseguir la comunicación entre los extremos.

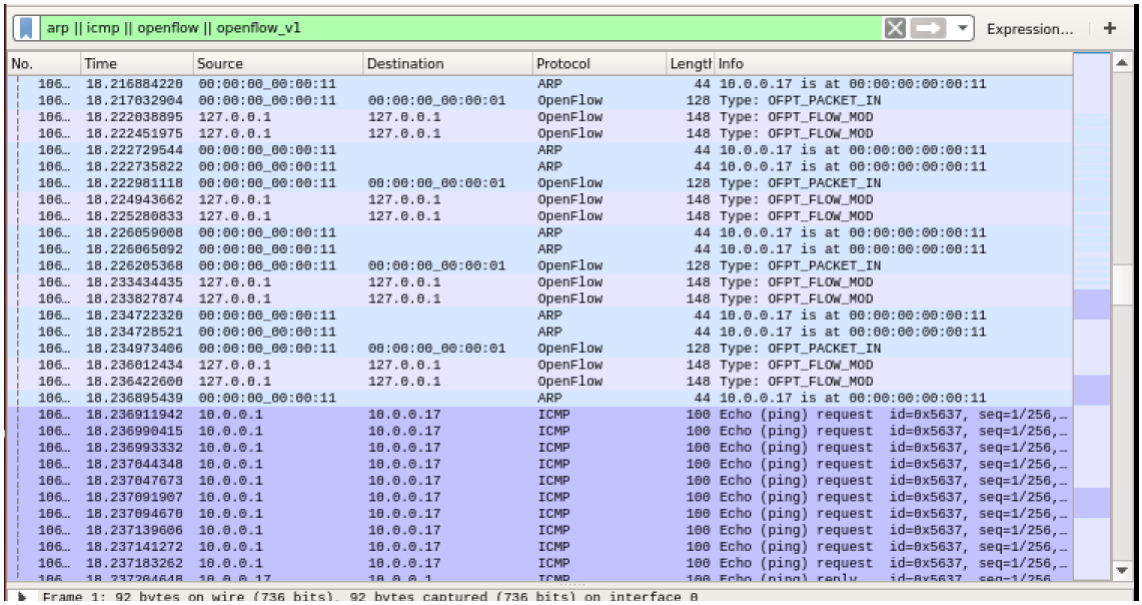


Imagen 5.2.12: ARP e ICMP en Wireshark

En este caso el camino realizado por el paquete ICMP sería:

h1 --> s3 --> s2 --> s1 --> s6 --> s9 --> h17.

La segunda vez que se realiza el ping los switches ya cuentan con las tablas de flujo rellenas, h1 cuenta con h17 en su tabla ARP y ya no es necesario inundar la red con mensajes ARP.

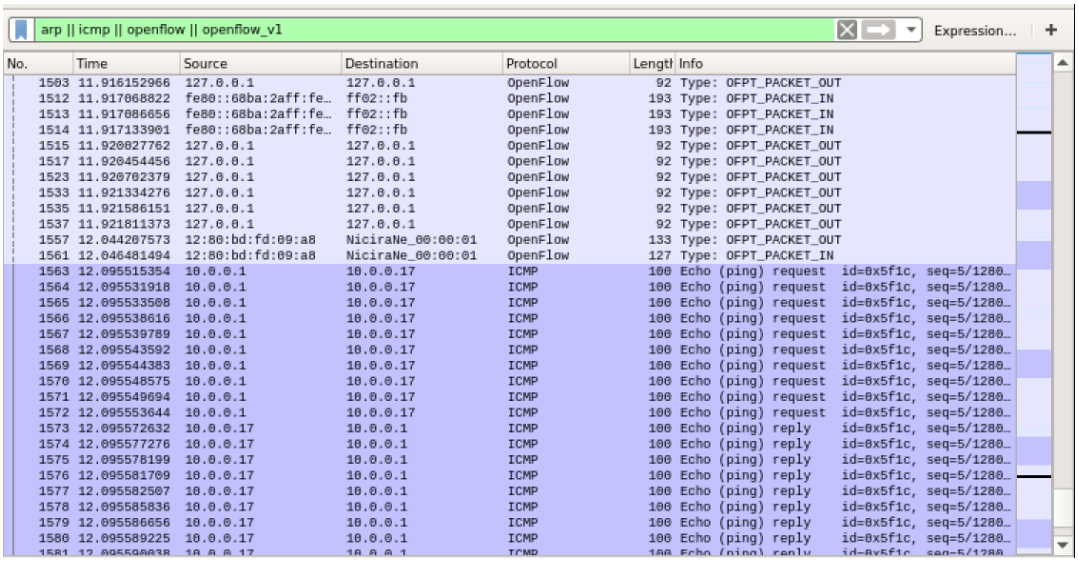


Imagen 5.2.13: Mensajes ICMP en Wireshark

Para comprobar el anche de banda entre h1 y h17 se lanzan dos consolas xterm, una para cada máquina, se lanza iperf como servidor en h17 y se conecta h1 como cliente. En este caso, el ancho de banda está en el entorno de los 6.5 Gbps

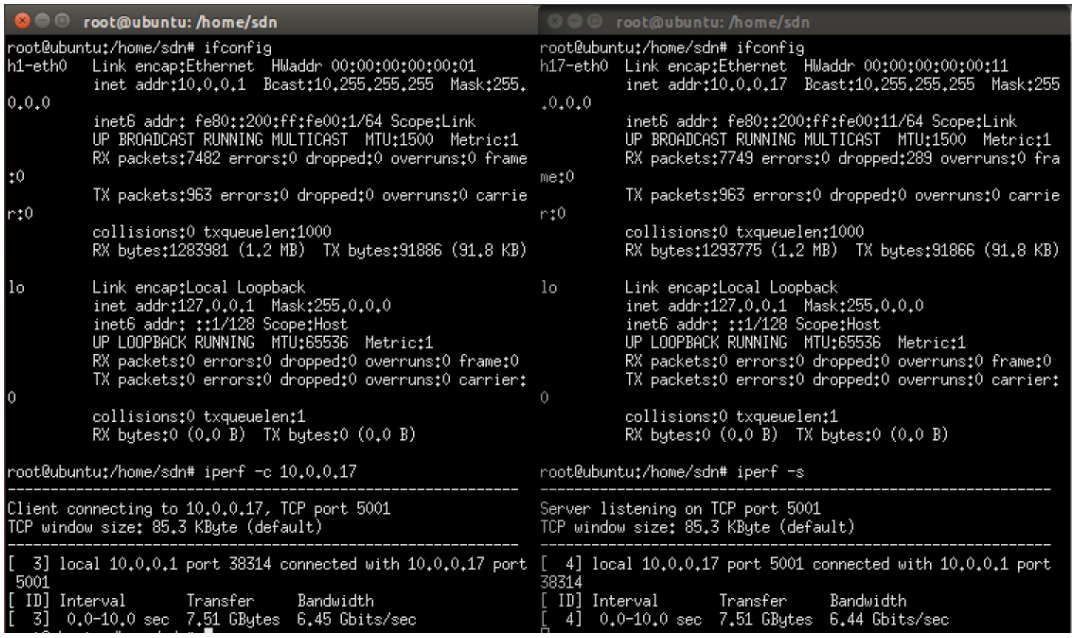


Imagen 5.2.14: iperf entre h1 y h17

5.3 Topología creada con Miniedit.

Para la siguiente topología es necesario lanzar Miniedit, el entorno gráfico de Mininet para la construcción y edición de topologías de forma sencilla. Vamos a desarrollar un esquema que cuanta con cuatro switches, todos ellos unidos al controlador, y a los que se conectaran dos hosts a cada uno.

Lanzamos Miniedit con el comando

```
./mininet/examples/miniedit.py
```

y se abre una nueva ventana para la edición de la red.

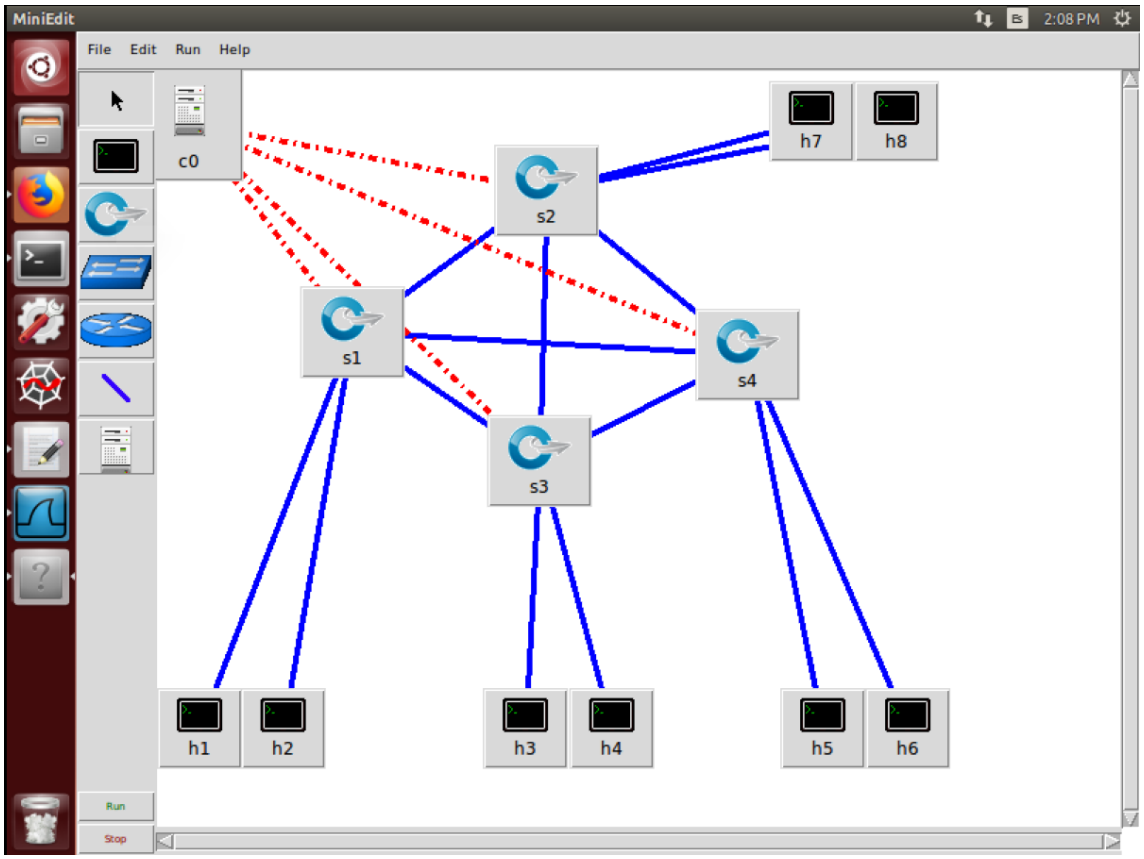


Imagen 5.3.1: Topología creada con Miniedit.

Editamos las características del controlador para que se conecte a POX y poder realizar las pruebas oportunas. Para ello debemos editar el parámetro “Controller Type” y seleccionar “Remote Controller”, la IP con a la que se deben de realizar las peticiones y el puerto en el que se esperan los mensajes.

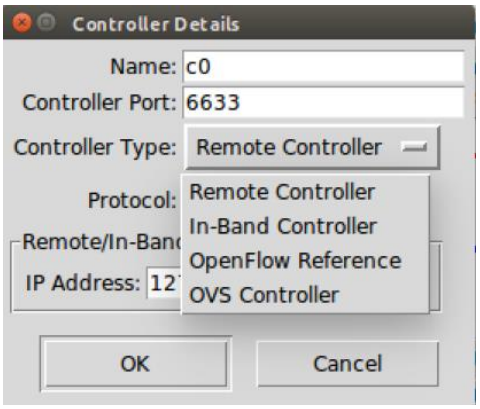


Imagen 5.3.2:Edición del controlador.

A continuación, editamos los enlaces para asemejarlos a una red algo más real, en la que los enlaces entre switches cuentan con un ancho de banda de 500 Mbps y un retardo de 2 ms y los enlaces entre host y switch cuentan con un ancho de banda de 100 Mpbs y un retardo de 5 ms. Para ello hacemos click derecho en el enlace y pulsamos sobre "Properties" y añadimos los parámetros como en las imágenes 6.3.3 y 6.3.4.

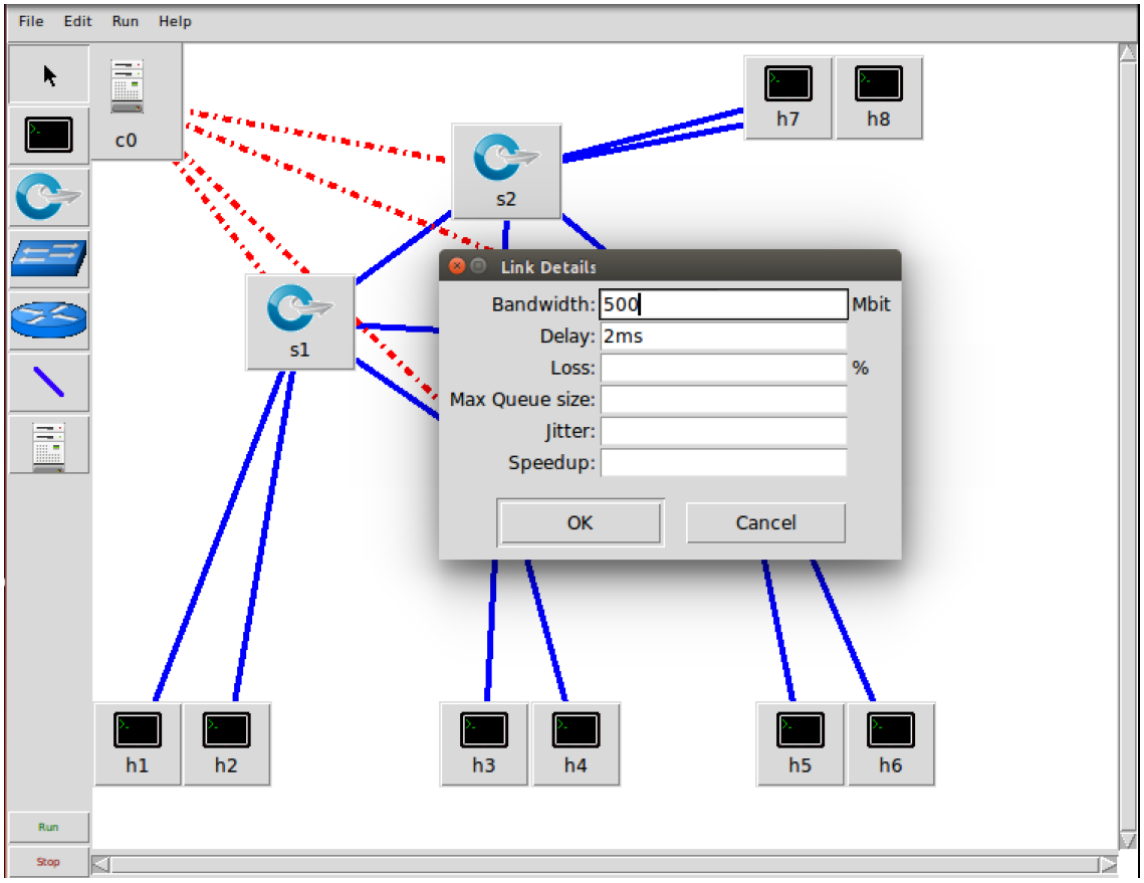


Imagen 5.3.3: Propiedades links entre switches.

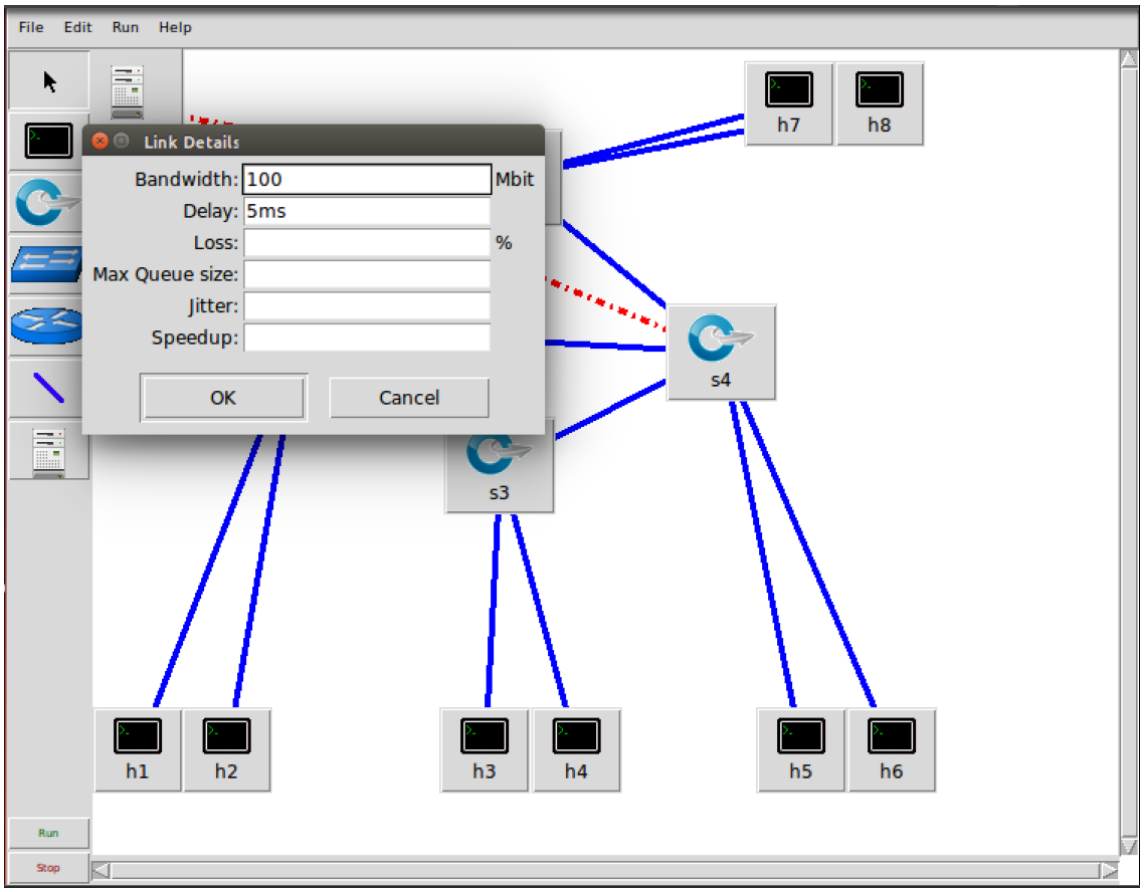


Imagen 5.3.4: Propiedades links entre host y switch.

Como podemos observar, este cambio de configuración en los enlaces produce una salida en el CLI de Mininet, advirtiendo de los detalles de los enlaces creados.

```
root@ubuntu:/home/sdn# ./mininet/examples/miniedit.py
topo=none
New link details = {'delay': '5ms', 'bw': 100}
New link details = {'delay': '5ms', 'bw': 100}
New link details = {'delay': '5ms', 'bw': 100}
New link details = {'delay': '5ms', 'bw': 100}
New link details = {'delay': '5ms', 'bw': 100}
New link details = {'delay': '5ms', 'bw': 100}
New link details = {'delay': '5ms', 'bw': 100}
New link details = {'delay': '5ms', 'bw': 100}
New link details = {'delay': '2ms', 'bw': 500}
New link details = {'delay': '2ms', 'bw': 500}
New link details = {'delay': '2ms', 'bw': 500}
New link details = {'delay': '2ms', 'bw': 500}
New link details = {'delay': '2ms', 'bw': 500}
New link details = {'delay': '2ms', 'bw': 500}
```

Imagen 5.3.5: Salida Mininet.

Para guardar la topología y poder simularla en cualquier momento se debe exportar en un archivo ".py" en "File/Export Level 2 Script". Sin embargo, este archivo no puede ser abierto y editado de nuevo en Miniedit. Para poder manipular de nuevo la topología se debe pulsar en "save", que genera un archivo ".mn"

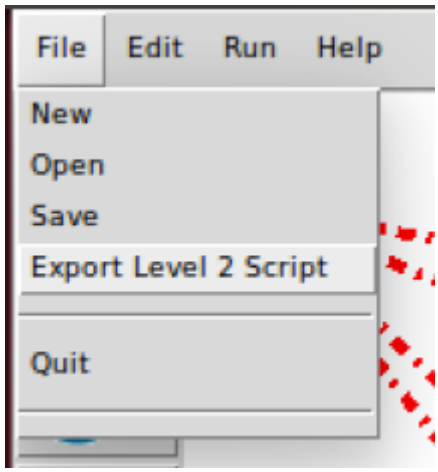


Imagen 5.3.6: Exportar archivos en Miniedit

Se abren tres terminales y se consigue acceso super usuario en cada una de ellas. Iniciamos POX y POSDesk, Mininet y Wireshark.

En el primero se va a lanzar POX y POSDesk con el siguiente comando

```
./pox/pox.py samples.pretty_log web messenger
messenger.log_service messenger.ajax_transport openflow.of_service
openflow.webservice poxdesk openflow.discovery poxdesk.tinytopo
forwarding.l2_pairs openflow.spanning_tree
```

Añadimos "openflow.spanning_tree" ya que la topología cuenta con redundancia en los enlaces entre swicthes. En caso de no iniciarlo van a aparecer mensajes de error y no se va a poder realizar ninguna prueba en la red.

Para iniciar la red en este caso es algo diferente, ya que se trata de un script python que describe una topología y se encarga de iniciar la propia Mininet. Por tanto de debe iniciar con el siguiente comando

```
mn -c //limpiar posibles datos de ejec. anteriores
python ./mininet/custom/topo4s8h.py
```

En el tercer terminal se inicia Wireshark con

```
wireshark
```

Si capturamos paquetes desde el inicio de la conexión podemos comprobar que existe un intercambio de paquetes OpenFlow entre los switches y c0 para establecer los parámetros de la conexión e identificar los posibles datos iniciales con los que cuentan ambos.

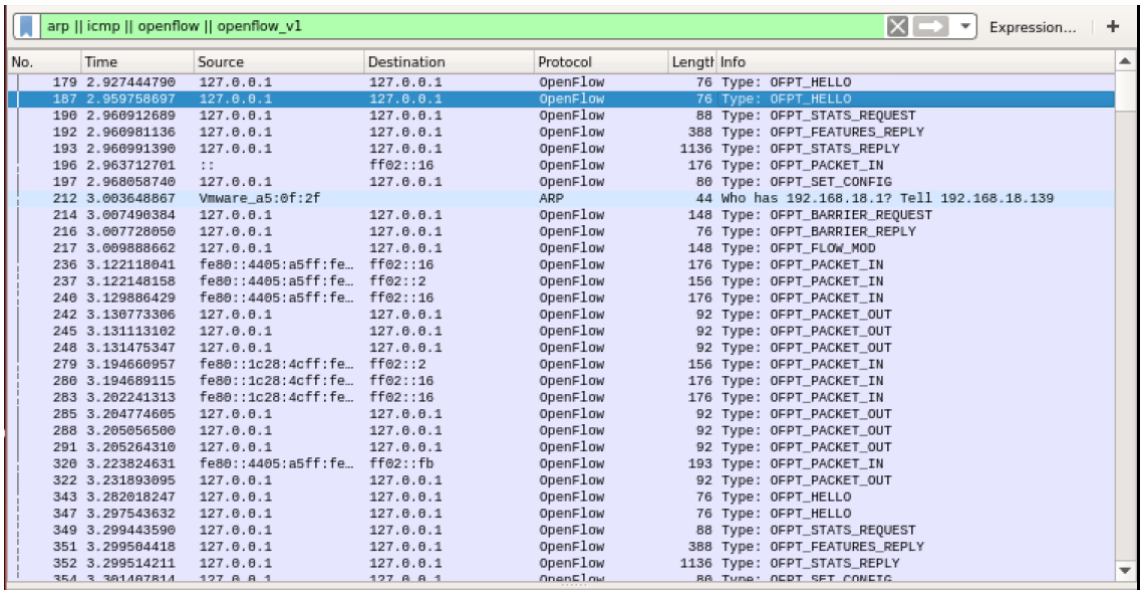


Imagen 5.3.7: establecimiento conexión entre switch y controlador.

y realiza un chequeo constante del estado de la red para descubrir los posibles cambios que ocurren en la red.

En el momento del despliegue de la red la tabla de flujo del switch s1 se encuentra vacía como se puede consultar en el TableViewer de s1 (00-00-00-00-00-01)

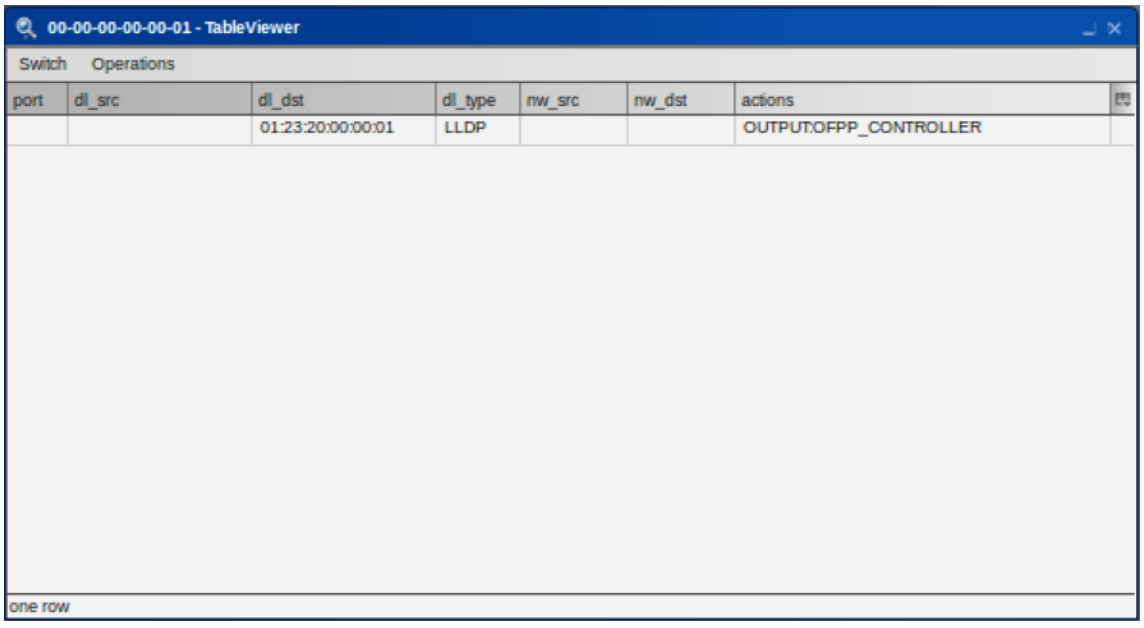


Imagen 5.3.8: tabla de flujo de s1 al inicio de la simulación.

En la sección TopoViewer se puede ver que se cuenta con un switch, aunque los hosts no se muestren como sí pasa en Mininet.

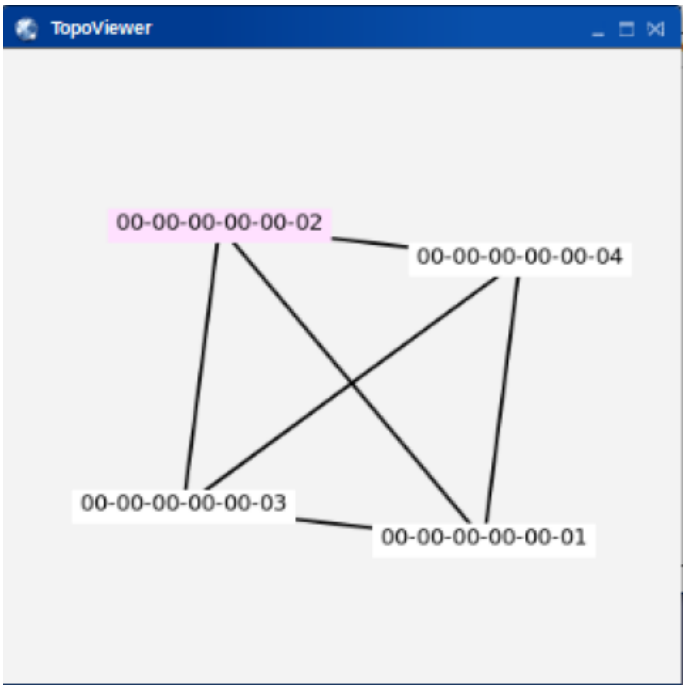


Imagen 5.3.9: topología mostrada en POXDesk

```
mininet> net
h6 h6-eth0:s3-eth5
h1 h1-eth0:s1-eth4
h7 h7-eth0:s4-eth4
h4 h4-eth0:s2-eth5
h5 h5-eth0:s3-eth4
h8 h8-eth0:s4-eth5
h3 h3-eth0:s2-eth4
h2 h2-eth0:s1-eth5
s2 lo:  s2-eth1:s1-eth1 s2-eth2:s3-eth1 s2-eth3:s4-eth3 s2-eth4:h3-eth0 s2-eth5:h4-eth0
s3 lo:  s3-eth1:s2-eth2 s3-eth2:s4-eth1 s3-eth3:s1-eth3 s3-eth4:h5-eth0 s3-eth5:h6-eth0
s4 lo:  s4-eth1:s3-eth2 s4-eth2:s1-eth2 s4-eth3:s2-eth3 s4-eth4:h7-eth0 s4-eth5:h8-eth0
s1 lo:  s1-eth1:s2-eth1 s1-eth2:s4-eth2 s1-eth3:s3-eth3 s1-eth4:h1-eth0 s1-eth5:h2-eth0
c0
mininet>
```

Imagen 5.3.10: salida al comando net en Mininet

donde se ve que existen ocho hosts, de h1 a h8, conectados a cuatro switches como se muestra en la tabla

host	h1	h2	h3	h4	h5	h6	h7	h8
switch	s1	s1	s2	s2	s3	s3	s4	s4
puerto host	eth0	eth0	eth0	eth0	eth0	eth0	eth0	eth0
puerto switch	eth4	eth5	eh4	eth5	eth4	eth5	eth4	eth5

Imagen 5.3.11: tabla de conexión entre hosts y switches

y un controlador c0.

Al realizar un ping desde h2 a h5 se ve que el tiempo de respuesta del primer paquete es muy superior al resto, 161 ms frente a 30 ms.

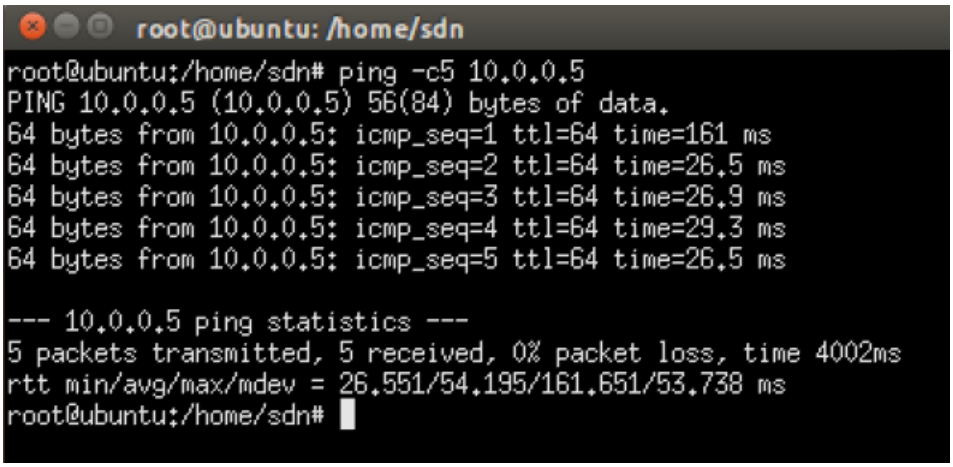


Imagen 5.3.12: ping de h2 a h5

Al realizar el ping de forma inversa vemos como eso no pasa.

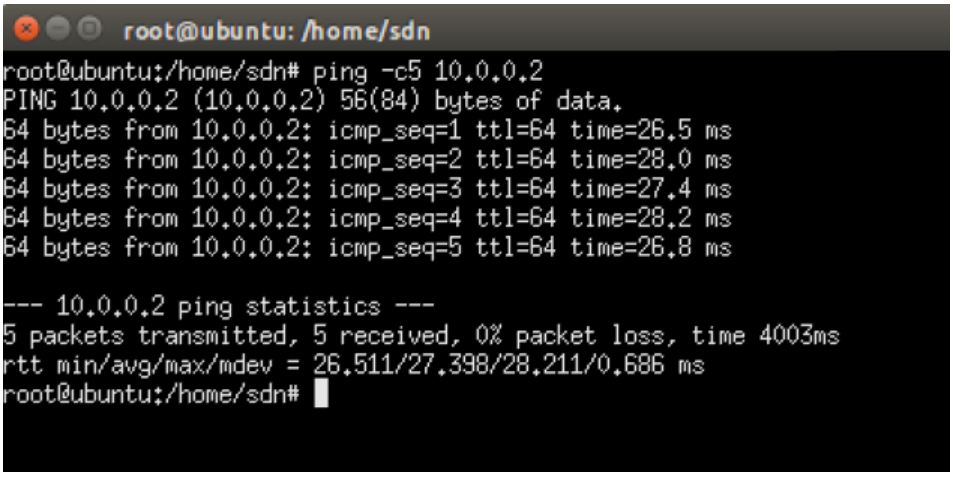


Imagen 5.3.13: ping de h5 a h2

En ambas ocasiones, el ping se estabiliza entorno a 28 ms. Esto se debe a la inclusión de los retardos en la configuración de los enlaces, propiciando unos resultados más acordes a lo que sucedería de realizar estar pruebas en un entorno real.

Una vez realizado este intercambio de mensajes, la tabla de s1 cuenta con las entradas necesarias para futuras comunicaciones.

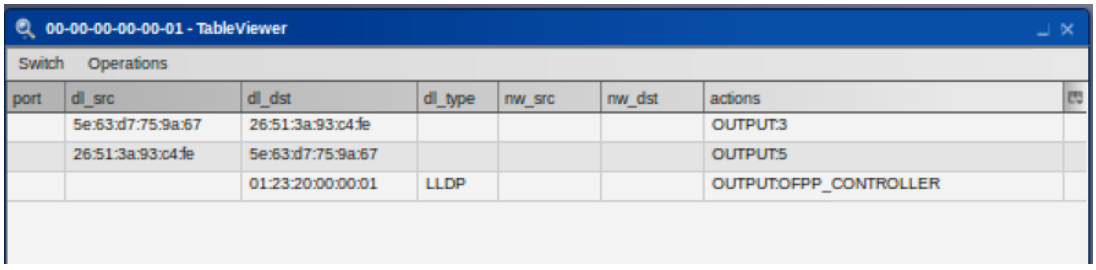


Imagen 5.3.14: tabla de flujo de s1 actualizada.

Como podemos observar en Wireshark, a fin de obtener la dirección ethernet correspondiente a la dirección IP requerida, en este caso 9.0.0.5, se lanza un ARP en la que h5 responde que a la IP 9.0.0.5 le corresponde la dirección 26:51:3a:93:c4:fe. En ese momento s1 solicita saber si hay modificaciones en las tablas de flujo mediante un mensaje OFPT_FLOW_MOD, al cual el controlador envía una actualización y ya se produce el intercambio de mensajes ICMP (ping).

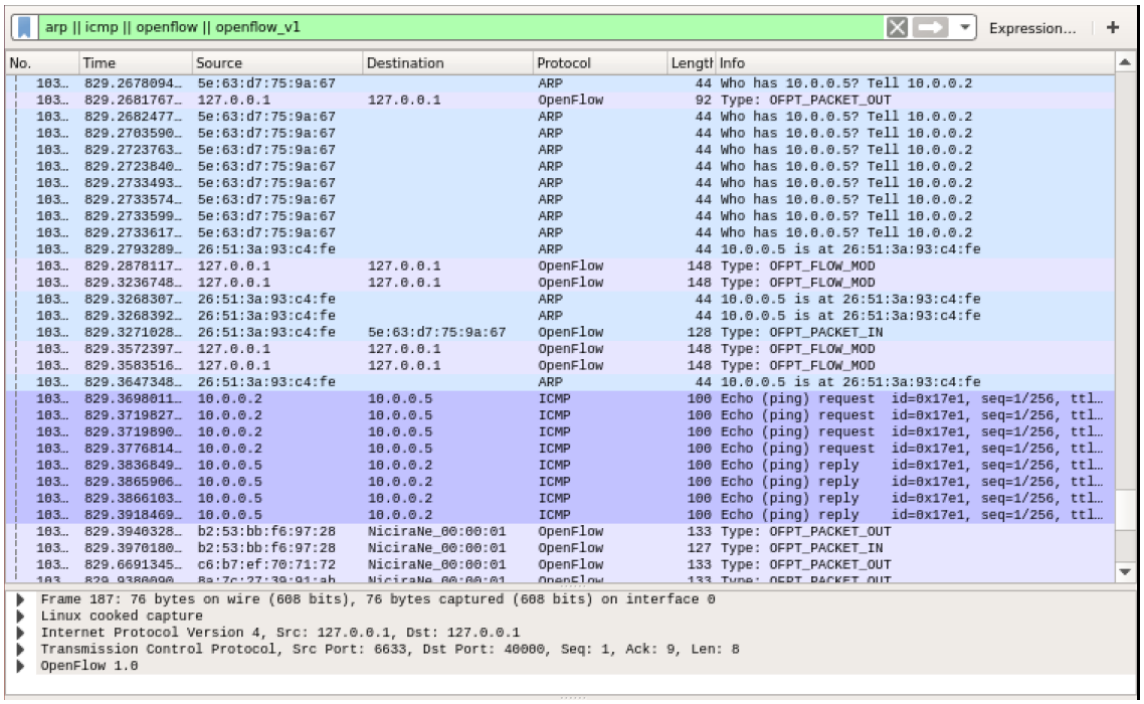


Imagen 5.3.15: primer ping de h2 a h5.

Al realizar el proceso de nuevo comprobamos que ya no es necesario el uso de ARP, ya que h2 cuenta con la dirección ethernet de h5 en su tabla ARP y realiza directamente el ping.

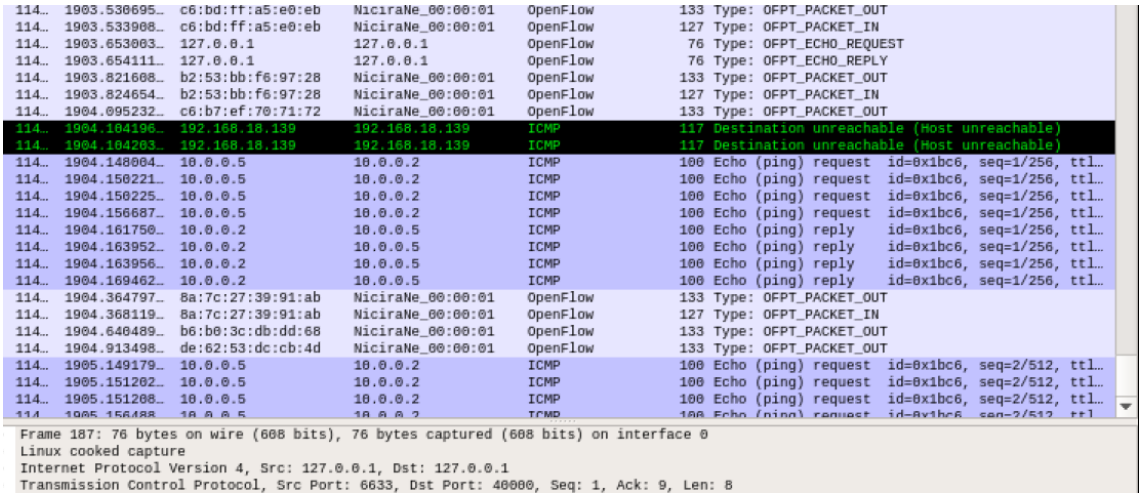


Imagen 5.3.16: ping de h5 a h2.

Para la comprobación del ancho de banda en la comunicación entre h2 y h5, levantamos iperf en h5 como servidor y realizamos un test desde h2 como cliente.

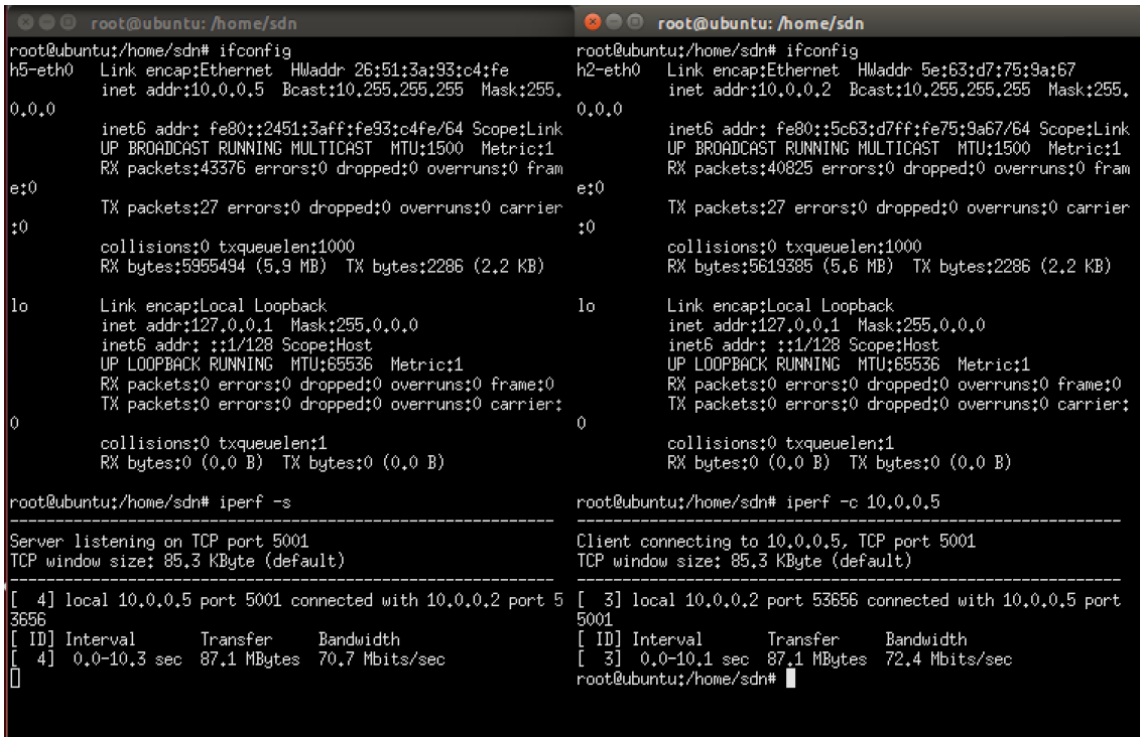


Imagen 5.3.17: Test de ancho de banda entre h2 y h5.

Como podemos observar en la imagen anterior, el ancho de banda está en torno a los 70 Mbps. Esto se debe a la limitación de 100 Mbps introducida en el enlace y a la saturación presente al hacer el test.

Como consideración, la limitación impuesta a los enlaces supone un retraso en el inicio de las pruebas, ya que necesita un tiempo grande, mayor al minuto, para el descubrimiento total de la red y su consecuente puesta a punto.

5.4. Topología programada en Python

Como último paso de las pruebas, se programa una topología directamente en Python, la cuál va a contar con un total de ocho equipos intermedios, 15 hosts y un controlador externo (POX).

El código desarrollado para establecer la topología se puede consultar en el ANEXO IV.

La topología creada es la que se puede observar en la siguiente imagen creada para la fácil compresión de la misma.

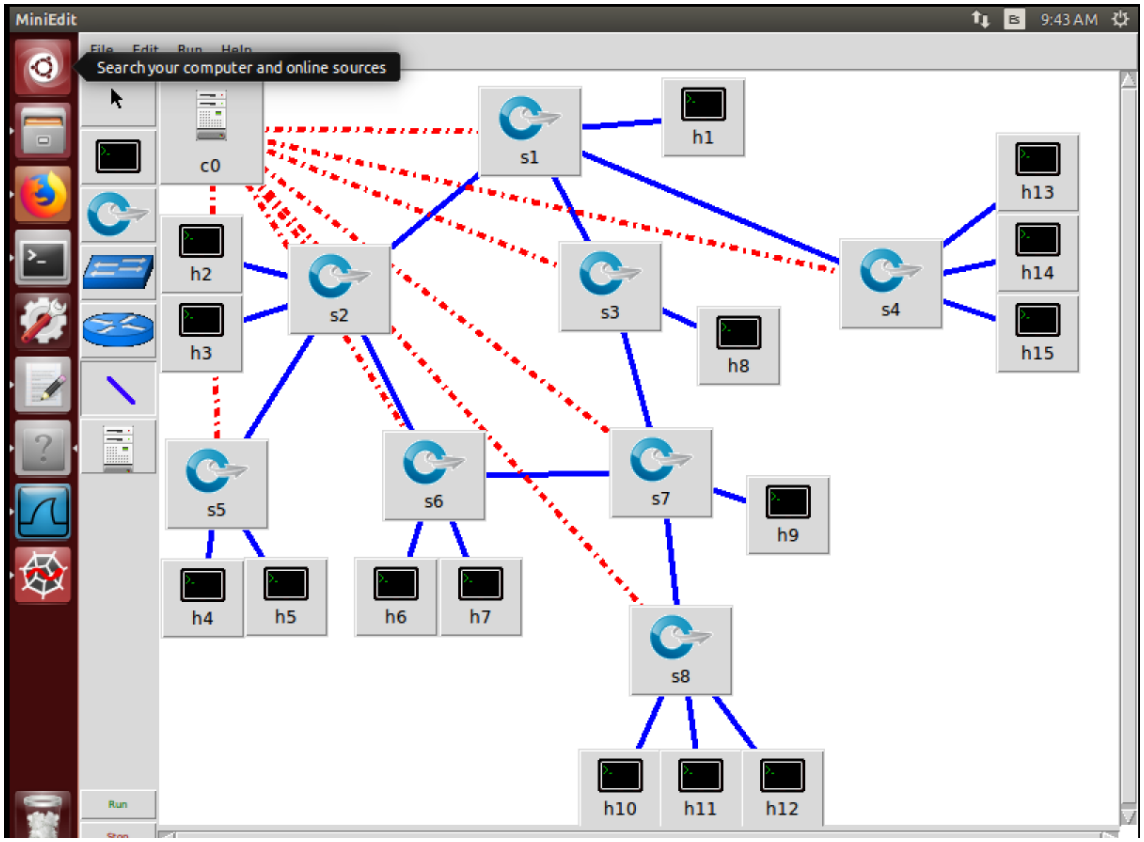


Imagen 5.4.1: topología recreada en Miniedit.

Se inicia una primera consola para lanzar POX y POXDesk mediante el comando

```
./pox/pox.py samples.pretty_log web messenger
messenger.log_service messenger.ajax_transport openflow.of_service
openflow.webservice poxdesk openflow.discovery poxdesk.tinytopo
forwarding.l2_pairs openflow.spanning_tree
```

Esta topología se ha guardado en la ruta /home/sdn, por lo que la forma de invocar la topología es la siguiente

```
python topoProgramada.py
```

Los equipos de h1 a h15 obtienen las IPs de 9.0.0.1 a 9.0.0.15 y las MAC 00:00:00:00:00:01 a 00:00:00:00:00:0f respectivamente.

En una tercera terminal iniciamos Wireshark con

```
wireshark
```

Si capturamos paquetes desde el inicio de la conexión podemos comprobar que existe un intercambio de paquetes OpenFlow entre todos los switches y c0 para establecer los parámetros de la conexión e identificar los posibles datos iniciales con los que cuentan ambos.

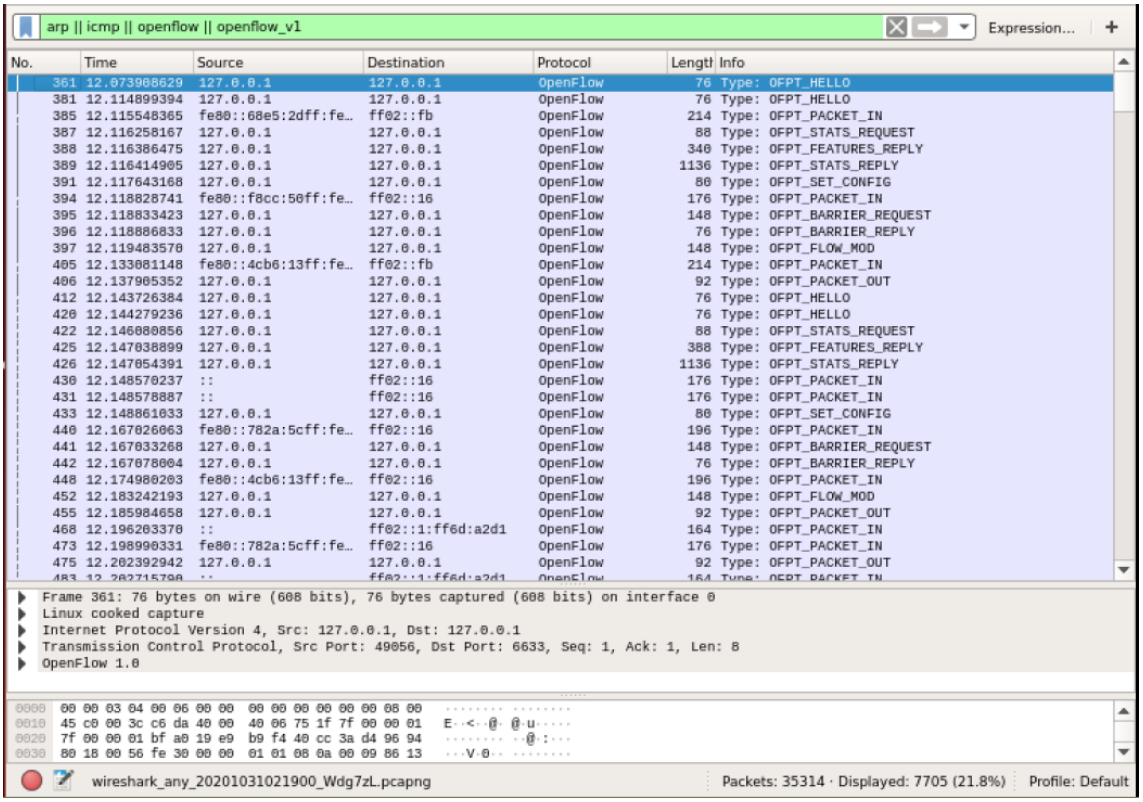


Imagen 5.4.2: establecimiento conexión entre los switches y el controlador

y realiza un chequeo constante del estado de la red para descubrir los posibles cambios que ocurren en la red.

En el momento del despliegue de la red las tablas de flujo de los switches está vacía. Esto se puede comprobar consultando de flujo del switch s1, que se encuentra vacía como vemos en el TableViewer de s1 (00-00-00-00-00-01).

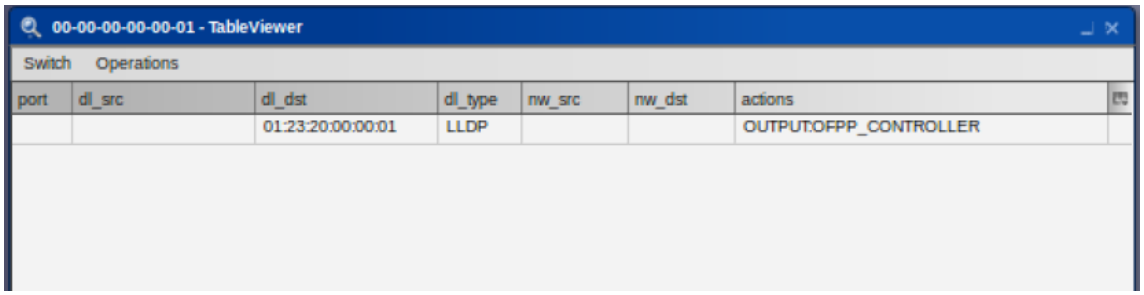


Imagen 5.4.3: TableViewer de s1 al inicio de la simulación.

En la sección TopoViewer podemos ver que se cuenta con ocho equipos intermedios, aunque los hosts no se muestren como sí pasa en Mininet,

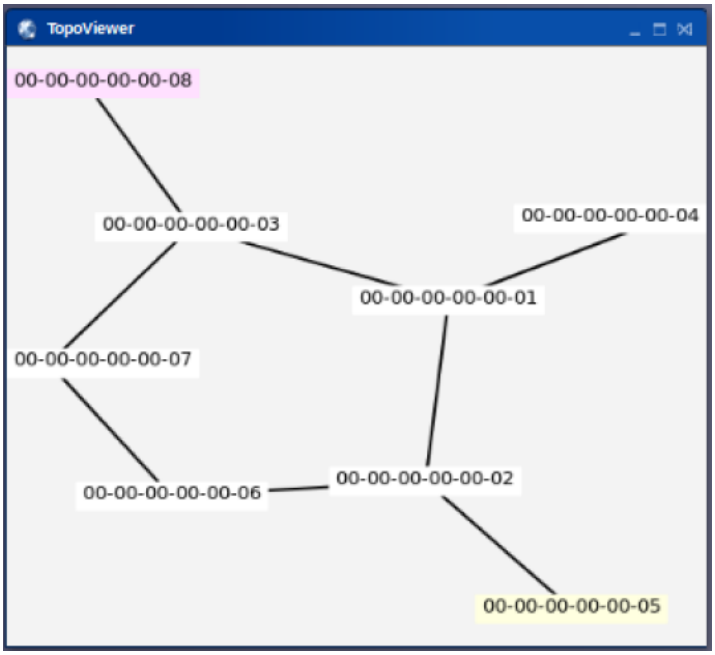


Imagen 5.4.4: topología mostrada en POXDesk.

```
mininet> net
h1 h1-eth0:s1-eth4
h2 h2-eth0:s2-eth4
h3 h3-eth0:s2-eth5
h4 h4-eth0:s5-eth2
h5 h5-eth0:s5-eth3
h6 h6-eth0:s6-eth3
h7 h7-eth0:s6-eth4
h8 h8-eth0:s3-eth4
h9 h9-eth0:s7-eth3
h10 h10-eth0:s8-eth2
h11 h11-eth0:s8-eth3
h12 h12-eth0:s8-eth4
h13 h13-eth0:s4-eth2
h14 h14-eth0:s4-eth3
h15 h15-eth0:s4-eth4
s1 lo:  s1-eth1:s2-eth1 s1-eth2:s3-eth1 s1-eth3:s4-eth1 s1-eth4:h1-eth0
s2 lo:  s2-eth1:s1-eth1 s2-eth2:s5-eth1 s2-eth3:s6-eth1 s2-eth4:h2-eth0 s2-eth5:h3-eth0
s3 lo:  s3-eth1:s1-eth2 s3-eth2:s7-eth1 s3-eth3:s8-eth1 s3-eth4:h8-eth0
s4 lo:  s4-eth1:s1-eth3 s4-eth2:h13-eth0 s4-eth3:h14-eth0 s4-eth4:h15-eth0
s5 lo:  s5-eth1:s2-eth2 s5-eth2:h4-eth0 s5-eth3:h5-eth0
s6 lo:  s6-eth1:s2-eth3 s6-eth2:s7-eth2 s6-eth3:h6-eth0 s6-eth4:h7-eth0
s7 lo:  s7-eth1:s3-eth2 s7-eth2:s6-eth2 s7-eth3:h9-eth0
s8 lo:  s8-eth1:s3-eth3 s8-eth2:h10-eth0 s8-eth3:h11-eth0 s8-eth4:h12-eth0
c0
```

Imagen 5.4.5: salida al comando net en Mininet.

donde podemos ver que existen 15 hosts, de h1 a h15, conectados a ocho switches como se muestra en la tabla

host	h1	h2	h3	h4	h5	h6	h7	h8	h9	h10	h11
switch	s1	s2	s2	s5	s5	s6	s6	s3	s7	s8	s8
puerto host	eth0	eth0	eth0	eth0	eth0	eth0	eth0	eth0	eth0	eth0	eth0
puerto switch	eth4	eth4	eth5	eth2	eth3	eth3	eth4	eth4	eth3	eth2	eth3

h12	h13	h14	h15
s8	s4	s4	s4
eth0	eth0	eth0	eth0
eth4	eth2	eth3	eth4

Imagen 5.4.6: tabla de conexiones entres hosts y switches.

y un controlador c0.

Al realizar un ping desde h3 a h9 se ve que el tiempo de respuesta del primer paquete es muy superior al resto, 116ms frente a 22ms.

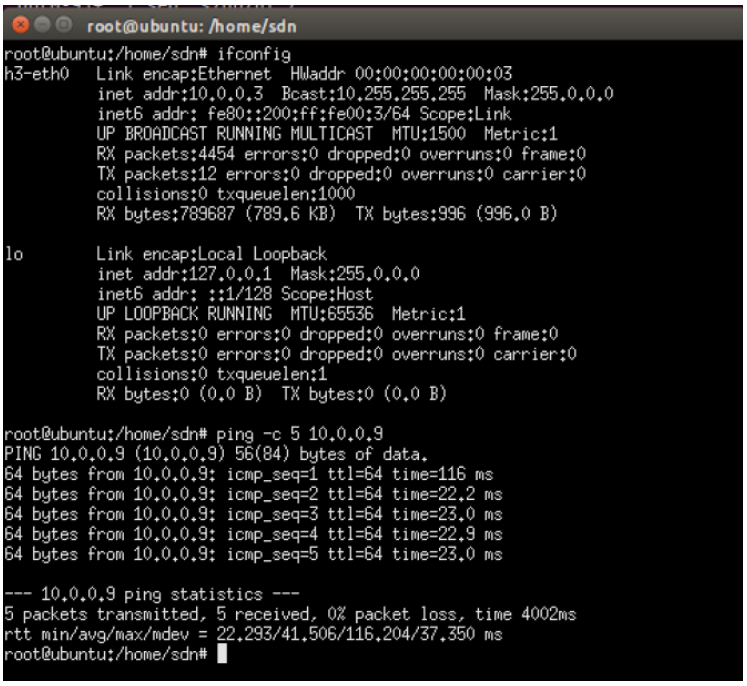


Imagen 5.4.7: ping de h3 a h9.

Capturando el tráfico podemos ver cómo se producen mensajes ARP para saber qué dirección física corresponde a la IP 9.0.0.9 y los switches mandan mensajes al controlador de consulta de la mejor ruta disponible.

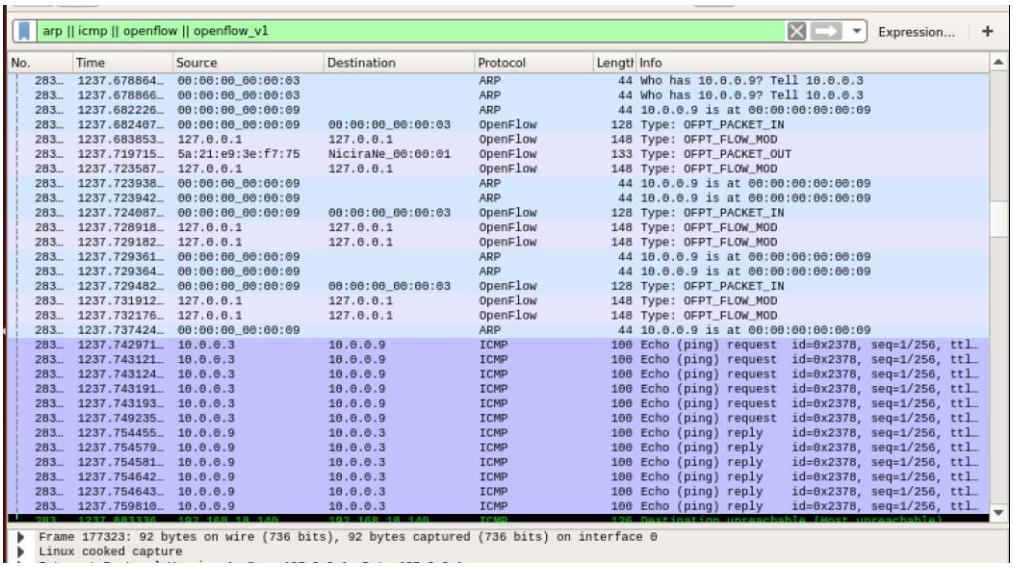


Imagen 5.4.8: mensajes capturados primer ping.

Al realizar el ping a continuación de forma inversa, esto no ocurre, sino que se mantiene estable entorno a lo 22ms.

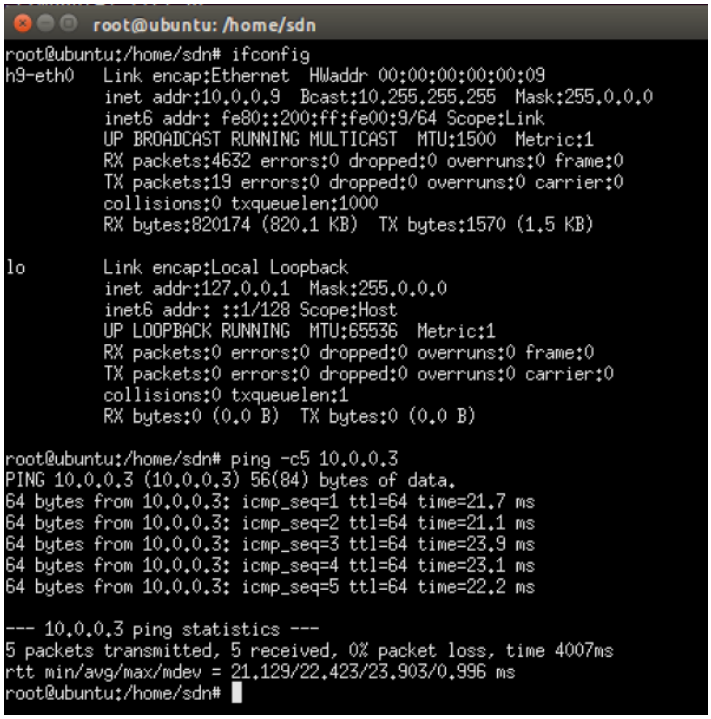


Imagen 5.4.9: ping de h9 a h3.

Y vemos que ya no son necesarios los mensajes ARP y las consultas al controlador, iniciándose la comunicación de forma directa, únicamente con los cinco ICMP ping request de h9 a h3 y sus cinco ping reply.

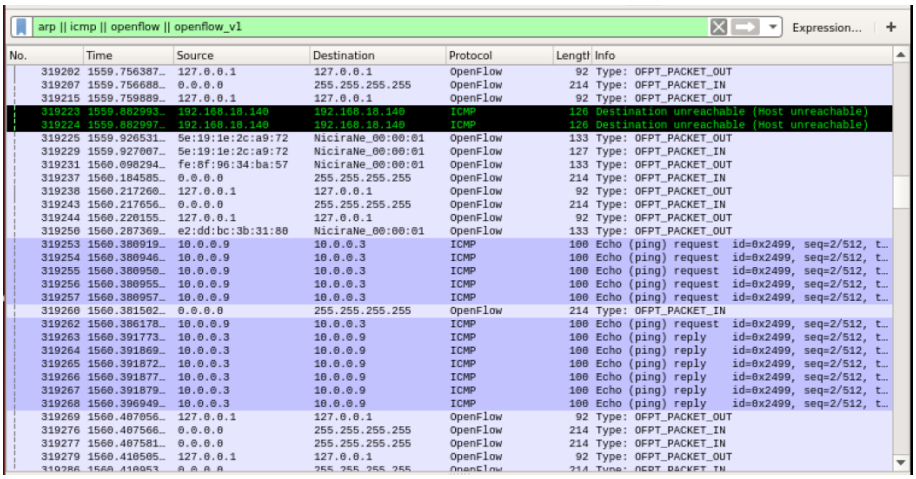


Imagen 5.4.10: mensajes capturados segundo ping.

Esto se debe a que los equipos intermedios cuentan en sus tablas de flujo con la información correspondiente a dicha ruta de comunicación y no necesitan realizar la consulta al controlador.

En la tabla de flujo de s2 podemos comprobar cómo los paquetes con MAC de destino 00:00:00:00:00:09 provenientes de 00:00:00:00:00:03 los reenvía por el puerto eth3 (s6) y cuando el sentido es el contrario los saca por eth5 (h3).

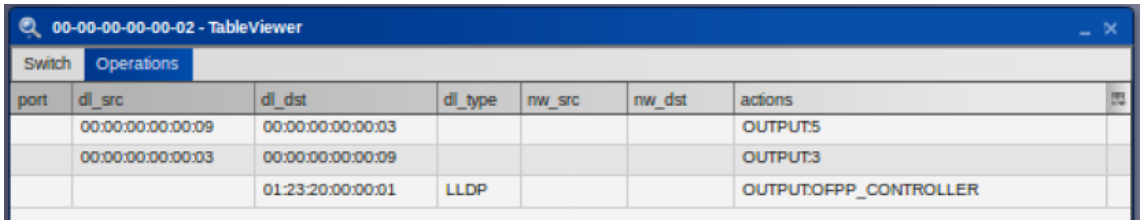


Imagen 5.4.11: tabla de flujo de s2 tras los mensajes ICMP de ping.

En cuanto al ancho de banda entre los dispositivos lo podemos con iperf desde Mininet haciendo uso de

```
iperf h2 h11
```

lo que inicia un servidor iperf en h11 como servidor, y lo lanza desde h2 como cliente contra h11.

Imagen 5.4.12: test de ancho de banda entre h2 y h11.

Podemos comprobar que es de 400Mbps, aceptable si tenemos en cuenta la limitación existente de 500mbps en los enlaces entre los hosts y los switches.

6. Conclusiones

Debido a la supremacía de fabricantes como CISCO, Ubiquiti o HP,con Aruba, en el ámbito de las redes y a la imposibilidad, tanto técnica como logística, de contar con todas las nuevas tecnologías que surgen en el mercado, se necesitaba de un entorno software de fácil instalación y uso para posibilitar la realización de prácticas en este entorno. Para ello se necesitaba realizar un estudio de los componentes existentes en las redes que hacen uso de este paradigma y que pueden diferir del paradigma clásico. Una vez que contamos con una visión general de la situación en torno a SDN estábamos listos para elegir los elementos con los que queremos desplegar nuestro laboratorio virtual.

Los entornos de virtualización más conocidos, como GNS3 o EVE-NG, cuentan con una curva de aprendizaje mucho más rápida que Mininet y su entorno gráfico facilita la creación de las topologías. Por contra, cuentan con dos grandes limitaciones como son la dificultad existente para encontrar y añadir las imágenes de los equipos que se quieren simular y su peor rendimiento limitan en gran medida la creación de topologías complejas. En el lado contrario se encuentra Mininet que, a pesar de no contar con un entorno gráfico por defecto, cuenta con todo lo necesario para desplegar una red desde el mismo momento de su instalación, lo que lo hace perfecto para iniciarse en el paradigma SDN.

Por otro lado, debido a la reducida oferta de controladores open source, y la dificultad de instalación de OpenDaylight, controlador open source más usado en la actualidad, nos induce a buscar alternativas menos atractivas desde el punto de vista gráfico. El controlador elegido, POX, a pesar de no contar con una interfaz gráfica por defecto, es fácilmente instalable y configurable. Además, existe la extensión POXDesk que provee al controlador de una interfaz gráfica que facilita la interacción en gran medida.

Para poder exprimir toda la información que nos ofrece la simulación es necesario hacer uso de un sniffer, nosotros hemos elegido Wireshark por su estudio durante el grado. Mediante el análisis de los paquetes generados mediante herramientas de testeo, como iperf o ping, y los mensajes OpenFlow generados durante la comunicación ha sido posible entender de forma bastante completa el funcionamiento real de la red y su administración.

Tras la finalización del proyecto, a pesar de las dificultades para encontrar información sobre algunos elementos como las northbound APIs o algunos controladores, podemos concluir que se ha conseguido cumplir con los objetivos de forma exitosa.

7. Líneas futuras

Las líneas futuras del proyecto podrían ser muchas, ya que OpenFlow permite una gran flexibilidad en las acciones a realizar. Pero las más interesantes, que podrían aportar un mayor grado de conocimiento y más útiles para entornos profesionales serían las siguientes:

- Creación de topologías más complejas en las que se cuenten con otros tipos de elementos en la red, como servidores HTTP, y la implementación de balanceadores de carga. Dicha mejora es de gran utilidad en entornos empresariales y para administración de enlaces que tengan que lidiar con una gran cantidad de tráfico de forma habitual.

- Desarrollo de un Firewall. Esto supone un aumento de seguridad en la red y la posibilidad de crear zonas especiales a las que sólo ciertos equipos de la red pudieran tener acceso.

- Migración a un controlador más avanzado con una interfaz gráfica más avanzada y que permita una mayor interacción con la red de manera intuitiva. Aunque POX se trata de un controlador sencillo de aprender y de implementar, las utilidades que ofrecen están limitadas y no cuenta con la gran comunidad que tienen detrás otros controladores como Floodlight o OpenDaylight.

8. Bibliografía

1] R. M.-B. Asensio, «Biografía de Joseph Licklider, uno de los padres de internet,» 27 09 2016. [En línea]. Available: https://www.um.es/docencia/barzana/BIOGRAFIAS/Biografia-JCR-Licklidder.php. [Último acceso: 27 05 2020].

2] J. Penalva, «Todo Internet cabía en un papel A4 hace 40 años: la historia del nacimiento de ARPANET,» 17 5 2019. [En línea]. Available: https://www.xataka.com/historia-tecnologica/todo-internet-cabia-en-un-papel-a4-hace-40-anos-la-historia-del-nacimiento-de-arpanet. [Último acceso: 2 11 2020].

3] «ARPANET United States defense program,» [En línea]. Available: https://www.britannica.com/topic/ARPANET . [Último acceso: 3 6 2020].

4] ms.gonzalez, «Historia de Internet – nacimiento y evolución,» 17 9 2013. [En línea]. Available: http://redestelematicas.com/historia-de-internet-nacimiento-y-evolucion/. [Último acceso: 28 5 2020].

5] I. D. M. Group, «IT Digital Media Group,» 4 5 2020. [En línea]. Available: https://www.ituser.es/actualidad/2020/05/la-tasa-de-trafico-de-datos-supera-los-91-terabits-decix-analiza-la-evolucion-de-internet. [Último acceso: 4 6 2020].

6] J. Pérez Martinez, Z. Frías Barroso y A. Urueña López, «La evolución de Internet en España: de Tesys a la econimía digital.,» 5 2018. [En línea]. Available: https://www.ontsi.red.es/sites/ontsi/files/50%20A%c3%b1os%20de%20la%20Red%20de%20Redes.pdf. [Último acceso: 10 11 2020].

7] D. De la Torre, «Los datos de Internet en España en la última década confirman una revolución digital,» Movistar Telefónica S.A., 17 5 2018. [En línea]. Available: https://blogthinkbig.com/internet-datos-espana. [Último acceso: 10 11 2020].

8] B. A. A. Nunes, M. Mendoca, X.-N. Nguyen, K. Obraczka y T. Turletti, «A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks,» *IEEE Communications Surveys & Tutorials*, vol. XVI, nº 3, pp. 1617-1634, 2014.

9] I. K. K. M. y. J. V. A.T. Campbell, «Open signaling for atm, internet and mobile networks (opensig'98),» *ACM SIGCOMM Computer Communication Review*, vol. 29, nº 1, pp. 97-108, 1999.

10] J. M. S. W. D. S. D. J. W. y. G. J. M. D. L. Tennenhouse, «A survey of active network research,» *IEEE Communications Magazine*, vol. 35, nº 1, pp. 80-86, 1997.

11] I. Leslie, S. Crosby y S. Rooney, «Devolved Control of ATM Networks,» 07 02 2000. [En línea]. Available: https://www.cl.cam.ac.uk/research/srg/netos/projects/archive/dcan/. [Último acceso: 21 07 2020].

12] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan y H. Zhang, «A clean slate 4d approach to network control and

management,» *ACM SIGCOMM Computer Communication Review,* vol. 35, nº 5, pp. 41-54, 2005.

[13] R. Enns, «NETCONF Configuration Protocol; RFC 4741,» 2006. [En línea]. Available: https://tools.ietf.org/html/rfc4741. [Último acceso: 23 07 2020].

[14] J. D. Case, M. Fedor, M. L. Schoffstall y J. Davin, «Simple network management protocol (SNMP),» 1990. [En línea]. Available: https://tools.ietf.org/html/rfc1157. [Último acceso: 23 7 2020].

[15] M. Casado, M. Freedman, J. Pettit, J. Luo, N. McKeown y S. Shenker, «Ethane: Taking control of the enterprise,» *ACM SIGCOMM Computer Communication Review,* vol. 37, nº 4, pp. 1-12, 2007.

[16] «Open Network Fundation,» [En línea]. Available: https://www.opennetworking.org/mission/. [Último acceso: 1 8 2020].

[17] «SDN definition,» [En línea]. Available: https://www.opennetworking.org/sdn-definition/. [Último acceso: 1 8 2020].

[18] D. Joachimpillai, J. Hadi Salim, A. Farrel y J. Halpern, «Forwarding and Control Element Separation (forces),» IETF, [En línea]. Available: https://datatracker.ietf.org/wg/forces/about/. [Último acceso: 11 8 2020].

[19] «OpenFlow,» Open Networking Foundation, [En línea]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf. [Último acceso: 3 8 2020].

[20] J. Videl Nonell y B. Marin Fumanal, «REDES TRADICIONALES VS SDN DEFINIDAS POR SOFTWARE,» 11 3 2020. [En línea]. Available: https://blogs.salleurl.edu/es/redes-tradicionales-vs-sdn-definidas-por-software. [Último acceso: 2 11 2020].

[21] «RFC Index,» IETF, [En línea]. Available: https://tools.ietf.org/rfc/index. [Último acceso: 3 8 2020].

[22] «OpenFlow Switch Specification v1.5.1,» 26 3 2015. [En línea]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf. [Último acceso: 21 8 2020].

[23] B. Stephens, A. Cox, W. Felter, C. Dixon y J. Carter, «Past: scalable ethernet for data centers,» *8th international conference on Emerging networking experiments and technologies, CoNEXT '12,* pp. 49-60, 2012.

[24] «OpenFlow Switch Specification v1.0.0,» 31 12 2009. [En línea]. Available: https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf. [Último acceso: 21 8 2020].

[25] R. Sridhar, «SDN Series Part Three: NOX, the Original OpenFlow Controller,» 15 12 2014. [En línea]. Available: https://thenewstack.io/sdn-series-part-iii-nox-the-original-openflow-controller/. [Último acceso: 5 11 2020].

[26] D. Erickson, Standford University, 4 2 2013. [En línea]. Available: https://openflow.stanford.edu/display/Beacon/Home.html. [Último acceso: 5 11 2020].

[27] S. Rao, «DN Series Part Six: OpenDaylight, the Most Documented Controller,» 20 1 2015. [En línea]. Available: https://thenewstack.io/sdn-series-part-vi-opendaylight/ . [Último acceso: 5 11 2020].

The Linux Fundation, [En línea]. Available: https://www.linuxfoundation.org/projects/networking/ . [Último acceso: 5 11 2020].
[28]

NEC Corporation of America, «Enabling Modern IT Environments,» NEC Corporation of America, [En línea]. Available: https://www.necam.com/sdn/ . [Último acceso: 5 11 2020].
[29]

IBM, «Software Defined Networking (SDN) Services,» IBM, [En línea]. Available: https://www.ibm.com/services/network/software-defined . [Último acceso: 5 11 2020].
[30]

Hewlett Packard Enterprise Development LP, «HPE VAN SDN Controller Software - Overview,» Hewlett & Packard, [En línea]. Available: https://support.hpe.com/hpesc/public/docDisplay?docId=emr_na-c03967699#N10012 . [Último acceso: 5 11 2020].
[31]

W. Zhou, L. Li, M. Luo y W. Chou, «REST API Design Patterns for SDN Northbound API,» *28th International Conference on Advanced Information Networking and Applications Workshops*, 2014.
[32]

Standord University, «Ethane: A Security Management Architecture,» [En línea]. Available: http://yuba.stanford.edu/ethane. [Último acceso: 15 10 2020].
[33]

A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma y S. Banerjee, «Devoflow: scaling flow management for high-performance networks,» *SIGCOMM Comput. Commun,* vol. 4, nº 41, pp. 254-265, 2011.
[34]

L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann y T. Vazao, «Towards programmable enterprise wlans with odin,» *Proceedings of the first workshop on Hot topics in software defined networks, HotSDN '12,* pp. 115-120, 2012.
[35]

M. Bansal, J. Mehlman, S. Katti y P. Levis, «Openradio: a programmable wireless dataplane,» *Proceedings of the first workshop on Hot topics in software defined networks,* pp. 109-114, 2012.
[36]

L. Galaxy Technologies, «GNS3,» [En línea]. Available: https://www.gns3.com/. [Último acceso: 25 10 2020].
[37]

EVE-NG, «Emulated Virtual Enviroment Next Gerenation,» EVE-NG Ltd, [En línea]. Available: https://www.eve-ng.net/. [Último acceso: 25 10 2020].
[38]

Mininet, «Mininet An Instant Virtual Network on your Laptop (or other PC),» Mininet Team, [En línea]. Available: http://mininet.org/.
[39]

VMWare, Inc., «VMWARE,» [En línea]. Available: https://www.vmware.com/es/products/workstation-player/workstation-player-evaluation.html. [Último acceso: 25 10 2020].
[40]

Ubuntu, «Ubuntu 14.04.06 LTS,» [En línea]. Available: https://releases.ubuntu.com/14.04/. [Último acceso: 25 10 2020].
[41]

B. Linkletter, «http://www.brianlinkletter.com,» [En línea]. Available: http://www.brianlinkletter.com/how-to-install-mininet-sdn-network-simulator. [Último acceso: 25 10 2020].
[42]

«noxrepo,» [En línea]. Available: https://noxrepo.github.io/pox-doc/html/#installing-pox/. [Último acceso: 25 10 2020].
[43]

«Donghowa,» 12 10 2012. [En línea]. Available: http://donghowa.net/ubuntu-sdnopenflow-installation-guide-2-pox-poxdesk/. [Último acceso: 25 10 2020].
[44]

DONGHOWA, «http://donghowa.net/,» [En línea]. Available:
45] http://donghowa.net/ubuntu-sdnopenflow-installation-guide-2-pox-poxdesk/. [Último acceso: 28 10 2020].

B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C.
46] Lynch, J. Postel, L. G. Roberts y S. Wolff, «Breve historia de internet,» Internet Society, [En línea]. Available: https://www.internetsociety.org/es/internet/history-internet/brief-history-internet/. [Último acceso: 27 5 2020].

History Computer, «Histpory Computer,» [En línea]. Available: history-
47] computer.com/Internet/Maturing/TCPIP.html. [Último acceso: 4 6 2020].

VMWare Inc, «VMWARE,» [En línea]. Available:
48] https://www.vmware.com/es/products/workstation-player/workstation-player-evaluation.html. [Último acceso: 25 10 2020].

Ubuntu, «Ubuntu 14.04.06 LTS,» [En línea]. Available:
49] https://releases.ubuntu.com/14.04/. [Último acceso: 25 10 2020].

Mininet, «Mininet documentation,» [En línea]. Available:
50] https://github.com/mininet/mininet/wiki/Documentation. [Último acceso: 28 10 2020].

Mininet, «Mininet Python API Reference Manual,» [En línea]. Available:
51] http://mininet.org/api/hierarchy.html. [Último acceso: 28 10 2020].

J. L. Henao Ramirez, «Guía de implentación y uso del emulador mininet,»
52] 2015. [En línea]. Available: http://repositorio.utp.edu.co/dspace/bitstream/handle/11059/5713/00678H493%20_Anexo.pdf;jsessionid=84231F37BA888D5F3C8ED69B5BACFF3F?sequence=2. [Último acceso: 28 10 2020].

Standfor University, «POX Wiki,» [En línea]. Available:
53] https://openflow.stanford.edu/display/ONL/POX+Wiki. [Último acceso: 28 10 2020].

P. V. Tijare y D. Vasudevan, «THE NORTHBOUND APIs OF
54] SOFTWARE DEFINED NETWORKS,» *INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY,* vol. 5, nº 10, pp. 501-513, 2016.

9. Anexos

9.1 ANEXO I. Instalación del software necesario

9.1.1. Instalación VMware

Para proceder a la instalación accedemos a [40] y elegimos la versión para Windows.

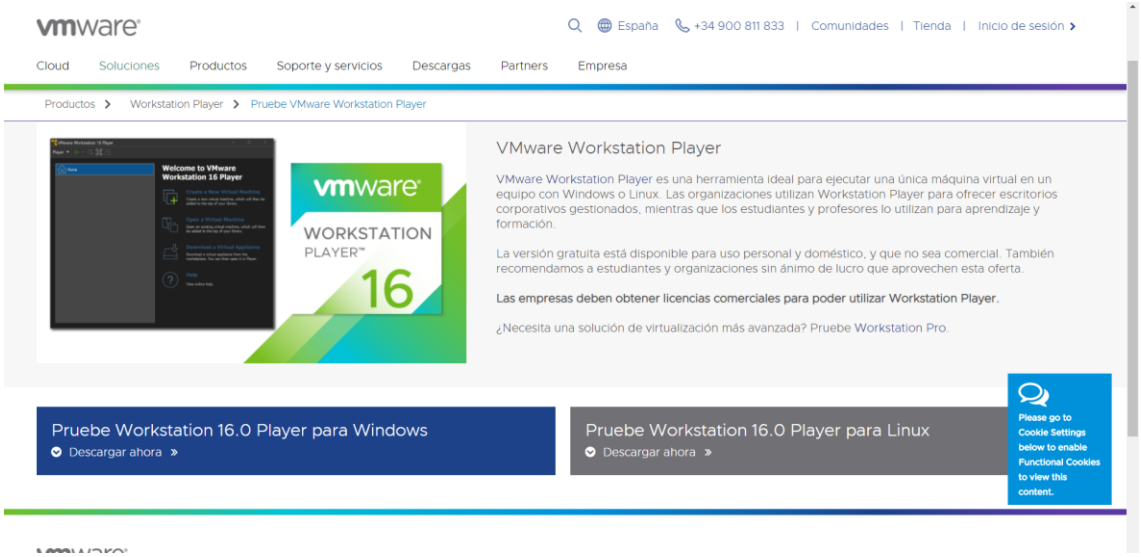


Imagen 9.1.1: Página de descarga de VMWare Workstation

9.1.2. Descarga de Ubuntu 14.04 y creación de máquina virtual

Accedemos al siguiente enlace [41] y descargamos la versión que deseemos, en nuestro casi la de 64 bits AMD64.

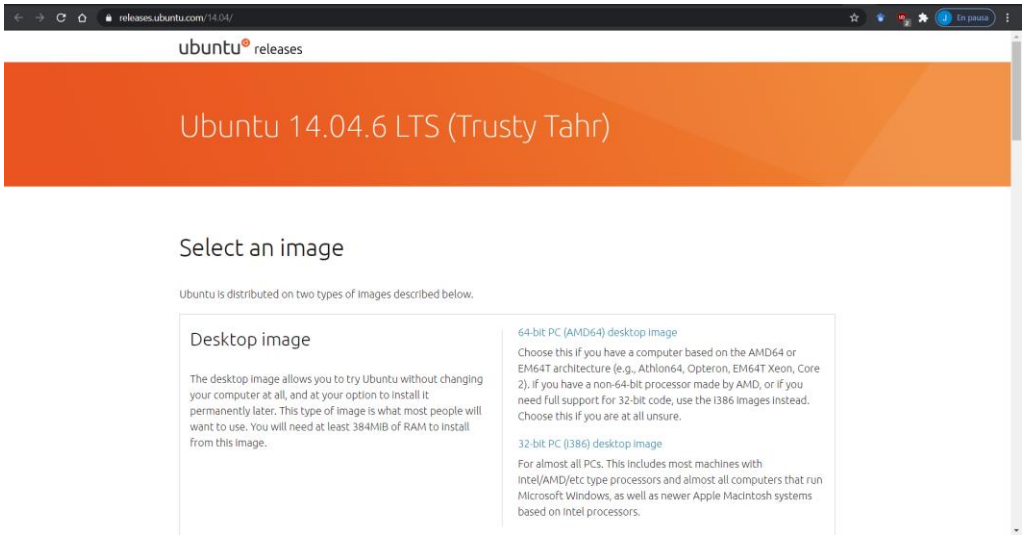


Imagen 9.1.2: página descarga Ubuntu 14.04.

Una vez descargada la imagen abrimos VMWare y creamos una máquina usando la iso de Ubuntu descargada.

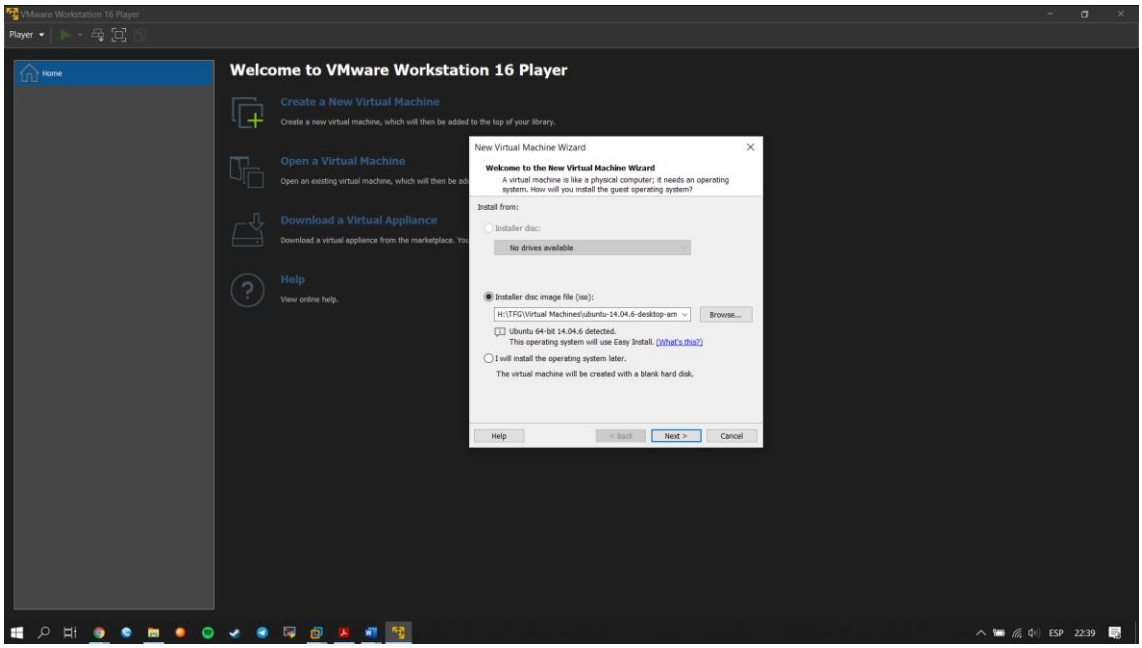


Imagen 9.1.3: Creación de máquina virtual con Ubuntu 14

Se establecen un usuario y contraseña para entrar a la máquina. En este caso escogemos como user/pass “sdn” en ambos.

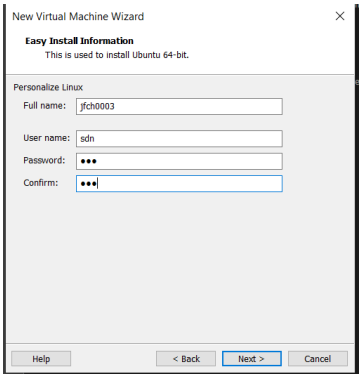


Imagen 9.1.4: Elección de user/pass para la máquina virtual

Se elige un nombre adecuado para la máquina virtual.

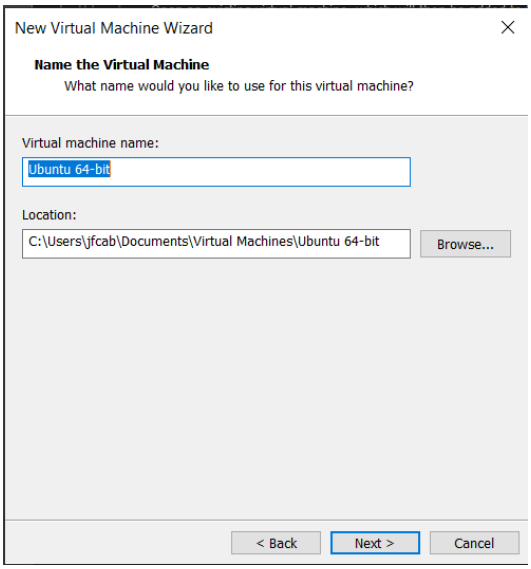


Imagen 9.1.5: elección de nombre para la máquina virtual.

Se le asignan recursos y se puede customizar el hardware a utilizar.

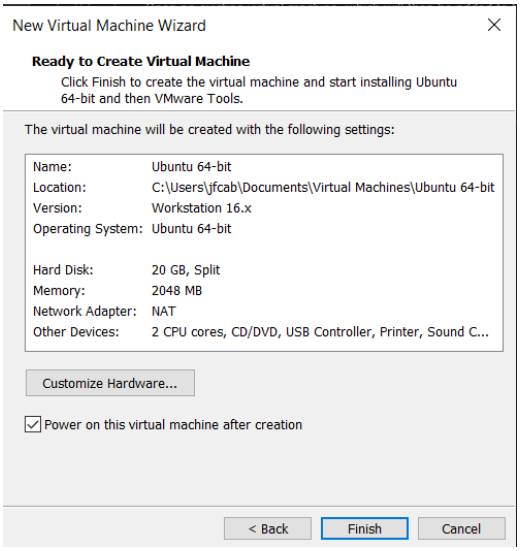


Imagen 9.1.6: selección del hardware utilizable por la máquina virtual

Una vez en este punto, se le da a finalizar y se lanza la máquina.

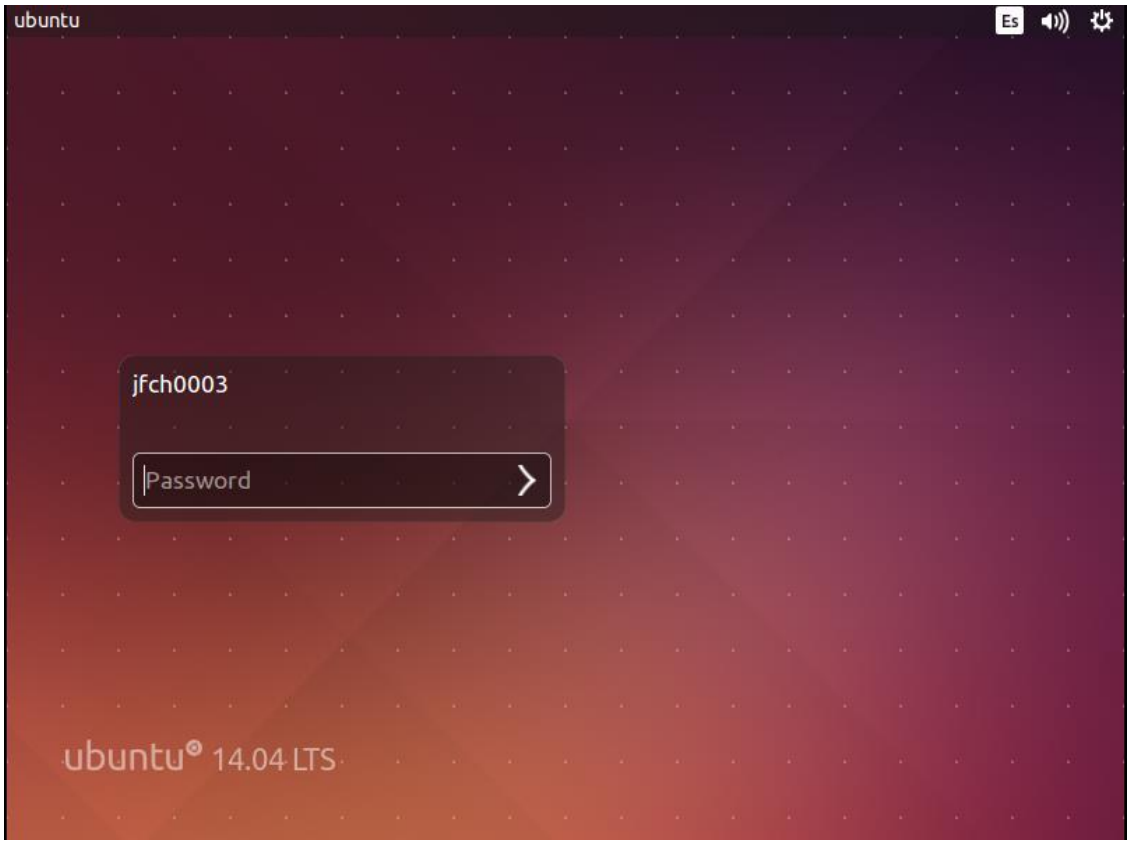


Imagen 9.1.7: Primer inicio de la máquina virtual

9.1.3. Instalación de Mininet y Wireshark

El proceso de instalación de Mininet es muy simple, ya que consiste en la clonación de un proyecto de GitHub, elegir la versión y ejecutar el asistente de instalación que proporciona. El proceso se describe en [42], pero lo vamos a realizar un poco diferente para ajustarse a la idiosincrasia del proyecto:

- Lo primero es obtener permisos de super usuario ejecutando el comando sudo su

```
$ sudo su
```

- Actualizamos los paquetes del SO antes de realizar los cambios para evitar posibles errores.

```
$ apt-get update
$ apt-get upgrade
$ apt-get dist-upgrade
```

- Añadimos el paquete git para la descarga del repositorio GitHub y clonamos el del proyecto Mininet, entramos en la carpeta creada y elegimos una versión

```
$ apt-get git install
$ git clone git://github.com/mininet/mininet
$ git checkout -b 2.2.2
Switched to a new branch '2.2.2'
```

- Antes de la instalación se puede comprobar las opciones de instalación con el siguiente comando

```
$ ~/mininet/util/install.sh -h
```

- Ya que vamos a instalar un controlador externo, elegimos la siguiente opción, la cual instala el propio Mininet, la southbound API OpenFlow y una versión de Open vSwitch.

```
$ ~/mininet/util/install.sh -nfv
```

- También se va a aprovechar e instalar Wireshark usando el mismo asistente de instalación mediante el siguiente comando

```
$ ~/mininet/util/install.sh -w
```

9.1.4. Instalación del controlador POX

A continuación, se va a proceder a la instalación del controlador POX, siguiendo el método explicado en el propio repositorio GitHub [43]:

- Clonamos el repositorio del proyecto mediante el siguiente comando:

```
$ git clone http://github.com/noxrepo/pox
$ cd pox
```

- Accedemos a la rama deseada y está listo para funcionar:

```
~/pox$ git checkout dart
```

9.1.5. Instalación de la extensión POXDesk

POX es un controlador que, por defecto, no incluye una interfaz gráfica para visualizar lo que ocurre en la red, por lo que vamos a recurrir a una extensión llamada POXDesk. El proceso se describe en [44]:

- Una vez instalado POX, se accede a la carpeta /ext y se clona el siguiente repositorio

```
$ cd ext
$ git clone https://github.com/MurphyMc/poxdesk
```

- Se accede a la carpeta creada /poxdesk y se descarga el siguiente archivo

```
$ wget http://downloads.sourceforge.net/qooxdoo/qooxdoo-2.0.1-sdk.zip
$ unzip qooxdoo-2.0.1-sdk.zip
$ mv qooxdoo-2.0.1-sdk qx
$ cd ../..
```

- Una vez realizado este proceso, se puede invocar el controlador desde la carpeta /pox con el siguiente comando

```
$ ./pox.py forwarding.l2_learning samples.pretty_log web messenger messenger.log_service messenger.ajax_transport openflow.of_service openflow.webservice poxdesk openflow.discovery poxdesk.tinytopo
```

Para el acceso a la interfaz gráfica se hace uso de un navegador mediante la siguiente dirección "`http://127.0.0.1:8000/poxdesk`".

9.2. ANEXO II. Pequeño manual de uso de Mininet y Miniedit.

9.2.1. Mininet

El inicio de Mininet se realiza mediante el comando

```
sdn@ubuntu$ sudo mn
```

al cual se le pueden añadir una serie de opciones.

Entre las opciones más usuales se encuentra la de desplegar una topología predefinida, pudiendo añadir ciertos parámetros para su personalización. Esto se realiza con el comando `--topo=`TOPOLOGÍA. Para desplegar una topología predefinida existen tres opciones:

- `--topo=single, <n>` (1 switch, n hosts).
- `--topo=linear, <x>, <y>` (x siwtches, y nodos).
- `--topo=tree, <d>, <f>` (profundidad d, fanout f).

El comando

```
sdn@ubuntu$ sudo mn
```

sería equivalente a utilizar

```
sdn@ubuntu$ sudo mn --topo=single,2
```

ya que la topología desplegada cuenta con un único switch y dos hosts conectados a ella.

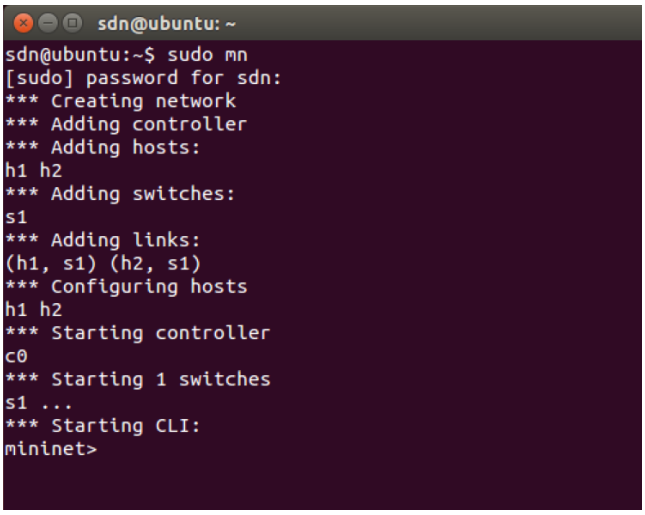


Imagen 9.2.1: Inicio de topología por defecto en Mininet

Una vez desplegada la red, para visualizar las conexiones entre los equipos se hace uso del siguiente comando:

```
mininet> net
```

Y que, en la topología por defecto daría una salida tal que así

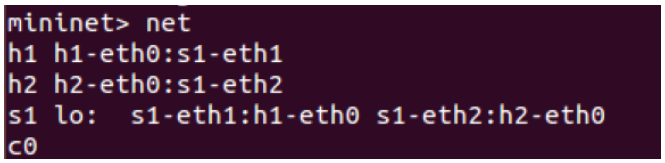


Imagen 9.2.2: Salida al comando net

El análisis de la salida sería el siguiente:

- Existen dos hosts, h1 y h2, un switch s1 y un controlador c0, que en este caso corre dentro del propio Mininet.
- h1 conecta desde su interfaz eth0 con la interfaz eth1 de s1
- h2 conecta desde su interfaz eth0 con la interfaz eth2 de s1
- s1 conecta, de forma local, con h1 y h2 desde las interfaces eth1 y th2 respectivamente con las interfaces eth0 de los hosts

Para comprobar la conectividad entre las partes existen varias herramientas, entre las que destacan:

```
pingall: realiza un test de conectividad entre los hosts.
```

- pingallfull: realiza un ping entre todos los hosts y te devuelve el tiempo máximo, medio y mínimo de la respuesta.

- También se pueden realizar las mismas acciones mediante el siguiente comando y funciona igual que un ping lanzado en Linux del terminal h1 al terminal h2, con las mismas opciones

```
mininet> h1 ping h2
```

- Otra opción es ejecutar `sdn@ubuntu$ sudo mn -x` para iniciar la simulación y se abrirá una ventana xTerm para cada equipo en la red, donde se pondrán realizar las pruebas de conectividad de forma directa en los equipos.

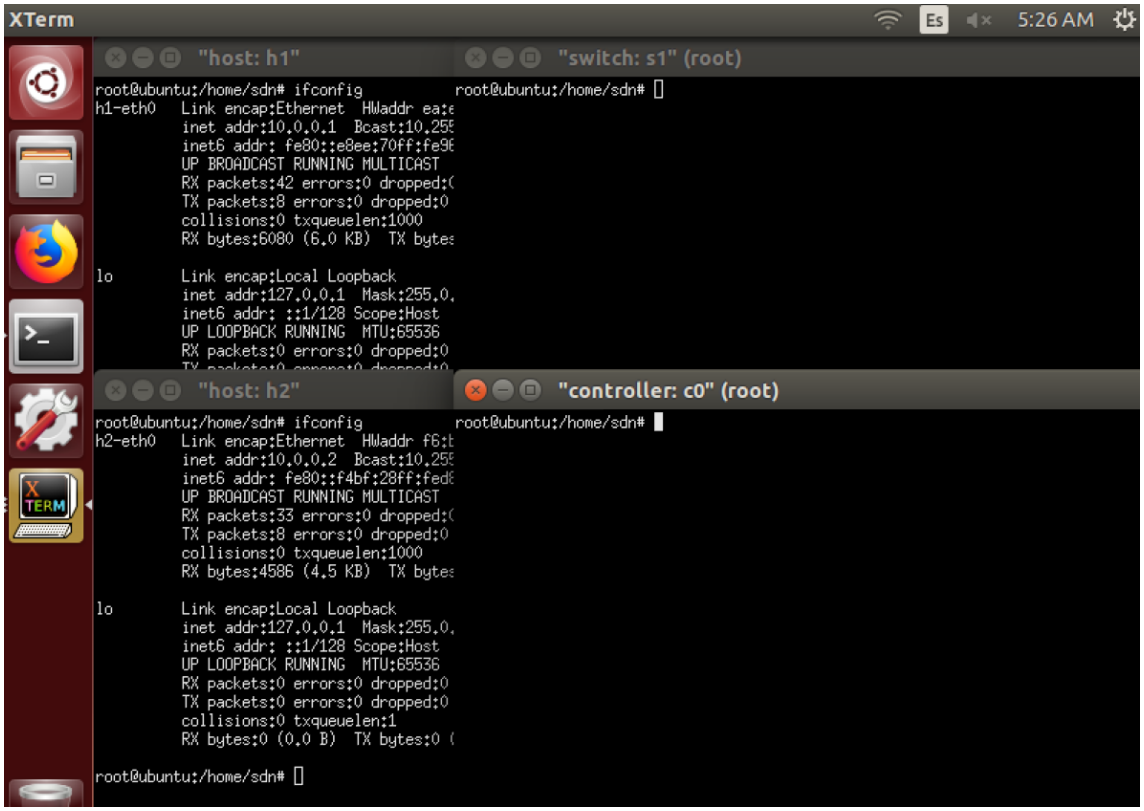


Imagen 9.2.3: Ventanas xTerm de los dispositivos de la red.

Esto puede llegar a no ser operativo en redes con gran cantidad de equipos, pudiendo generarse las terminales necesarias de forma puntual con el comando

```
HOST xterm
```

Otra herramienta que Mininet pone al servicio del usuario se trata de iperf, donde se permite conocer el ancho de banda disponible entre dos puntos.

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['50.3 Gbits/sec', '50.3 Gbits/sec']
```

Imagen 9.2.4: iperf realizado con h1 como cliente y h2 como servidor.

Esto sería equivalente a abrir un terminal de h2 y ejecutar

```
iperf -s
```

para iniciar un servidor de descarga en el puerto 5001, y en el lado de h1 ejecutar

```
iperf -c IP_H2
```

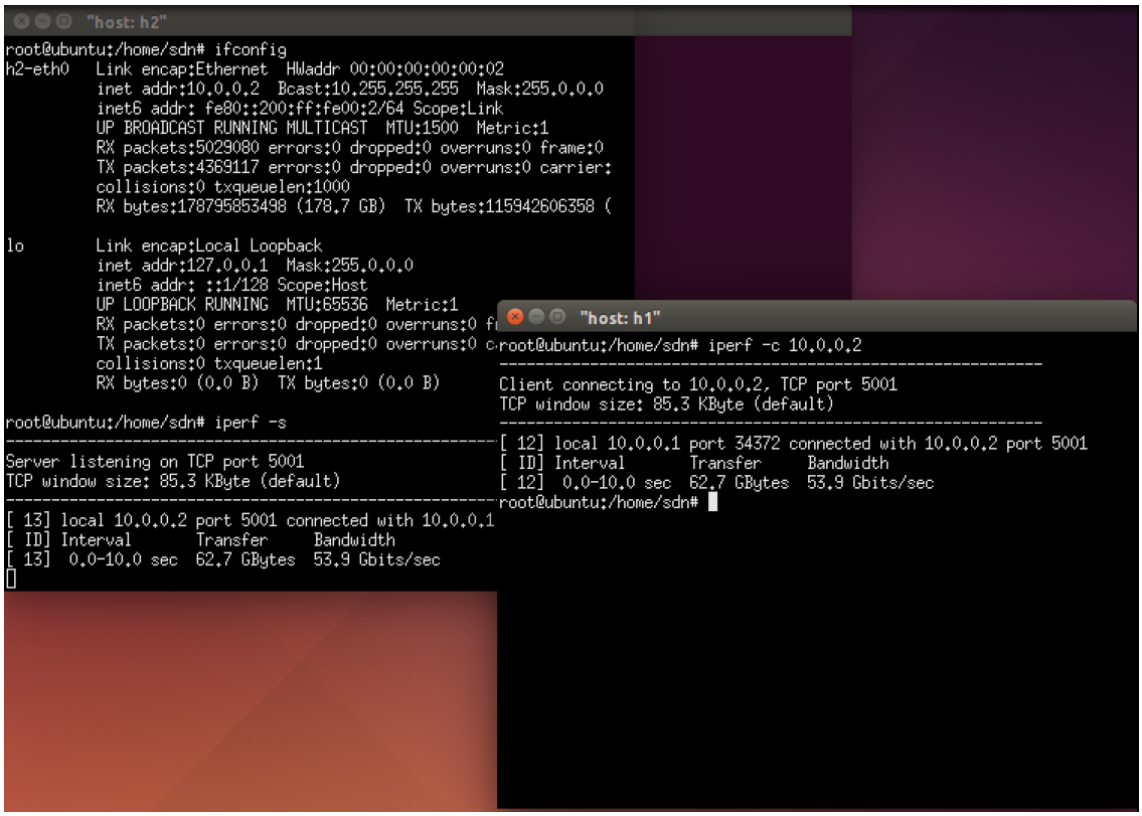


Imagen 9.2.5: iperf entre h1 y h2 mediante terminales.

9.2.2. Miniedit

Para el inicio de Miniedit se consigue acceso de superusuario, entramos en la carpeta mininet y ejecutamos el siguiente comando

```
root@ubuntu:/home/sdn/mininet# ./examples/miniedit.py
```

que como resultado abre la siguiente ventana

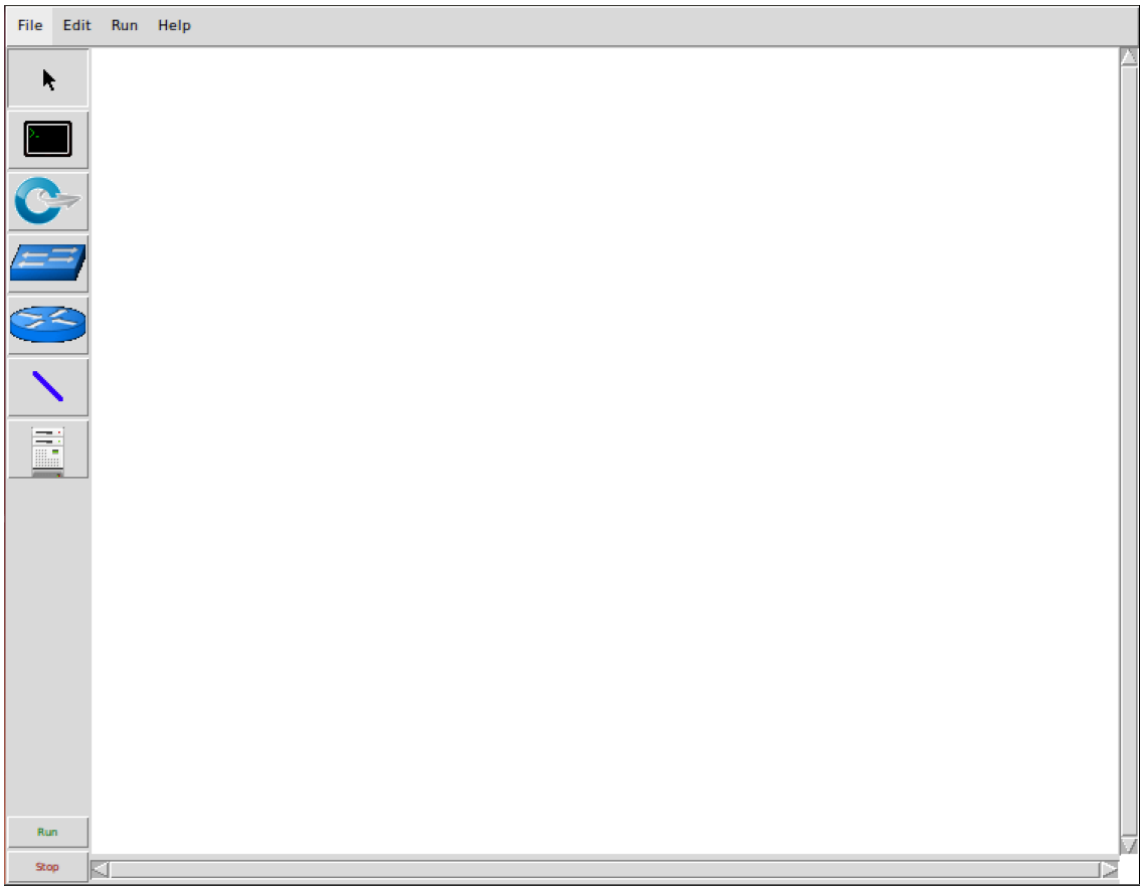


Imagen 9.2.6: Ventana de Mininet para la creación y edición de topologías.

Al pinchar en los elementos de la columna de la izquierda se selecciona de arriba a abajo:

- Select: selecciona y permite recolocar los equipos en la red.
- Host: añade tantos hosts como veces se pinche en la ventana destinada para tal fin.
- Switch: añade tantos switches como veces se pinche en la ventana destinada para tal fin.
- Legacy Switch: añade tantos legacy switches como veces se pinche en la ventana destinada para tal fin.
- Legacy Router: añade tantos legacy routes como veces se pinche en la ventana destinada para tal fin.
- Netlink: se selecciona equipo de inicio mediante um click que se mantiene hasta el equipo al que se quiere conectar. Esta operación se puede realizar tantas veces como se desee.

- Controller: añade tantos controladores como veces se pinche en la ventana destinada para tal fin.

Para acceder a las propiedades se hace click derecho sobre los equipos, se desliza hacia “properties” y se abre de forma automática la ventana de propiedades.

A fin de poder trabajar sobre la topología se puede exportar un .py que podemos ejecutar de dos maneras:

- `python ./RUTA_ARCHIVO/NOMBRE_ARCHIVO.py`
- `mn –custom ./RUTA_ARCHIVO/NOMBRE_ARCHIVO.py`

9.3. ANEXO III. Pequeño manual de POX y POXDesk

9.3.1. POX

Para el lanzamiento del controlador no es suficiente con entrar a la carpeta en la que se cuenta alojado y lanzarlo, sino que resulta necesario añadir los módulos que se quieren utilizar.

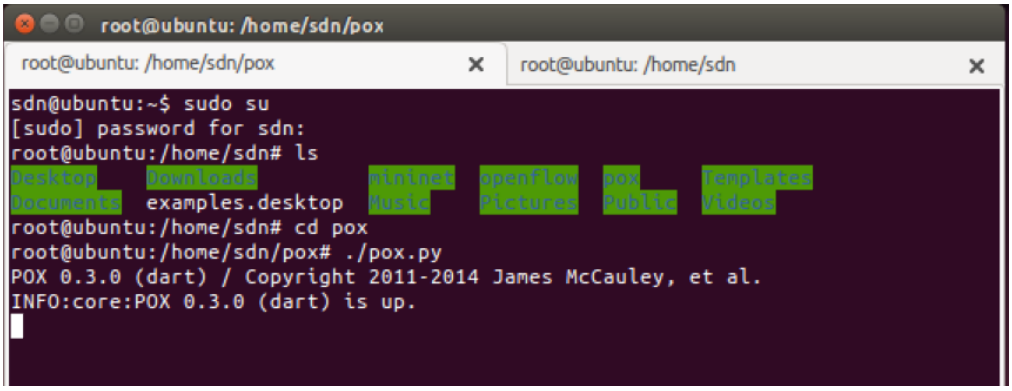


Imagen 9.3.1: Lanzamiento POX sin módulos.

Lanzamos un pingall en Mininet para comprobar su correcto funcionamiento.

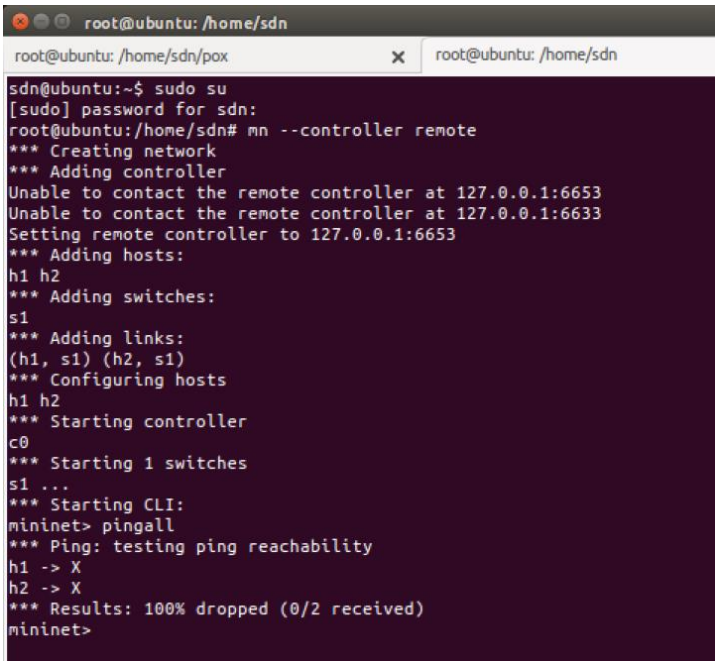


Imagen 9.3.2: pingall realizado en topología por defecto con POX sin módulos.

Como se puede observar, el controlador se encuentra en funcionamiento, Mininet no consigue establecer comunicación con él y, por tanto, al realizar un test de conectividad, se realiza la consulta al controlador, se vence el timeout y los paquetes de ping son descartados. El host con mac 00:00:00:00:00:01 (h1) lanza una petición ARP para conocer quién tiene la IP 9.0.0.2, correspondiente a h2.

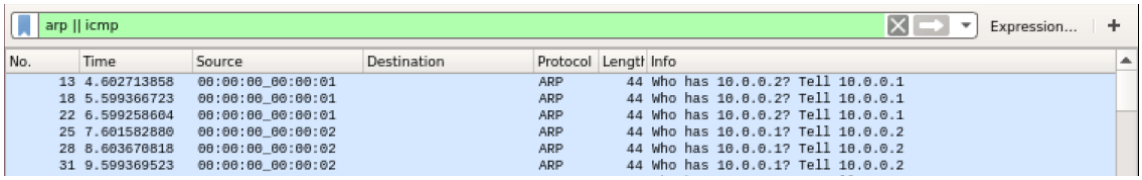


Imagen 9.3.3: Mensajes de ARP generados.

Este mismo proceso llevado a cabo desde fuera de la máquina virtual entre equipos de mi red interna da el siguiente resultado:

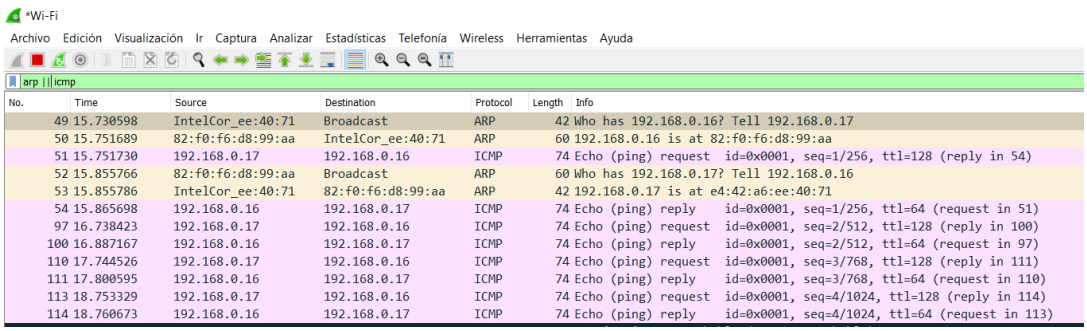


Imagen 9.3.4: ping entre pc y smartphone en red privada

Se puede ver que se realiza utiliza ARP para realizar un broadcast y preguntar qué dirección física cuenta con una dirección IP para poder almacenar en la tabla ARP y establecer una comunicación.

Para la activación del learning switch para que se pueda aprender las direcciones de los hosts presentes en la red se requiere de la inclusión del módulo forwarding.l2_pairs. De este modo se puede observar que el switch establece conexión con el controlador, y a la hora de hacer el ping se realiza el ARP, se consulta al controlador la acción a realizar, se devuelve la acción al switch y ya se puede comunicar las dos partes.

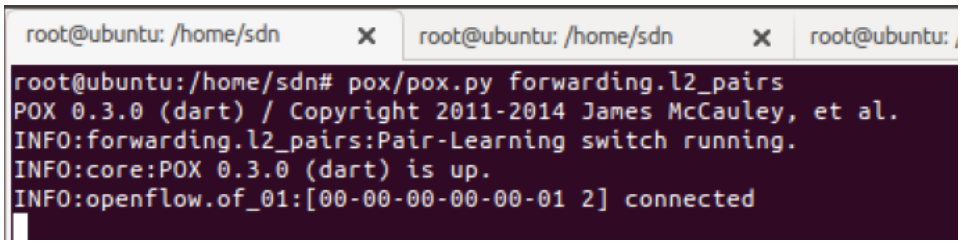


Imagen 9.3.4: activación del módulo learning switch en POX.

Al realizar un ping con el módulo activado ya sí hay conectividad entre los equipos en la red.

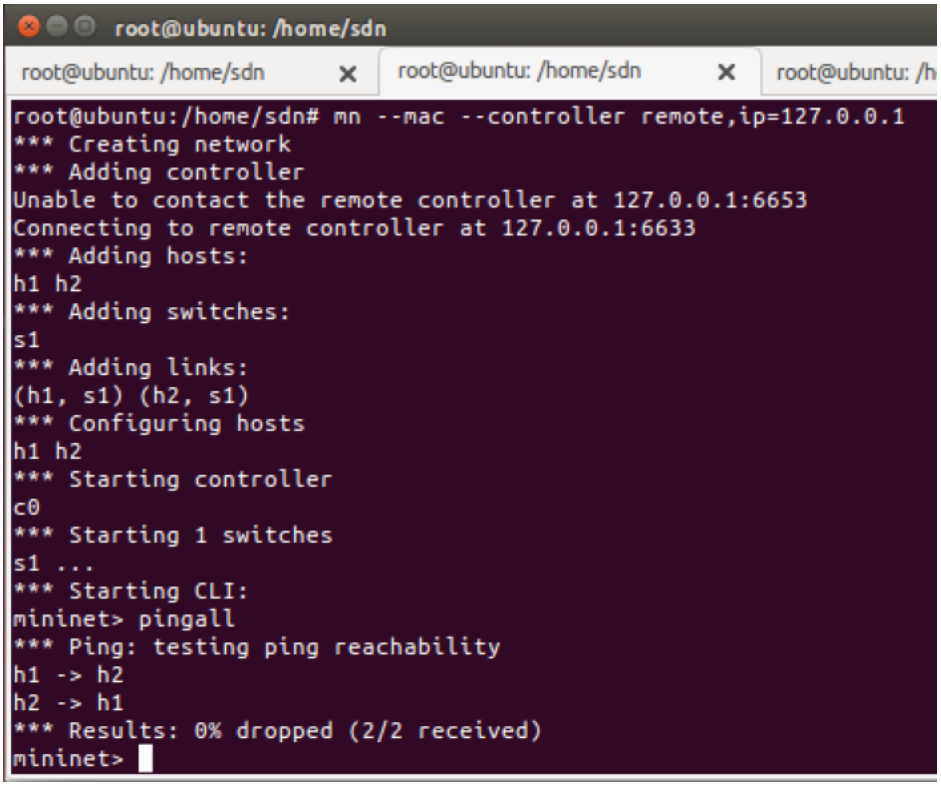


Imagen 9.3.5: pingall para comprobar conectividad.

En este momento los paquetes presentes en la red muestran la respuesta a los mensajes ARP e ICMP que hacen posible comprobar la conectividad entre los terminales.

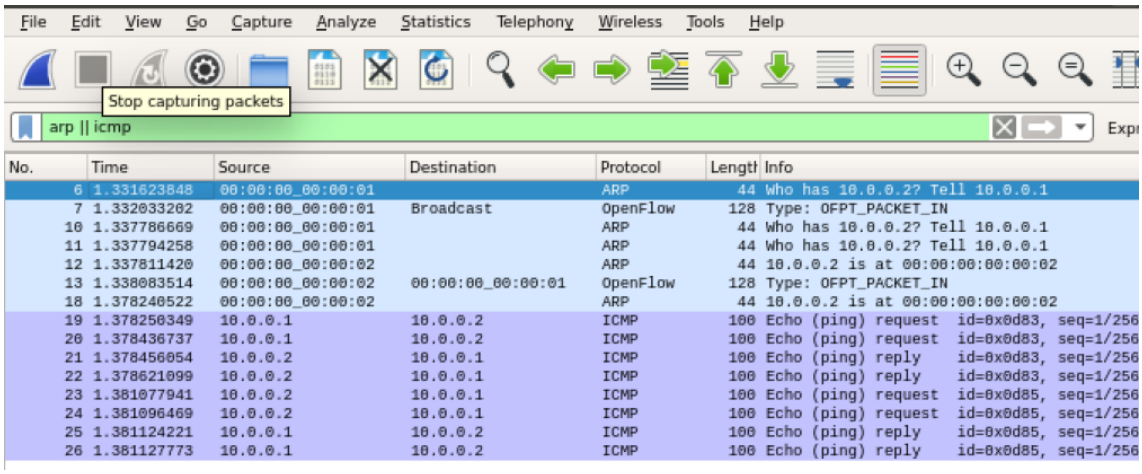


Imagen 9.3.6. paquetes ARP e ICMP capturados con Wireshrak.

9.3.2. POXDesk

Para el inicio de POXDesk se necesita la ejecución de ciertos módulos. Como podemos ver en [45], los módulos necesarios para el correcto funcionamiento de POXDesk son los siguientes:

- samples.pretty_log: no es estrictamente necesario, pero ayuda a la interpretación de los logs que se muestran en la consola.
- web: inicia el servicio web en 127.0.0.1:8000.
- messenger: inicia el servicio de envío y recepción de mensajes.
- messenger.log_service: inicia el log del servicio de mensajes.
- messenger.ajax_transport: módulo encargado de la codificación de mensajes.
- openflow.of_service: inicia el servicio openflow.
- openflow.webservice: inicia el servicio web de openflow.
- poxdesk: inicia la GUI de POX en https://127.0.0.1/poxdesk/resource.
- openflow.discovery: realiza un descubrimiento de la topología y sus relaciones, aunque no descubre equipos finales.
- poxdesk.tinytopo: modulo encargado de la representación de la topología descubierta con el módulo anterior.

De este modo el comando completo para el inicio de POX junto a POXDesk queda de la siguiente manera

```
./pox/pox.py samples.pretty_log web messenger messenger.log_service
messenger.ajax_transport  openflow.of_service  openflow.webservice
poxdesk openflow.discovery poxdesk.tinytopo forwarding.l2_pairs
```

9.4. ANEXO IV. Código en Python de la topología programada.

9.4.1. Código de topoProgramada.py

```
from mininet.net import Mininet
from mininet.node import RemoteController
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink

def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8') #a cada host creado le
asigna la
                                        #primera  ip  libre  de
ese pool

    info( '*** Add controller\n' )
    c0=net.addController(name='c0',
                         controller=RemoteController,
#controlador remoto
                         ip='127.0.0.1',   #IP donde se encuentra
el controlador
                         protocol='tcp',         #protocolo    de
transporte usado
                         port=6633) #puerto para las com. con el
controlador

    info( '*** Add switches\n')
    s1 = net.addSwitch('s1')  #creacion de switch por defecto
    s2 = net.addSwitch('s2')
    s3 = net.addSwitch('s3')
    s4 = net.addSwitch('s4')
    s5 = net.addSwitch('s5')
    s6 = net.addSwitch('s6')
    s7 = net.addSwitch('s7')
    s8 = net.addSwitch('s8')
    info( 's1, s2, s3, s4, s5, s6, s7, s8\n')

    info( '*** Add hosts\n')
    h1  =  net.addHost('h1',  mac='00:00:00:00:00:01')  #cre  un
host por defecto
    h2 = net.addHost('h2', mac='00:00:00:00:00:02')
    h3 = net.addHost('h3', mac='00:00:00:00:00:03')
    h4 = net.addHost('h4', mac='00:00:00:00:00:04')
    h5 = net.addHost('h5', mac='00:00:00:00:00:05')
    h6 = net.addHost('h6', mac='00:00:00:00:00:06')
    h7 = net.addHost('h7', mac='00:00:00:00:00:07')
    h8 = net.addHost('h8', mac='00:00:00:00:00:08')
    h9 = net.addHost('h9', mac='00:00:00:00:00:09')
    h10 = net.addHost('h10', mac='00:00:00:00:00:0a')
```

```
        h11 = net.addHost('h11', mac='00:00:00:00:00:0b')
        h12 = net.addHost('h12', mac='00:00:00:00:00:0c')
        h13 = net.addHost('h13', mac='00:00:00:00:00:0d')
        h14 = net.addHost('h14', mac='00:00:00:00:00:0e')
        h15 = net.addHost('h15', mac='00:00:00:00:00:0f')
        info( 'h1, h2, h3, h4, h5, h6, h7, h8, h9, h10, h11, h12,
h13, h14, h15\n')

        info( '*** Add links\n')
        net.addLink(s1, s2) #crea enlace
        net.addLink(s1, s3)
        net.addLink(s1, s4)
        net.addLink(s2, s5)
        net.addLink(s2, s6)
        net.addLink(s3, s7)
        net.addLink(s3, s8)
        net.addLink(s6, s7)
        h1s1 = {'bw':1000,'delay':'5ms'}         #datos limitacion
enlace
        net.addLink(h1, s1, cls=TCLink , **h1s1) #bw tiene que
estar
        h2s2= {'bw':1000,'delay':'5ms'}           #entre 0 y 1000
Mb
        net.addLink(h2, s2, cls=TCLink , **h2s2)
        h3s2 = {'bw':1000,'delay':'5ms'}
        net.addLink(h3, s2, cls=TCLink , **h3s2)
        h4s5 = {'bw':1000,'delay':'5ms'}
        net.addLink(h4, s5, cls=TCLink , **h4s5)
        h5s5 = {'bw':1000,'delay':'5ms'}
        net.addLink(h5, s5, cls=TCLink , **h5s5)
        h6s6 = {'bw':1000,'delay':'5ms'}
        net.addLink(h6, s6, cls=TCLink , **h6s6)
        h7s6 = {'bw':1000,'delay':'5ms'}
        net.addLink(h7, s6, cls=TCLink , **h7s6)
        h8s3 = {'bw':1000,'delay':'5ms'}
        net.addLink(h8, s3, cls=TCLink , **h8s3)
        h9s7 = {'bw':1000,'delay':'5ms'}
        net.addLink(h9, s7, cls=TCLink , **h9s7)
        h10s8 = {'bw':1000,'delay':'5ms'}
        net.addLink(h10, s8, cls=TCLink , **h10s8)
        h11s8 = {'bw':1000,'delay':'5ms'}
        net.addLink(h11, s8, cls=TCLink , **h11s8)
        h12s8 = {'bw':1000,'delay':'5ms'}
        net.addLink(h12, s8, cls=TCLink , **h12s8)
        h13s4 = {'bw':1000,'delay':'5ms'}
        net.addLink(h13, s4, cls=TCLink , **h13s4)
        h14s4 = {'bw':1000,'delay':'5ms'}
        net.addLink(h14, s4, cls=TCLink , **h14s4)
        h15s4 = {'bw':1000,'delay':'5ms'}
        net.addLink(h15, s4, cls=TCLink , **h15s4)
        info( '*** Starting network\n')
        net.build()
        info( '*** Starting controller\n')
        for controller in net.controllers:
            controller.start()
        info('c0')

        info( '*** Starting switches\n')
```

```
        net.get('s1').start([c0]) #inicia la com. entre s1 y c0
        net.get('s2').start([c0])
        net.get('s3').start([c0])
        net.get('s4').start([c0])
        net.get('s5').start([c0])
        net.get('s6').start([c0])
        net.get('s7').start([c0])
        net.get('s8').start([c0])

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

        CLI(net) #inicia el CLI de Mininet para la interacacion con
el usuario
        net.stop()

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