UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

Estudio del controlador SDN Ryu sobre una Raspberry-Pi Model 4

Alejandro Julián Pachés

Tutor: José Óscar Romero Martínez

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2019-20

Valencia, 26 de mayo de 2020

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS
VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS
Gestión ambiental verificada
REG.NO. ES-CV-000030

Agradecimientos

Me gustaría hacer una mención especial a todas las personas que me han ayudado a llegar hasta aquí; no sólo este trabajo final de carrera en particular, si no durante los cuatro maravillosos pero duros años de carrera. En primer lugar, a mis padres, a mi hermana y a mi pareja por su apoyo incondicional y ánimos en los buenos y en los malos momentos. También a mis amigos de clase que han estado siempre a mi lado en estos años y me han ayudado siempre que me ha hecho falta. Por supuesto, a mi tutor de TFG Óscar Romero ya que sin él, este trabajo no se podría haber llevado a cabo.

Acknowledgments

I would like to make a special mention to all the people who have helped me to get here; not only this particular final degree job, but during the four wonderful but hard years of career. First of all, to my parents, my sister and my partner for their unconditional support and encouragement in good and bad times. Also, to my class friends who have always been by my side in these years and have helped me whenever I have needed. Of course, to my TFG tutor Oscar Romero since without him that this work could not have been carried out.

Resumen

En la actualidad, la demanda de servicios de telecomunicación es tan grande que los operadores deben buscar soluciones muy complejas y que satisfagan la creciente necesidad de acceso a Internet. La poca escalabilidad de las redes tradicionales junto con los requerimientos específicos de cada cliente provoca que las empresas deban invertir mucho tiempo de personal especializado en la configuración de estas redes, lo que genera un aumento de los costes. Con SDN, la red actual, muy estática, puede evolucionar a una red inteligente mucho más abierta, flexible, escalable y reprogramable. Lo que dota a la red de la capacidad de modificar su comportamiento de forma automática en tiempo real, reduciendo costes y complejidad. En este TFG se explicará de manera teórica en qué consisten estas redes y qué pueden ofrecer, acompañándolo de una simulación práctica y evaluación de los resultados. Así mismo, se hará hincapié en la utilización de un controlador SDN Ryu en la simulación de estas redes y qué puede ofrecer. Para la simulación de la red se utilizará el software Mininet y el controlador Ryu, todo ubicado en una Raspberry-pi Model 4.

Resum

En l'actualitat, la demanda de serveis de telecomunicació és tan gran que els operadors han de buscar solucions molt complexes i que satisfacen la creixent necessitat d'accés a Internet. La poca escalabilitat de les xarxes tradicionals junt amb els requeriments específics de cada client provoca que les empreses hagen d'invertir molt de temps de personal especialitzat en la configuració d'aquestes xarxes, el que genera un augment dels costos. Amb SDN, la xarxa actual, molt estàtica, pot evolucionar a una xarxa intel·ligent molt més oberta, flexible, escalable i reprogramable. El que dota la xarxa de la capacitat de modificar el seu comportament de forma automàtica en temps real, reduint costos i complexitat. En este TFG s'explicarà de manera teòrica en què consisteixen aquestes xarxes i què poden oferir al mercat actual, acompanyant-lo d'una simulació pràctica i avaluació dels resultats. Així mateix, es posarà èmfasi en la utilització d'un controlador SDN Ryu en la simulació d'aquestes xarxes i què pot oferir. Per a la simulació de la xarxa s'utilitzarà el programa Mininet i el controlador Ryu ubicat en una Raspberry.

Abstract

Nowadays, the demand for telecommunication services is so great that operators must look for very complex solutions that satisfy the growing need for Internet access. The low scalability of traditional networks together with the specific requirements of each client causes companies to invest a lot of time from specialized staff in the configuration of these networks, which generates an increase in costs. With SDN, the current network, very static, can evolve into a smart network that is much more open, flexible, scalable and reprogrammable. What gives the network the ability to modify its behavior automatically in real time, reducing costs and complexity. This TFG will explain in a theoretical way what these networks consist of and what they can offer to the current market, accompanied by a practical simulation and evaluation of the results. Likewise,

emphasis will be placed on the use of a Ryu SDN controller simulating these networks and what it can offer. For the network simulation, the Mininet software and a Ryu controller located in a Raspberry will be used.

Índice

Índice de figuras

Índice de tablas

Capítulo 1. Introducción

Hoy en día es indispensable vivir sin tener acceso a internet. Los dispositivos con acceso a la red están presentes en nuestras vidas a todas horas y, cada vez, en mayor medida. Según un estudio del 31 de enero del 2020, el 59% de la población mundial utiliza internet en su día a día. Si hablamos de datos y cantidad de información, el volumen global de tráfico de datos anual en 2020 está muy cerca del zettabyte (10^{21} bytes), el tráfico de datos móviles en 2022 será seis veces mayor al de 2017, y eso que hoy en día el 52% de la población mundial utiliza su teléfono para acceder a Internet. Respecto al Internet de las Cosas (IoT), en los últimos años está teniendo un gran impacto y cuenta ya con más de 50 millones de dispositivos, cifra que en dos años será duplicada. Por no hablar de la creciente demanda de los servicios en la nube (Cloud) que permiten a las personas almacenar miles y miles de GB.

Estos datos e información no solo vienen en formato de texto, ya que la información que más almacenamiento ocupa es la multimedia. Millones de archivos de audio, imagen y vídeo son compartidos por la red día a día. Las plataformas de streaming como Netflix o HBO son una herramienta muy popular entre la población, y es que en 2020 el porcentaje de usuarios que utilizan estos servicios o IPTV es superior al porcentaje de usuarios que utilizan la televisión TDT, pese a que ésta sigue teniendo todavía un fuerte impacto.

Además, la velocidad de las redes ha pasado de 8,7 Mbps de media a 28,5 Mbps. Y es que para que un usuario pueda disfrutar de todos estos servicios sin elevados tiempos de espera debido a su conexión, necesitan mejores velocidades de banda ancha.

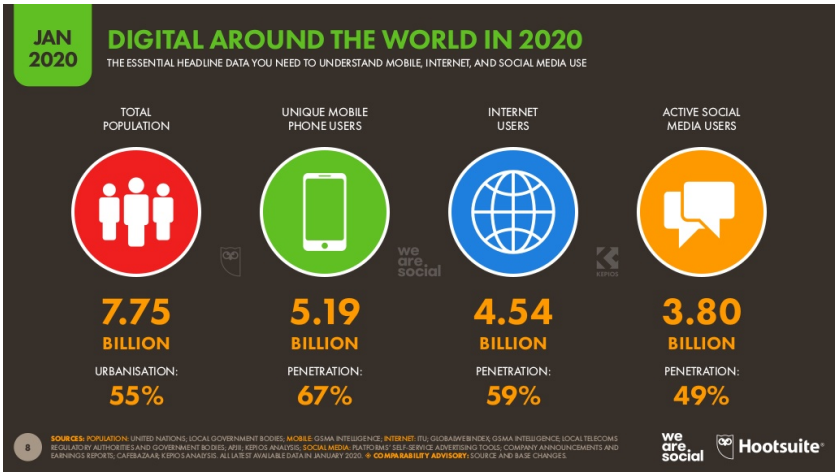


Figura 1: Estadísticas uso de Internet enero 2020

Fuente: marketing4ecommerce

Todos estos datos dejan ver la creciente demanda que tienen los usuarios por utilizar grandes cantidades de datos y a gran velocidad, lo que supone un importante reto para todas las empresas del sector de las telecomunicaciones. Las redes tradicionales soportan cada vez más débilmente toda esta transformación debido a sus limitaciones, como son por ejemplo la complejidad de los protocolos de red, el tiempo necesario para reconfigurar y gestionar los elementos de una red cuando es necesario, la poca escalabilidad que tienen a la hora de aumentar el tamaño de la red o la dependencia que hay del vendedor en los

ciclos de producción de los equipamientos. Es por ello que las corporaciones están comenzando a buscar soluciones a este problema. Aquí es donde SDN se convierte en la estructura de red deseada, con el potencial de revolucionar el mundo de las redes.

Las redes definidas por software (SDN por sus siglas en inglés Software-defined Networking) es una arquitectura de redes ágil que permite a la red ser controlada de manera inteligente y centralizada utilizando aplicaciones de software. Las redes SDN permiten diseñar, desarrollar y administrar redes mediante la separación de los planos de control (software) y reenvío (hardware).

Para conocer cómo funciona, podemos comparar sus "tres capas": la de aplicación, control e infraestructura con un portátil en el que, por un lado, están las aplicaciones, por otro, el propio hardware del equipo y, por último, el sistema operativo, que es el que hace que interactúen y operen entre sí los anteriores. Podríamos, por tanto, afirmar que SDN será "el sistema operativo de las redes".

La tecnología SDN ha aparecido a raíz de diversas innovaciones que se iniciaron hace más de 20 años, con la aparición del Internet que conocemos hoy en día, y que intensificó la necesidad de evolucionar hacia las redes programables. A principios de los 2000 surgió la idea de separar la capa de datos y la de control debido al gran aumento del volumen de tráfico y el requerimiento de una red más manejable.

Casi diez años más tarde, investigadores de la universidad de Stanford crearon el Clean Slate Program para la experimentación de redes universitarias más tratables y locales, que dio lugar a Openflow. Openflow es un estándar abierto de la arquitectura SDN que permite al plano de control interactuar con el de datos.

La red que permite obtener SDN consiste en tres elementos básicos: los controladores SDN, las API descendentes y las API ascendentes. Estos elementos, de forma conjunta, permiten obtener diversos beneficios como la implementación rápida y automatizada de aplicaciones, movilidad a gran escala, mayor flexibilidad y reducción de los costos.

Vistas las ventajas que ofrecían este tipo de redes a las corporaciones, operadores y proveedores de servicio, y debido al crecimiento monumental del contenido multimedia, la computación en la nube, el impacto del uso del móvil y la necesidad de reducir costos a la empresa, se comenzó a recurrir a esta nueva tecnología para solucionar todas estas fuerzas competitivas.

En 2011 se funda la Open Network Foundation (ONF) que es el consorcio industrial que lidera la evolución SDN y la estandarización de los elementos críticos de esta arquitectura. Algunas de las compañías que lo forman son Deutsche Telekom, Facebook, Google, Microsoft o Yahoo!. Pero en la actualidad cuenta con más de 140 miembros.

El IDC calcula que el mercado mundial de SDN de centros de datos valdrá más de doce mil billones de dólares en el año 2022. Lo que permite ver la importancia que está comenzando y va a tener la arquitectura de redes SDN.

1.1 Motivación

La principal motivación por la que he decidido elegir el tema de las redes definidas por software es debido a que es una tecnología muy innovadora y puntera, y que en el futuro va a ser de vital importancia no solo a nivel empresarial, si no a nivel global ya que debido a la gran dependencia y cantidad de datos que circulan día a día por la red, la evolución hacia las redes SDN va a ser inevitable. Y es que en el futuro será fundamental tener conocimientos de esta tecnología en el ámbito de la telemática.

Además, me parece muy interesante la forma en la que se ha decidido solucionar los problemas de las redes tradicionales mediante la separación de la capa de control y datos, que más adelante explicaré detalladamente. Así como el hecho de virtualizar, como si de un programa se tratara, toda una red que desde su aparición en los años 80' ha sido siempre muy estática y cerrada.

Por otro lado, la posibilidad de poder simular virtualmente un entorno SDN tanto en un ordenador personal como en una Raspberry-Pi mediante la utilización de cierto software, sin demasiadas dificultades, y sin la necesidad de hardware específico, hace del estudio de este tipo de redes algo muy interesante para el estudiante, que puede profundizar en el aprendizaje de las arquitecturas de red sin la necesidad de estar en un laboratorio.

1.2 Objetivos

Los principales objetivos de esta tesis son explicar en qué consiste y cómo funciona una tecnología tan innovadora y actual como la arquitectura SDN, que ventajas tiene respecto a las redes tradicionales y cómo van a afectar al mercado tecnológico actual. Así mismo, se hará especial mención en la utilización del controlador SDN Ryu, un controlador open source que nos permite multitud de posibilidades a la hora de trabajar con este tipo de redes.

En este trabajo se quiere también concienciar al lector de las cada día más limitadas redes tradicionales y de la necesidad de virtualizar y utilizar la tecnología SDN para poder soportar el exponencial crecimiento de internet. La evolución hacia las redes SDN es un paso obligatorio que debe dar el sector de las telecomunicaciones, y en este proyecto se quiere persuadir al lector mostrando el elevado potencial que pueden tener este tipo de redes respecto a las tradicionales.

El entorno de trabajo sobre el que trabajaremos en este proyecto es una Raspberry-Pi Model 4, que es un ordenador de capa reducida al cual le hemos instalado un sistema operativo basado en Linux que nos permitirá implementar el software necesario de este proyecto, que es Mininet y Ryu.

Por último, se pretende mostrar la potente herramienta Mininet, que permite simular una red SDN y ver su funcionamiento, lo que podría ser objeto de estudio en prácticas de laboratorio de las asignaturas de telemática. Mediante esta herramienta implementaremos diferentes escenarios SDN que utilizarán el controlador Ryu comentado anteriormente.

Con todo ello, se realizará un estudio de los resultados obtenidos para comprobar la eficacia, correcto funcionamiento y gestión de una red virtualizada que utiliza el controlador Ryu, concluyendo con una reflexión sobre esta tecnología y su futura transcendencia en el mercado de las telecomunicaciones.

1.3 Contenidos y organización del trabajo

Los contenidos de este trabajo son:

- Breve estudio de las redes tradicionales actuales
- Estudiar los conceptos teóricos de las redes definidas por software
    - Conocer los componentes de las SDN
    - Conocer la arquitectura de estas redes
- Estudiar las ventajas y desventajas de las redes SDN
- Comparativa con las redes tradicionales
- Contextualizar la tecnología SDN en el actual y futuro mercado tecnológico
- Comentar los desafíos de seguridad de las SDN
- Explicación teórica del software y hardware utilizado para la simulación práctica
    - Mininet y Miniedit
    - Controlador Ryu
    - Raspberry-pi
- Creación de diferentes entornos virtuales SDN, junto con sus simulaciones, obtención y análisis de los resultados para mostrar diversas aplicaciones del controlador Ryu.

La organización llevada a cabo para la realización del proyecto consiste en:

- Obtención de información: En esta primera parte se busca una primera toma de contacto con las redes SDN y conocer el contexto en el que éstas aparecieron.
- Investigación y estudio teórico de Mininet: En esta sección aprendemos en que consiste Mininet y qué permite realizar, así como dónde descargarlo y cómo.
- Investigación y estudio teórico de la Raspberry-pi: Conocer qué especificaciones tiene y qué permite realizar una Raspberry-Pi Model 4.
- Investigación y estudio teórico del controlador Ryu: Lectura de la documentación de Ryu, así como de otras páginas para conocer qué permite realizar este controlador.
- Instalación de una VM y formación en Python: Se ha decido implementar el proyecto primero en una VM para ver los posibles errores de instalación que puedan surgir y cuál es la mejor forma de instalar el software.
- Instalación de Mininet, Miniedit y Ryu en la VM: Depuración de errores de instalación debido a versiones de Python, librerías etc.
- Toma de contacto con el software: Elaboración de distintos entornos de simulación básicos y pequeñas pruebas para conocer las herramientas y el potencial de ellas.
- Instalación del SW necesario en la Raspberry: Instalación del sistema operativo Raspbian en la Raspberry, así como de Mininet y Ryu.
- Creación de diversos entornos de simulación con Ryu: Elaboración de las distintas topologías según las aplicaciones que queremos implementar con el controlador Ryu.
- Simulación de las redes y análisis de los resultados: Ejecución de los archivos Python en el controlador Ryu y respectivas pruebas para comprobar el correcto funcionamiento de las distintas aplicaciones.
- Obtención de conclusiones: Recapitulación de información teórica y práctica obtenida durante la tesis y elaboración de una conclusión del proyecto.

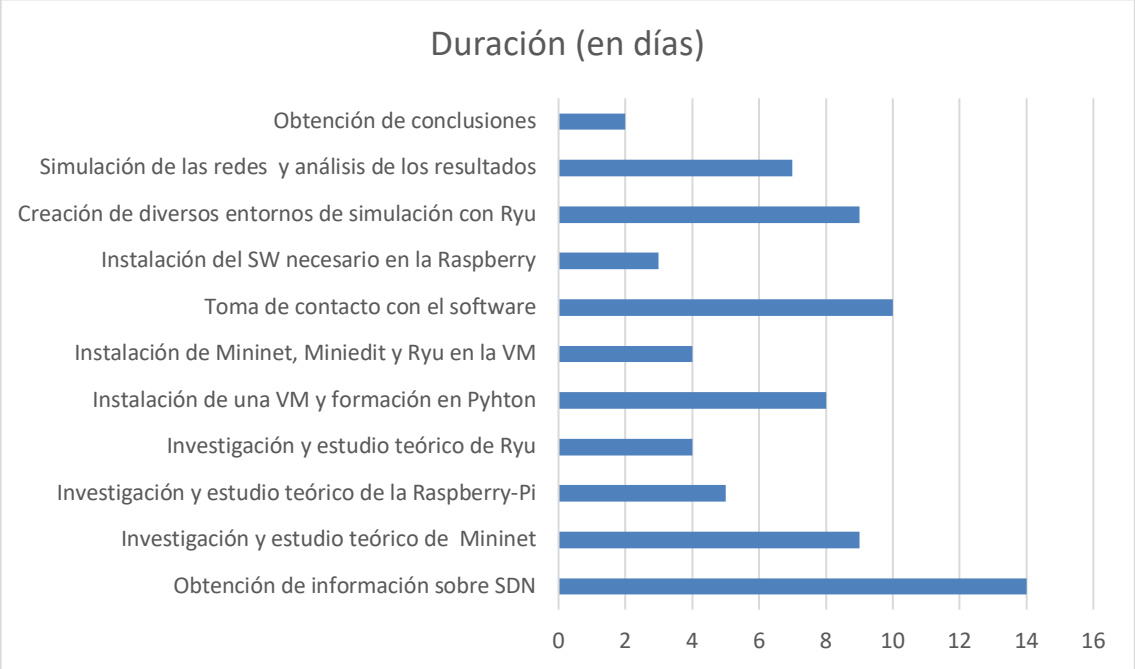


Diagrama temporal del proyecto.

Capítulo 2. Contexto tecnológico actual

2.1 Redes tradicionales

A través de los años, la arquitectura de las redes ha ido evolucionando a pasos agigantados. Desde el servicio telefónico tradicional (POTS por sus siglas en inglés Plain Old Telephone Service) de finales del siglo XIX hasta las actuales redes basadas en el protocolo TCP/IP como OSPF, MPLS o MetroEthernet. La razón por la que las redes han ido evolucionando es debido a los grandes avances tecnológicos y a las nuevas necesidades de las personas.

Las redes IP convencionales conectan lugares a grandes distancias y satisfacen las necesidades de conectividad de sus usuarios. Para realizar cada una de estas operaciones cada paquete tiene que pasar por routers, switches, firewalls y demás elementos los cuales toman decisiones individuales de enrutamiento que hacen difícil la gestión centralizada de la red.

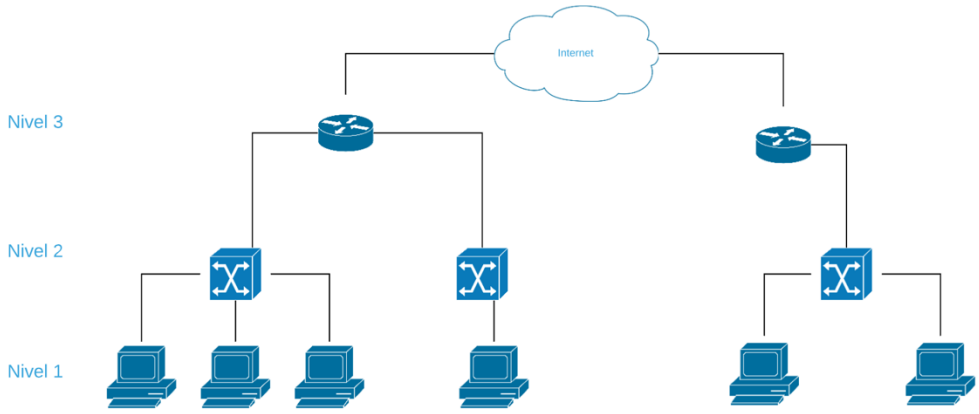


Figura 2: Red Tradicional

Fuente: Propia

La estructura y el modo de funcionamiento de las redes informáticas actuales están definidos en varios estándares, siendo el más importante y extendido de todos ellos el modelo TCP/IP utilizado como base para el modelo de referencia OSI. La comunicación de estas redes se lleva a cabo en dos diferentes capas: la capa física y la capa lógica.

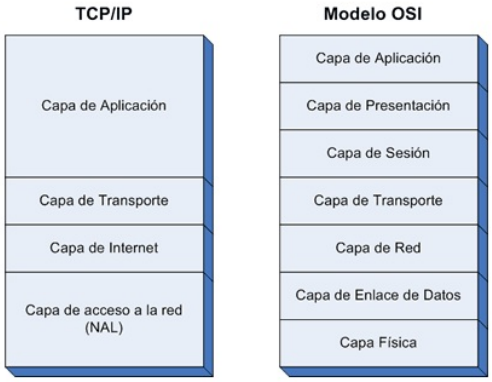


Figura 3: Comparativa modelos OSI-TCP/IP

Fuente: Javiergarzas.com

La capa física incluye todos los elementos de los que hace uso un equipo para comunicarse con otros equipos dentro de la red, como, por ejemplo, las tarjetas de red, los cables, las antenas, etc. Es decir, la capa física está formada por el hardware de la red que permite transmitir y recibir datos. Por otra parte, la capa lógica está formada por los protocolos de comunicación que rigen internet y que proporcionan servicios.
Sin embargo, este tipo de redes cada vez soportan más débilmente los grandes avances de internet. Pese a que las grandes multinacionales intentar renovar estos tipos de redes para cubrir estas necesidades, no son soluciones definitivas ya que para poder paliar este problema sería necesario un nuevo concepto de arquitectura de red. Las arquitecturas antiguas construidas mediante varias capas de acceso, agregación y core no son adecuadas para los nuevos perfiles de tráfico. No basta con ampliar el número de interfaces de los equipos existentes o el aumento de velocidad de dichas interfaces, sino que se necesita un cambio de arquitectura y diseño de la red.

Y es que las redes actuales tienen bastantes limitaciones:

- **Dificultad**: Las redes tradicionales están formadas por multitud de protocolos diferentes. Cada modelo estructura el funcionamiento de una red de manera distinta. Los protocolos están repartidos por las diferentes capas, y éstos han sido definidos independientemente los unos de los otros debido a que cada protocolo se encarga de resolver una problemática. La coexistencia de multitud de protocolos provoca una gran dificultad de configuración y gestión a la hora de querer modificar la red para adaptarla a cambios.
- **Rigidez**: Ante un cambio de topología que se quiera realizar, o bien ante fallos o inconvenientes que pueda presentar la red, ésta ofrece un elevado grado de inflexibilidad. Y es que, pese a que la propia red es capaz de detectar los fallos, recalcular las rutas y actualizar las tablas de encaminamiento automáticamente, es un proceso lento que muestra cómo de estática es la red.
- **Imposibilidad de escalabilidad**: En los últimos años, las empresas de ven obligadas a aumentar el tamaño de la red. Sin embargo, a medida que la red crece se

vuelve más difícil de controlar debido a la gran cantidad de elementos de red que deben ser configurados y gestionados.

- **Dependencia del vendedor**: Los equipamientos de la red no evolucionan a la vez que las redes y es que, los ciclos de producción de estos elementos por parte de las grandes empresas vendedoras abarcan más de tres años. Además, muchos protocolos de red provienen de un mismo fabricante que limita la infraestructura de la red a equipos de este mismo.
- **Saturación**: El gran incremento del volumen de datos actual provoca una elevada saturación en los elementos y en la utilización de las rutas más eficientes de la red. Pese a que hoy en día ya existan técnicas para mejorar la calidad del servicio (QoS por sus siglas en inglés Quality of Service) o técnicas de fragmentación, el administrador de la red se ve periódicamente en la obligación de configurar manualmente la red para reducir la latencia y satisfacer las necesidades de las nuevas tecnologías.
- **Inseguridad**: La complejidad de tráficos en la red puede ocasionar problemas de seguridad, que necesariamente deben ser abordados desde el acceso a la red. Además, las soluciones utilizadas por las empresas para poder escalar la red, a menudo muy robustas, son puntos débiles que dan lugar a ciberataques. Por último, debido a la poca centralización que ofrecen este tipo de redes, la recopilación de datos y estadísticas del tráfico de una red se vuelve compleja.

2.2 Demanda tecnológica actual

Como ya hemos comentado, el marco tecnológico actual ha evolucionado a pasos agigantados en la última década. El gran impacto de los dispositivos con acceso a internet en nuestras vidas ha provocado un incremento de la cantidad de datos e información que son día a día transmitidos por las redes, las cuales están siendo llevadas a sus límites. Lo que ha provocado la necesidad de transformar la arquitectura de red actual. Algunos de estos motivos de querer esta transformación son:

- **El Big data**: Es el conjunto de la gran cantidad de datos que hay hoy en día junto con la idea de explotar comercialmente estos datos mediante técnicas de procesado para crear nuevos servicios comerciales, manejándolos de una forma más estructurada. La aparición del big data requiere un procesamiento masivo de datos por parte de los servidores de la red, junto con una constante demanda de ancho de banda.
- **Los servicios Cloud**: Es una tecnología que permite ofrecer servicios de computación a través de una red que suele ser internet. Estos servicios aumentan en gran medida la complejidad de la red y la necesidad de aumentar la seguridad de ésta. Además, requiere una escalabilidad dinámica de los recursos y almacenamiento que las redes de hoy en día difícilmente pueden proveer.
- **Eliminación del modelo cliente-servidor**: Las aplicaciones y servicios de internet han evolucionado desde un simple modelo entre el cliente y el servidor hacia numerosas arquitecturas como las redes P2P, aplicaciones MVC (modelo vista controlador), utilización de bases de datos o posibilidad de conectarse a las aplicaciones desde distintos puntos, entre otros.
- **Costes temporales y materiales de las empresas**: La creciente demanda está provocando que las empresas desvíen muchos recursos económicos en conseguir

reducir el tiempo de configuración de las redes, así como la contratación de técnicos profesionales que administren y gestionen éstas.

Una vez vistas las limitaciones que tienen las redes actuales y las necesidades tecnológicas que hay hoy en día, las empresas entendieron que necesitaban un cambio en el modelo arquitectónico actual, y no valía seguir con introduciendo parches para solucionar problemas puntuales. En este contexto, surgen las redes definidas por software (SDN). Pero ¿qué son las SDN? ¿cómo funcionan? ¿qué ventajas tienen respecto a las redes tradicionales?

Capítulo 3. Fundamentos de las redes definidas por software

3.1 Introducción

Las redes definidas por software o software-defined networking (SDN) son el resultado de cientos de pruebas que se han llevado durante años con la idea de buscar soluciones a los límites que tenían las redes tradicionales actuales.

Las SDN es un tipo de arquitectura de red ágil que surge con el concepto de querer virtualizar las redes para que así puedan ser dinámicas, es decir, que puedan programarse y reprogramarse fácilmente sin un elevado requerimiento de personal y tiempo, y siempre de una forma inteligente, dinámica y centralizada, mediante el uso de aplicaciones de software.

La arquitectura de las redes definidas por software se ha realizado partiendo de los fundamentos básicos de un conmutador de tráfico tradicional, el cual está compuesto internamente de dos planos, el plano de control que se ocupa de las tareas virtuales y el plano de datos que se ocupa de las labores físicas.

El enrutador tradicional principalmente tiene dos funciones:

• En el plano de control la función principal es obtener información de la topología de la red y actualizarla cuando sea necesaria, almacenando la información localmente en tablas. La actualización se lleva a cabo principalmente mediante protocolos como los de enrutamiento, que eligen la mejor ruta a seguir por los paquetes según distintos criterios

• En el plano de datos la tarea principal es reenviar los paquetes según las instrucciones que haya tomado el plano de control.

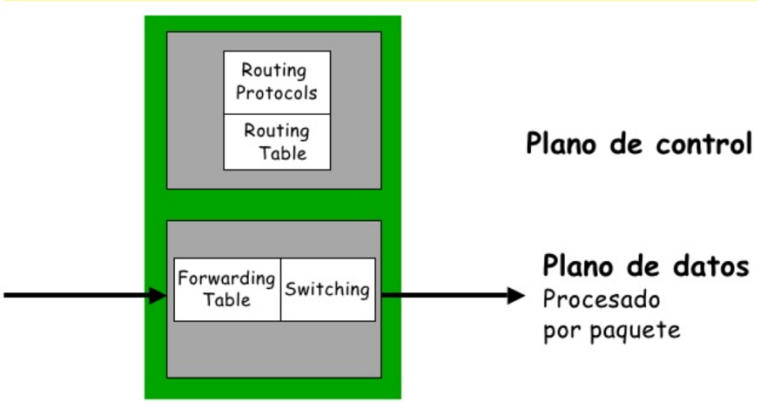


Figura 4: Arquitectura de un enrutador

Fuente: Área de Ingeniería Telemática de la Universidad de Navarra

Cuando un conmutador recibe un paquete, las reglas integradas en el firmware propietario del dispositivo le dicen al conmutador a qué dirección tiene que transmitir el paquete. Sin embargo, el proceso de enrutamiento tenía problemas como el control de los elementos de la red o la heterogeneidad en los procesadores de red, por lo que era necesario implementar algún protocolo a la hora de implementar el plano de datos con el de control.

Así pues, las empresas e investigadores decidieron investigar con el fin de intentar evitar la alta integración que tenían los planos de control y datos.

Estos avances se llevaron a cabo en la arquitectura SDN, dónde en vez de integrar conjuntamente estos dos planos, lo que se quería era separarlos, es decir, que se implementaran en hardware independiente. La división en planos hace más eficiente la conmutación de los paquetes, ya que sí la información necesaria para procesarlos se encuentra en la tabla de reenvío, no es necesario la intervención del plano de control, además el hecho de que los planos residan en dispositivos diferentes viene ligado junto con la aparición del switch SDN, dónde residirá el plano de datos, y el controlador SDN, donde se ubicará el plano de control. Debido a esta reubicación de los planos de control y datos en distintos dispositivos, será necesario un punto de comunicación entre ellos; este punto será una interfaz API. La API permitirá al controlador SDN gestionar el plano de datos.

A través de la nombrada división de planos, se consigue la posibilidad de realizar una configuración de manera remota y dinámica en todos los dispositivos de la red, así como de proporcionar una visión global de la topología que permite elaborar acciones más exactas. Y todo esto, de una forma centralizada, a través del controlador. Todo esto permite ahorrar tiempo en configuraciones locales y específicas para cada dispositivo.

A modo de resumen, podríamos decir que una red definida por software permite:

- Implementar y ampliar con mayor facilidad las funciones de la red.
- Diseñar el tráfico con una vista global de la red.
- Utilizar mejor los recursos de la red.
- Ser compatible con el dinamismo, duplicación y asignación de los recursos virtuales.
- Facilitar la carga administrativa de la configuración y suministro de funciones como calidad de servicios y seguridad.
- Reducir la complejidad.
- Habilitar aplicaciones para solicitar dinámicamente servicios de la red.
- Reducir el OPEX (OPerational EXpenditures) que son los gastos de funcionamiento.

3.2 Arquitectura SDN

La idea principal de las redes SDN consiste como se ha comentado anteriormente en el concepto de querer separar el plano de control y el plano de datos, permitiendo al gestor de la red programar el control de la red abstrayéndose de la capa de infraestructura.

Para separar estos dos planos, se propone trasladar el plano de control de los elementos físicos de la red como routers o switches a un nuevo elemento externo a la red, llamado controlador SDN. Este controlador es el dispositivo software que se encarga de tomar las decisiones de reenvío de los paquetes y construir y actualizar las tablas de enrutamiento de los elementos físicos. Además, con el uso de este dispositivo, se consigue controlar toda la red y configurar sus dispositivos físicos desde un solo punto centralizado, mediante el uso de un software de gestión el cual no depende de un fabricante en particular si no que es estándar. Este último punto es importante ya que las redes tradicionales, cerradas y dependientes de los fabricantes, evolucionan hacia un sistema abierto dónde

ya no existe esta dependencia. La posibilidad de que se pueda soportar un tejido de conmutación a través de hardware de múltiples vendedores y circuitos integrados de aplicación específica hace que la red SDN sea conocida como “la asesina de Cisco”.

- Plano de control: El plano de control es, como su nombre indica, el software que controla la red. El plano de control está a cargo del sistema operativo. Este plano se puede dividir en dos capas, la capa de control y la capa de aplicación.
    - Capa de control: Esta capa ofrece los servicios de gestión y adaptabilidad red. Está formada por el elemento principal de las redes SDN, el controlador o *controller*. Esta capa es ‘el cerebro’ de la red.
    - Capa de aplicación: Aquí se ofrecen las aplicaciones de negocio. Está formada por el software que se comunica con la capa de control y que permite transmitir al controlador SDN las necesidades y comportamientos deseados de la red.
- Plano de datos: Constituye todo lo referente a la lógica de reenvío de paquetes en hardware y tráfico de datos. Se encarga de manejar las colas de salida, modificar las cabeceras de control y reenviar información. Está formada por todos los elementos que constituyen una red de conmutación y que están interconectados entre sí, como por ejemplo los routers o los switches, por ello este plano es comúnmente llamado capa de infraestructura.

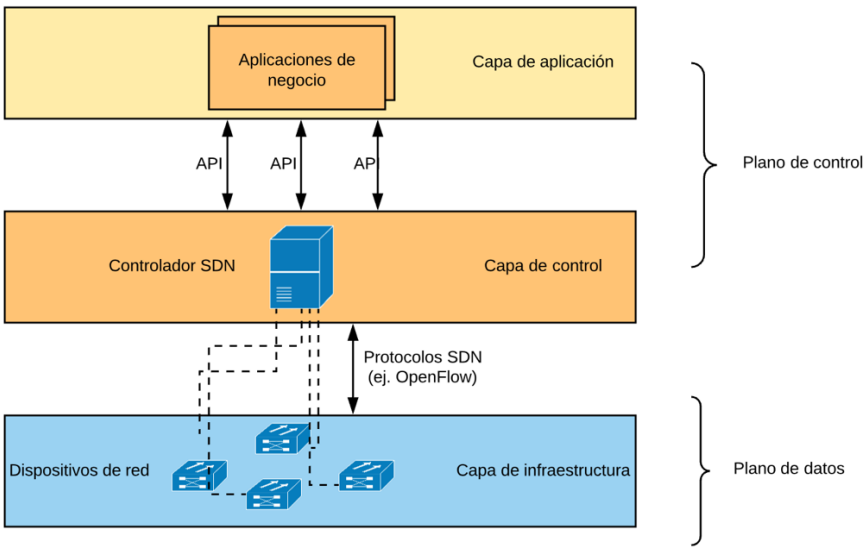


Figura 5: Arquitectura SDN

Fuente: Elaboración propia

Las formas de implementar una red SDN varían según el tipo de estructura que ésta tenga; así, las redes SDN pueden dividirse en:

SDN simétrica y asimétrica

Aunque el principio basado en software prevé la mejor concentración posible de la inteligencia de los dispositivos de red, hay otras maneras de utilizar una SDN en el que las tareas del plano de control se dividen en diversas unidades, es lo que se conoce como SDN asimétrica. Estas unidades de control disponen de la mínima información necesaria para el funcionamiento de la red por lo que, si una unidad de control cae, la red puede seguir funcionando. Sin embargo, se pueden producir algunos inconvenientes:

	Característica principal	Ventajas	Desventajas
SDN simétrica	Centralización del control	Se evitan redundancias y se reduce la carga de los componentes	La disponibilidad y estabilidad de la red dependen sólo de la unidad central de control.
SDN asimétrica	Distribución del control	Los sistemas siguen funcionando en caso de caída de elementos de control.	La gestión de la red se vuelve compleja y se produce información redundante

Tabla 1: Comparación entre SDN simétrica y asimétrica

SDN basada en el host o en la red

Según la posición de la lógica de control, el procesamiento de la red definida por software se puede distribuir en routers dedicados, al igual que en las redes tradicionales, y gestionarlos en la red.

Sin embargo, si el software SDN se ejecuta en el mismo sistema host donde se encuentra el administrador de la máquina virtual, se tiene la seguridad de que las capacidades para soportar la carga de datos siempre estarán disponibles.

SDN proactiva y reactiva

Según la forma en la que se reenvían los datos entre los planos de control y datos podemos hablar de un modelo de despliegue SDN proactivo o reactivo.

En el modo proactivo o flood-based la instancia de control se anticipa y envía los datos nuevos y las modificaciones a todos los nodos de la red. Este envío se puede hacer por broadcast o multicast, y es útil cuando se sigue un modelo simétrico ya que el control está centralizado. Sin embargo, cuando aumenta el tamaño de la red y el número de nodos que hay en ella, la carga de la red aumenta, lo que limita la escalabilidad de la red.

Las redes grandes suelen utilizar como alternativa el modelo reactivo o floodless en el que el plano de control garantiza el correcto funcionamiento de sus componentes mediante una transmisión controlada y reactiva de los datos, ya que sólo se utilizan los dispositivos implicados, los cuales extraen los datos importantes de las tablas de consulta (lookup table).

	Transmisión de mensajes	Ventajas	Desventajas
SDN proactiva	Tanto broadcast como multicast	Es fácil de implementar y se utiliza el algoritmo de camino más corto para enviar el paquete	La carga de la red aumenta a medida que se incrementa el número de nodos.
SDN reactiva	Uso de tablas de consulta	Los dispositivos obtienen sólo la información que necesitan	Los problemas en la entrega de información provocan retrasos.

Tabla 2: Comparación entre SDN proactiva y reactiva

3.2.1 Capa de aplicación

Es la capa de más alto nivel. Desde ésta se controla el comportamiento de la red utilizando la API del controlador, la cual puede ser utilizada por usuarios u otros softwares. Las aplicaciones de negocio de la red que se encuentran en esta capa establecen una conexión con la capa de control y le comunican los requisitos o necesidades de la red.

Hay diversas aplicaciones de negocio, pero entre ellas encontramos:

- Mantenimiento de la red: El hecho de tener un control centralizado y una visión global de la red desde el controlador facilita en gran medida el mantenimiento de la red. En SDN se comienzan a utilizar herramientas para realizar diagnósticos como OFRewind o ndb.
- Enrutamiento adaptativo: Se basa principalmente en el uso de dos técnicas, el balanceo de carga y el diseño de intercambio de información entre capas. Esta forma de enrutamiento permite integrar mejor las entidades y proporcionar QoS, entre otras ventajas.
- Intinerancia continua: Gracias al sistema abierto que permite SDN con distintos fabricantes y el control centralizado, surgen nuevas propuestas para ofrecer servicios sin interrupciones como la red europea de Wi-Fi sharing entre operadores (AOTEC e Intracom Telecom) que permite que un usuario tenga conectividad Wi-Fi en todas las localidades donde exista un operador local.
- Mejora de la seguridad de la red: Otra ventaja del control centralizado y la visión global de la red es que le permite al controlador poder analizar patrones en el tráfico de red para descubrir incidencias de seguridad como ataques de denegación de servicio (DoS por sus siglas en inglés Deny of Service), modificar reglas de reenvío, dar prioridades o limitar la privacidad de los usuarios o incluso modificar el destino de paquetes dudosos hacia sistemas de prevención de intrusión (IPS)
- Computación en la nube: Es, como ya hemos comentado, uno de los factores que ha provocado la búsqueda de soluciones como SDN. Y es que SDN se aprovecha de la conmutación virtual para establecer una comunicación entre maquinas virtuales que pertenecen al mismo host.
- Virtualización: La virtualización de la red hace uso de dos tecnologías, SDN y NVF (virtualización de funciones de red). La NVF permite a cada nodo

convertirse en un microcentro de datos capaz de ejecutar diversas aplicaciones, lo que flexibiliza la red. Con la implementación de estas tecnologías se consigue centralizar en un software todos los servicios que se ejecutan tradicionalmente en el hardware de un CPD, como el enrutamiento o la optimización WAN y Firewall. Para implementarla se suelen utilizar librerías de virtualización de red como libNetVirt en el controlador o FlowVisor, que es un software que permite elaborar instancias de redes virtuales y mejorar el nivel de abstracción de la capa de virtualización de red a través de la separación de los conmutadores, lo que permite obtener diversas zonas de gestión interconectadas utilizando un solo controlador.

3.2.2 Capa de control

La capa de control es la ubicación del controlador, el elemento básico de la red SDN, aunque puede haber más de un controlador. Estos son quienes mantienen la inteligencia de la red, ya que se encargan de centralizar la lógica, gestionar la capa de aplicación y la de infraestructura y controlar el comportamiento de la red, monitorizando todos los elementos de ésta.

Pese a que un solo controlador cumple con los requisitos básicos de una red SDN. Es recomendable disponer de más de un controlador en la red ya que cuánto mas grande es la red, mayor transferencia de información habrá y esto puede dificultar la visión global de la topología, el tiempo de toma de decisiones o limitar la escalabilidad. Por lo cual, implementar otro controlador permite distribuir el trabajo o bien amenizar labores de control y gestión. Además, uno de los puntos débiles de la arquitectura SDN es que el hecho de que todo el control esté centralizado en un dispositivo produce que éste sea objeto de ataques de ciberseguridad como el de denegación de servicio (DoS).

Una de las herramientas más extendidas para la utilización de múltiples controladores es HyperFlow. HyperFlow es una extensión de OpenFlow que permite un control basado en eventos distribuidos. Pese a que HyperFlow está lógicamente centralizado, permite ser distribuido físicamente, proporcionando escalabilidad y manteniendo los beneficios de la centralización del control de red. El hecho de sincronizar pasivamente las vistas de los controladores de red permite minimizar el tiempo de respuesta en la comunicación entre el plano de control y el de datos.

Los controladores se comunican con el hardware de la capa de infraestructura mediante protocolos como OpenFlow con el fin de obtener información del estado de la red. Con la información que reciben obtienen las necesidades de la red y actúan en respuesta a ellas. Algunas de las tareas que se pueden realizar desde esta capa son actualizar las tablas de reenvío de los dispositivos, migraciones hacia máquinas virtuales (VM) o realizar balanceo de carga.

A los controladores se les brindan una API abierta para poder ser programados desde la capa de aplicación. La información obtenida de la capa de infraestructura se envía a la capa de aplicación para que ésta realice actuaciones como gráficos de las topologías existentes en ese momento, las cuales se obtienen analizando el flujo entrante y saliente. El ejemplo más extendido es la creación de Matrices de Tráfico (TM).

La comunicación del controlador es constante y se utilizan lenguajes de programación como Java, C++ o Python. Según el lenguaje utilizado encontramos diferentes controladores como Maestro (Java) o NOX-MT (C++). Que la comunicación sea constante puede producir algunos problemas a la hora de realizar las configuraciones,

aunque han aparecido algunas herramientas que solucionan este problema como es el caso de VeriFlow o FlowChecker. Otras herramientas como Cbench permiten comparar el rendimiento que ofrecen los controladores.

3.2.3 Capa de infraestructura

En esta capa reside el hardware. Los componentes que la componen son los dispositivos de red, pero sólo su hardware, sin ningún tipo de inteligencia. Pueden ser equipos simples de bajo coste y no tienen que ser del mismo fabricante, sólo deben de contar con el método que les conecta al controlador, es decir, un protocolo como OpenFlow.

Los dispositivos hardware realizan dos funciones o tareas:

- Tarea de control: Aquí se produce la comunicación con el plano de control, es decir, con el controlador. Los dispositivos le transmiten al controlador la información de la red como la topología o las estadísticas de tráfico y éste le manda las instrucciones de reenvío según las necesidades que ha identificado de la red. El dispositivo debe contar con una unidad temporal de almacenamiento para guardar la información recibida del controlador.

- Tarea de datos: Los dispositivos se encargan de redirigir el tráfico según las instrucciones que reciban del plano de control. Cuando se recibe un paquete, éste se analiza y se identifica la instrucción de reenvío tomada en la capa superior.

En esta capa destacan las tablas de reenvío, que se basan en el tratamiento de flujo de datos de los paquetes desde su origen a su destino. Las acciones llevadas a cabo en cada flujo son definidas por las reglas de reenvío del controlador. Estos flujos son unidireccionales, por tanto, si se quiere controlar también la comunicación en el otro sentido, serán necesarias reglas de reenvío para el otro flujo.

Las tablas de flujo constan de un número de entradas de flujo con prioridad, las cuales están compuestas por dos elementos: los campos de coincidencia (match fields) y las acciones. Los campos de coincidencia permiten comparar (buscar coincidencias) los paquetes entrantes. Cuando el paquete entra se comparan los campos de coincidencia de uno en uno, por orden de prioridad y, cuando se encuentra una coincidencia completa, se selecciona. Si se han encontrado coincidencias en los campos de coincidencia especificados por ese flujo, se llevarán a cabo las acciones de reenvío establecidas por el controlador. El propio controlador puede añadir, editar o eliminar entradas de la tabla de flujos.

La actualización de las tablas de reenvío se hace en base a las tablas de flujo, y esta actualización puede ser de dos modos:

- Modo reactivo: El conmutador recibe un paquete que no coincide con ninguna entrada en la tabla de flujos. Entonces envía una notificación al controlador. El controlador creará una nueva regla de renvío y enviará una actualización a la tabla de flujo de forma que el conmutador ya pueda procesar el paquete. El resto de los paquetes del mismo flujo mencionado se transmitirán según la nueva regla y no será necesaria enviar de nuevo una notificación al controlador.

- Modo proactivo: El controlador detecta un cambio en la red como por ejemplo saturación de un enlace, el fallo de un equipo o puerto la caída de enlaces.

Entonces envía una actualización a las tablas de flujo de los conmutadores. El hecho de que la actualización se realice antes de que lleguen los paquetes al conmutador permite minimizar los tiempos de procesamiento de los paquetes.

También es posible que el reenvío normal del paquete por el puerto correspondiente o bien la eliminación de un paquete debido a que no se ha hallado ninguna coincidencia con los campos o bien porque es la acción a tomar según una regla específica. En estos casos, el proceso de reenvío se simplifica y se reducen costes y tiempos de transmisión. Otras acciones que pueden ser llevadas a cabo es la actualización de contadores o pasar el paquete a otra tabla.

3.3 Componentes de una red SDN

Los componentes que forman una red definida por software son todos aquellos dispositivos, tanto físicos como virtuales que componen la red y permiten implementar una arquitectura SDN. Cada componente tiene un papel esencial a desempeñar dentro de la red.

Los dispositivos SDN están formados por tres elementos: una API que permite la comunicación, una capa de abstracción y una función para procesar los paquetes. Una red SDN está compuesta principalmente por un controlador SDN, una API hacia el sur (southbound API), una API hacia el norte (northbound API), las aplicaciones SDN y los switches.

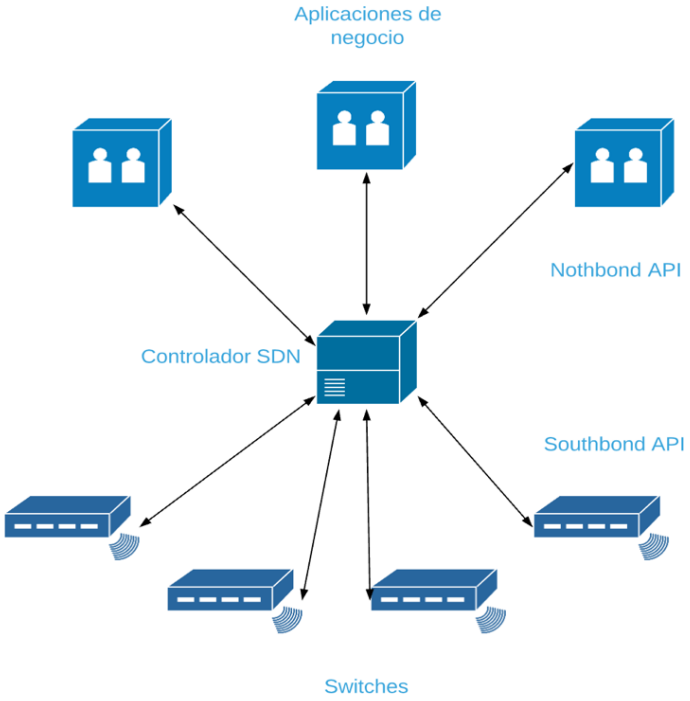


Figura 6: Componentes de una red SDN

Fuente: Elaboración propia

3.3.1 Controlador

Como ya hemos visto, el controlador o *controller* es el elemento más importante en una red SDN, ya que es su "cerebro". Sus tareas son retransmitir datos e información a los switches y routers de la red (capa de infraestructura) y a las aplicaciones de negocio.

Un controlador es un sistema de software basado en un conjunto de módulos que se pueden conectar y desconectar fácilmente. Los controladores te permiten obtener:

- Un manejo centralizado y eficiente de la red. Lo que incluye guardar información del estado de la red, información temporal e información sobre la topología establecida en ese momento.
- Un protocolo libre para poder gestionar el estado de la red. Destaca OpenFlow
- Una sesión de control segura (TCP) que establece el controlador con los agentes de los componentes de la red.
- Un software que permite descubrir dispositivos, la topología de la red, mecanismos de reenvío y servicios de información.
- Un conjunto de APIs que permiten interactuar entre los servicios del controlador y las aplicaciones de negocio.
- Entornos de desarrollo completos que permiten la expansión dinámica del controlador y la posterior publicación de las APIs.

Las tareas fundamentales de un controlador son el descubrimiento y detección de los dispositivos finales de la red como enrutadores o conmutadores, el control de la topología de la red y sus enlaces y la gestión del flujo. Para realizar estas funciones se necesita una serie de componentes funcionales que son: base de datos local que contenga información actual y estadísticas de la red, una función de control del plano de datos, un coordinador que permite configurar los entornos de cliente y servidor, un virtualizador que permite asignar recursos abstractos a clientes o aplicaciones particulares (no confundir con el término de NFV que consiste en virtualizar las funciones de red del hardware dedicado), un agente que es la entidad funcional que representa los recursos y capacidades del cliente en el entorno del servidor y, por último, interfaces para facilitar la integración con software de terceros.

El controlador también tiene una caché de flujo que muestra las *flow tables* de sus dispositivos, así como estadísticas de flujo de los switches o routers, pudiendo adaptar sus características a las necesidades de la red.

El primer controlador SDN creado fue llamado NOX y era de código cerrado. En 2008 el código fuente de NOX se liberó y se le permitió a la comunidad experimentar libremente.

Así, comenzaron a surgir diferentes controladores, tanto de carácter público (Beacon o POX) como privado (Google WAN Controller).

En 2011 surgió a raíz del controlador Beacon, Floodlight, uno de los controladores públicos más utilizados hasta el momento basado en la tecnología OpenFlow.

Más tarde, grandes empresas tecnológicas como HP, Cisco o IBM empezaron a implementar diseños propios y migraron de los controladores Beacon al controlador de Linux OpenDaylight.

Cada controlador tiene sus propias características. Los controladores basados en OpenFlow son uno de los tantos controladores que existen. Estos en particular destacan porque utilizan como API southbound el protocolo OpenFlow que permiten centralizar la

red y supervisar los componentes de ésta, aumentando la flexibilidad y eliminando los protocolos cerrados e independientes de cada proveedor.

Algunos de los controladores de código abierto destacados son:

- OpenDaylight: Es un controlador creado por la compañía Linux en 2013 muy utilizado gracias a el apoyo de grandes compañías como Cisco. Está escrito en Java y dispone de una API Rest y de un entorno gráfico, además de muchas otras funciones. Destaca porque soporta OpenFlow como API southbound, así como otros protocolos diferentes.
- Beacon: Fue uno de los controladores principales en los inicios de SDN y sirvió de referencia para otros como OpenDaylight o Floodlight. Utilizaba el lenguaje Java y destacó por ser multiplataforma y por utilizar una estructura OSGI que permitía configurar las aplicaciones sin la necesidad de desconectar los switches.
- Floodlight: Este controlador es el sucesor de Beacon pero utiliza Apache Ant. Es uno de los controladores más utilizados. Destaca porque integra también una API Rest y dos entornos gráficos, uno web y otro Java.
- NOX: Fue el controlador original del protocolo OpenFlow. Soporta aplicaciones escritas en C++ en sistemas Linux, sin embargo, actualmente ya no se utiliza.
- POX: Desarrollado a partir del controlador NOX, pero es soportado además de por sistemas Linux, por Windows y por Macintosh. Está escrito en Python.
- Open vSwitch: Es un tipo de controlador libre muy potente que puede trabajar tanto con OpenFlow como con su protocolo propio (OVSDB). Destaca porque es compatible con los kernel de Linux. Su lenguaje de programación es C.
- Cherry: Es un controlador OpenFlow escrito en Go que soporta las versiones 1.0 y 1.3 de OpenFlow.
- Ryu: El controlador Ryu (flujo en japonés) es un software basado en redes SDN con una API muy bien definida que permite a los desarrolladores crear y manejar redes y controlar aplicaciones. Ryu soporta varios protocolos de gestión de dispositivos como OpenFlow, Netconf o OF-config. Respecto a OpenFlow, soporta desde la versión 1.0 hasta la 1.5 y más extensiones. Ryu es libre y utiliza Apache. El lenguaje de programación utilizado en Python.

Pese a estos ejemplos, existen muchos más controladores de software libre. En la siguiente tabla podemos ver de modo esquemático las características de algunos de estos controladores gratuitos:

	Beacon	Floodlight	NOX	POX	Trema	Ryu	ODL
Soporte de OpenFlow	v1.0	v1.0	v1.0	v1.0	v1.3	v1.0-v1.5	
Virtualización	Mininet y Open vSwitch	Mininet y Open vSwitch	Mininet y Open vSwitch	Mininet y Open vSwitch	Herramienta virtual de simulación	Mininet y Open vSwitch	Mininet y Open vSwitch
Lenguaje utilizado	Java	Java	C++	Pyhton	Rudy/C	Python	Java

REST API	No	Sí	No	No	Sí	Sí	Sí
Interfaz Gráfica	Web	Web	Python+	Python +, Web	No	Web	Web
Plataformas soportadas	Linux, Macintosh, Windows, Android	Linux, Macintosh, Windows	Linux	Linux, Windows, Macintosh	Linux	Linux	Linux, Windows, Macintosh
Multiproceso	Sí	Sí	Sí	No	Sí	No	Sí
Código abierto	Sí	Sí	Sí	Sí	Sí	Sí	Sí
Documentación	Alta	Alta	Media	Baja	Media	Alta	Media

Tabla 3: Características de algunos controladores de código abierto

También podemos encontrar controladores comerciales de fabricantes como:

- Big Switch: Dispone del controlador Big Cloud Fabric que se ejecuta en el sistema operativo Switch Light y que permite una fácil transición de las redes actuales a redes SDN.
- Cisco: Destaca XNC, que es la versión comercial del controlador OpenDaylight, y se ejecuta en la Cisco One Platform Kit (onePK).
- HP: Esta compañía ha creado el controlador HP VAN SDN Controller que proporciona un kit de desarrollo de software con el fin de crear aplicaciones SDN en el controlador.

Ryu es uno de los controladores libres mejor documentados que hay hasta la fecha además de ser muy potente. Más adelante explicaremos el proceso de instalación de este controlador ya que es el controlador elegido para esta tesis.

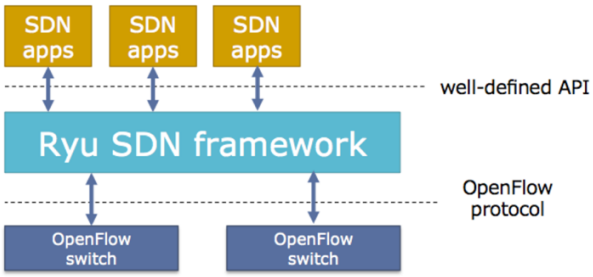


Figura 7: Funcionamiento del controlador Ryu

Fuente: Network Startup Resource Center. University of Oregon

Sin embargo, como hemos visto hay multitud de controladores abiertos disponibles. Para elegir un controlador hay que tener en cuenta algunos puntos como saber qué versiones de OpenFlow soporta, ver si tiene soporte para virtualizar la red, que tenga capacidad para realizar distintas funciones como enrutamiento en múltiples caminos o división de tráfico,

conocer el número de dispositivos que puede administrar simultáneamente, que ofrezca un alto nivel de procesado de las tablas de flujo, que aporte fiabilidad y seguridad a la red o que soporte a diversos fabricantes de controladores.

Hay que añadir que, pese a que la arquitectura SDN permite tener centralizado el plano de control, hoy en día el concepto de SDN no obliga a tenerlo centralizado.

3.3.2 API ascendentes

Las API ascendentes, Northbound API o API norte permiten, como ya hemos comentado, comunicar al controlador con los servicios y la lógica de negocio que hay por encima (capa de aplicaciones).

Estas API tienen diversas funciones, como por ejemplo facilitar la innovación o permitir la gestión y la automatización de la red para cumplir con los requisitos que tengan las aplicaciones. Estas interfaces son el punto más crítico de la red ya que deben soportar muchas aplicaciones y servicios. Hay un gran catálogo de interfaces situadas en diferentes lugares que permiten el control de diferentes tipos de aplicaciones a través del *controller*. En este ámbito destacan las REST (REpresentational State Transfer) APIs que utilizan en protocolo de internet HTTP y que permiten una amplia variedad de acciones gracias a que se programan en formatos como JSON o XML. Actualmente no hay una API ascendente basada en estándares.

3.3.3 API descendentes

Las API descendentes también son conocidas como southbound API o API sur. Estas API facilitan el control de la red, permitiendo al controlador gestionar los dispositivos y realizar cambios dinámicos según los requisitos y necesidades de la red. Es decir, comunica al controlador con el plano de datos. Estas API permiten también la programabilidad de las operaciones de reenvío, obtención de información de los dispositivos, datos estadísticos y notificación de eventos de la red.

En esta interfaz destaca el protocolo desarrollado por la Open Networking Foundation (ONF) conocido como OpenFlow, del que hablaremos más adelante. OpenFlow es el protocolo más utilizado para la elaboración de redes SDN, aunque existen otras variantes como Cisco OpFlex.

Hay un modelo de arquitectura SDN que, en vez de utilizar la figura del controlador, se compone de APIs que proporcionan los dispositivos de red distribuidos. Una ventaja es que permite mejorar los procesos de agilización y automatización de la red, además de que la sencillez de escribir software de herramientas de orquestación que responde rápidamente a los cambios de la red. Sin embargo, esta arquitectura SDN tiene bastantes limitaciones ya que, al eliminar la figura del controlador, el ingeniero de red debe configurar individualmente cada dispositivo y sincronizar el plano de control interno de cada elemento con el plano de control distribuido, lo que provoca la eliminación de los switches y routers más simples y menos costosos que se habían conseguido con la arquitectura SDN, volviendo a los dispositivos más costosos y complicados de utilizar. Por último, también se pierde el control del flujo que se conseguía con el enfoque con controladores.

3.3.4 Aplicaciones SDN

Las aplicaciones SDN son los programas que comunican las necesidades de la red para un funcionamiento óptimo y el comportamiento esperado de ésta. Estas necesidades son comunicadas al controlador mediante las anteriormente comentadas northbound APIs. Además, las aplicaciones son responsables también del control de las entradas programadas en los dispositivos, y pueden tomar sus propias decisiones sin la actuación de un controlador. Mediante las API, tienen la capacidad de gestionar los flujos de reenvío de paquetes, equilibrar cargas de tráfico para mejorar la calidad de servicio, modificar su carácter según la topología que hay o reencaminar los paquetes ante problemas de seguridad.

Una aplicación SDN puede invocar otros servicios externos y puede organizar cualquier número de controladores SDN para lograr sus objetivos. Estas aplicaciones requieren al menos una cierta cantidad de conocimiento a priori de sus entornos y roles. Una entidad del plano de aplicación puede actuar como:

- Un servidor de información, en cuyo caso expone un modelo de información para su uso por otras aplicaciones que, normalmente, son clientes que se comunican con el agente del servidor de aplicaciones SDN.
- Un cliente de modelo de información, en cuyo caso opera en una instancia de modelo de información expuesta por una entidad de servidor. La entidad del servidor suele ser un controlador SDN o una aplicación subordinada.
- Ambos roles simultáneamente

Que las aplicaciones puedan también desempeñar tareas de control aparte del controlador es otra ventaja de la arquitectura SDN, ya que la hace potencialmente escalable.

Las aplicaciones SDN se pueden clasificar según la función que desempeñan en la red, la plataforma de software que utilizan y la complejidad de estas:

- Algunas funciones específicas que pueden desarrollar son tareas de seguridad, de calidad de servicio, de ingeniería de tráfico, balanceo de carga etc. Otras aplicaciones son creadas para prestar servicios como computación en la nube, en redes móviles, en virtualizar la red o en NFV.
- Según su plataforma software, las aplicaciones SDN pueden ser internas o externas. Las internas son creadas como extensión de las funcionalidades del controlador SDN y su código se ejecuta en la plataforma software del controlador. Las aplicaciones externas pueden ejecutarse en una plataforma diferente al controlador y se comunican con éste mediante las API norte.
- Según su complejidad pueden ser simples o complejas. Las aplicaciones simples son aquellas cuyo código cumple sólo una función de la red mientras que las complejas poseen código más completo que permite implementar eficiente y simultáneamente varias funciones de red.

Para el desarrollo de estas aplicaciones existen tres grupos de herramientas. El primero son las herramientas por defecto que tienen los controladores sobre su propia plataforma como un lenguaje de programación, Kits SDK (Software Development Kit) o bibliotecas de Interfaces de Programación de Aplicaciones (API). Otro grupo son las plataformas de desarrollo de software tradicionales que permiten implementar estas aplicaciones con la capacidad de exponer sus servicios a través de la API ascendente. Y, por último, frameworks de programación para SDN que constan de su propio controlador, y son conocidos como DSL

3.4 OpenFlow

El protocolo de routing OpenFlow es un proyecto de código abierto resultado de seis años de colaboración en investigación entre la Universidad de Stanford y la Universidad de California. OpenFlow ofrece la posibilidad de controlar de una forma sencilla el modo en el que trabajan las redes. Para ello permite a un servidor de software determinar la ruta que seguirán los paquetes en una red de conmutadores. Con este protocolo, una red puede ser gestionada globalmente, sin tener que gestionar uno a uno los dispositivos hardware que forman la red. El propio servidor software es el que envía las instrucciones de reenvío a los conmutadores para que encaminen los paquetes.

OpenFlow permite separar en los equipos de enrutamiento el Datapath del Controlpath, es decir, permite dividir el reenvío de paquetes del encaminamiento de alto nivel. La comunicación de estos dos elementos se produce mediante este protocolo, consiguiendo dar un enfoque virtual a la red.

Otra ventaja es que permite a los usuarios crear políticas para encontrar rutas con ancho de banda disponible, baja latencia o congestión y menos saltos entre switches y routers de diferentes fabricantes. Es decir, permite crear networking programable independientemente del fabricante.

Comúnmente se suele confundir erróneamente SDN con OpenFlow. Sin embargo, OpenFlow es un elemento dentro de la arquitectura SDN, pero la inmensa popularidad que está obteniendo hace que se esté convirtiendo en el modelo estándar de implementación de redes SDN.

El protocolo OpenFlow se compone principalmente de tres elementos: Las tablas de flujo, un canal seguro y el controlador.

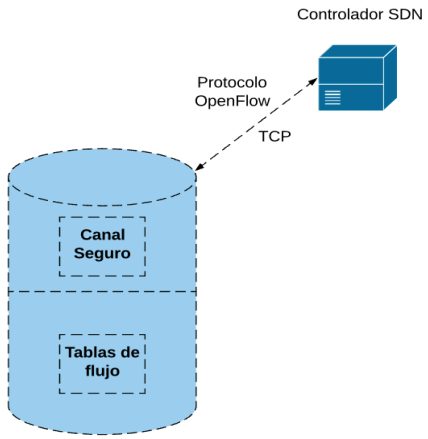


Figura 8: Arquitectura de OpenFlow

Fuente: Elaboración propia

La primera especificación de OpenFlow fue de diciembre de 2009 aunque con el paso de los años se han ido añadiendo nuevas versiones con el objetivo de mejorar el protocolo. Desde abril de 2015 OpenFlow se encuentra en la versión 1.5.1. Las principales mejoras que se han ido añadiendo al protocolo son:

- Versión 1.1: Esta versión fue lanzada en febrero de 2011 e incluía las tablas de grupos de los switches y la posibilidad de utilizar varias tablas de flujo en un mismo conmutador.
- Versión 1.2: En diciembre de 2011 se añade el soporte de IP versión 6. Además, se le permite a un switch conectarse con diversos controladores pudiendo añadir privilegios a éstos. Además, aparece el packet matching que permite comparar campos.
- Versión 1.3: En junio de 2012 se lanza esta versión con la aparición de la tabla de medidas que permite definir un conjunto de métricas que permiten implementar sobre el flujo de datos operaciones de control o calidad.
- Versión 1.4: Aparece en octubre de 2013 esta versión dónde se incluye el bundles mechanism que permite agrupar mensajes OpenFlow a la hora de configurar los switches. Se agrega también la opción de sincronizar las tablas de flujo.
- Versión 1.5: La versión más reciente del protocolo aparece en diciembre 2014 dónde destaca la posibilidad de basar las tablas de flujo en los puertos de salida del conmutador.

3.4.1 Las tablas de flujo

Los routers y switches Ethernet incluyen tablas de flujo compuestas a partir de memorias TCAM y RAM utilizadas para control de la *Quality of Service* (QoS) u obtención de estadísticas. Las tablas de flujo varían según el fabricante, pero tienen funciones comunes que los conmutadores y enrutadores SDN utilizan. Los conmutadores que utilizan OpenFlow están basados en estas tablas de flujo de los dispositivos Ethernet y cuentan con una tabla de flujo como la siguiente:

Match Fields	Priority	Counters	Instructions	Timeouts	Cookies	Flags

Tabla 4: Tabla de flujo

- Match Fields o campos de correspondencia: Definen flujo mediante el establecimiento de un conjunto de campos a comparar con los paquetes que se reciben.
- Prioridad: En el caso de que se verifiquen varias entradas de una tabla se elige la de mayor prioridad.
- Contadores: Se actualizan cuando se selecciona la entrada.
- Instrucciones: Las acciones a llevar a cabo cuando se selecciona la entrada
- Timeout: Máximo tiempo que la entrada puede estar inactiva.
- Cookie: El controlador puede guardar un valor.
- Flags: Tiene diferentes opciones de utilización.

Cuando un paquete entra en por un puerto del switch, los campos de cabecera de los paquetes son extraídos y se comparan los campos de las cabeceras con los campos de correspondencia (match fields) de las entradas de las tablas de flujo, teniendo en cuenta

la prioridad. Los campos de coincidencia pueden ser puerto de entrada o salida, direcciones MAC origen y destino, etiquetas MPLS o VLAN, direcciones IP, puertos, flags etc.

Cuando un campo de la cabecera del paquete coincide con un campo de correspondencia de la tabla de flujo, se realiza una acción, que puede ser:

- Enviar el paquete a su destino a través de la red: Si se conoce el destino y no se requiere ninguna acción adicional se trabaja de forma normal.
- Enviar el paquete al controlador: En caso de que se necesite procesar el paquete por el controlador para que éste tome medidas.
- Modificar campos del paquete: Añadir, eliminar o editar campos de las cabeceras del paquete.
- Descartar el paquete: Esta acción permite aumentar la seguridad de la red y utilizar el switch como un firewall.

Los conmutadores OpenFlow cuentan además con la posibilidad de que haya más de una tabla de flujo, es lo que se denomina Flow Pipeline (tubería de flujos) y entonces el paquete puede ser procesado por más de una tabla. Las tablas se comprueban en orden hasta que se verifique una regla. Si no se verifica ninguna regla en una tabla se llama "table miss" y la acción por defecto llevada a cabo puede ser pasar el paquete a la tabla posterior, reenviar el paquete por un puerto en concreto o reenviar los 200 bytes iniciales del paquete al controlador para que éste lo procese.

Para mejor comprensión sobre el procesado de paquetes en conmutadores OpenFlow, podríamos observar el siguiente diagrama de flujo:

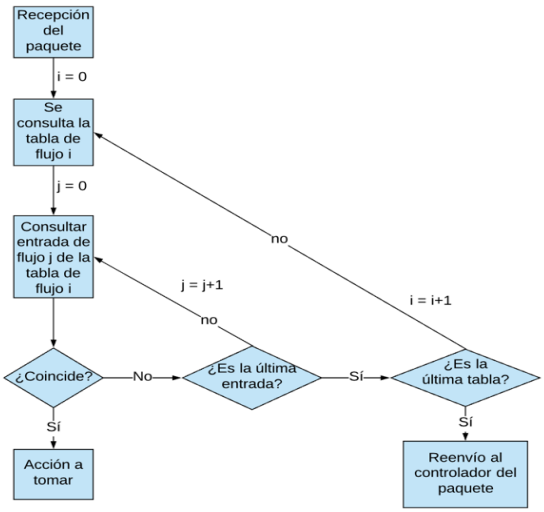


Figura 9: Diagrama de flujo del procesado de un paquete OpenFlow

Fuente: Propia

3.4.2 El canal seguro

El canal seguro se encuentra dentro del switch OpenFlow y permite conectar el switch al controlador. Se utiliza para que el controlador pueda enviar al switch comandos o

instrucciones sobre cómo encaminar un paquete o bien el envío de paquetes cuando el switch no sabe como procesarlos.

La comunicación switch-controlador se produce a nivel de transporte y se utiliza el protocolo TCP, aunque en ocasiones se puede utilizar TLS. El paquete OpenFlow tiene la misma estructura en la cabecera sea cual sea su versión. Los campos que la componen son:

- Versión: Son 8 bits utilizados para indicar la versión OpenFlow. Hay 5 posibles valores, siendo el primero para la versión 1.0 y el último para la versión que hay ahora mismo, la 1.4.
- Type: 8 bits que indican el tipo de mensaje que hay en el Payload. Actualmente hay 35 tipos distintos de mensajes por lo que los valores están comprendidos entre el 0 y el 34.
- Length: Estos 16 bits indican la longitud del mensaje OpenFlow, siendo el tamaño mínimo de 8 bits que es el valor de la cabecera.
- Xid: 32 bits que identifican la transacción que se está realizando en ese momento.

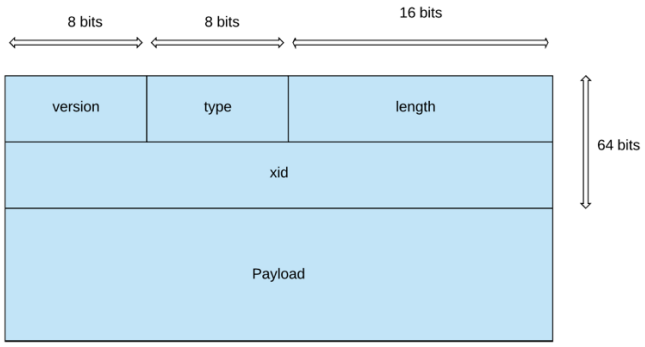


Figura 10: Estructura de un paquete OpenFlow

Fuente: Elaboración propia

Hay tres tipos de mensajes OpenFlow:

- Controlador a switch: Permiten modificar, editar o borrar definiciones de flujos, pedir información sobre el switch SDN o enviar paquetes del controlador al switch una vez se ha creado una nueva norma de flujo.
- Switch a controlador o mensajes asíncronos: Se utilizan para enviar el paquete que hay en el switch al controlador y para informar al controlador del descarte de un flujo, de un cambio de estado en el switch o de algún error.
- Mensajes simétricos: Se envían tanto por el switch como por el controlador. Algunos mensajes que podemos encontrar son el del inicio de conexión (HELLO), mensajes experimentales de extensiones de OpenFlow o bien mensajes para determinar la latencia de la conexión.

3.4.3 El controlador

El controlador es la aplicación software encargada de añadir y eliminar las entradas en la tabla de flujos de los dispositivos de red y que permiten gestionar correctamente el tráfico de la red.

La mayoría de los controladores SDN tienen como API sur predeterminada a OpenFlow debido a la gran popularidad que ha recibido, y que les permite controlar multitud de dispositivos de diferentes fabricantes. Aun que los controladores también pueden disponer de otras interfaces southbound adicionales.

El controlador tiene siempre la comunicación activa con los switches OpenFlow de la red a la espera de recibir mensajes. Actualmente el protocolo OpenFlow utiliza el puerto TCP número 6653 aunque anteriormente ha utilizado otros números de puertos.

Capítulo 4. Comparación con las redes tradicionales

La comparación entre las redes SDN y las redes tradicionales muestra la clara mejora de la arquitectura definida por software y es una de las causas por las que está apareciendo esta tendencia hacia las redes SDN. A lo largo de los anteriores puntos hemos podido comprobar algunas de las diferencias que podíamos encontrar en las redes definidas por software respecto a las redes tradicionales y las grandes ventajas que las primeras tienen respecto a las segundas.

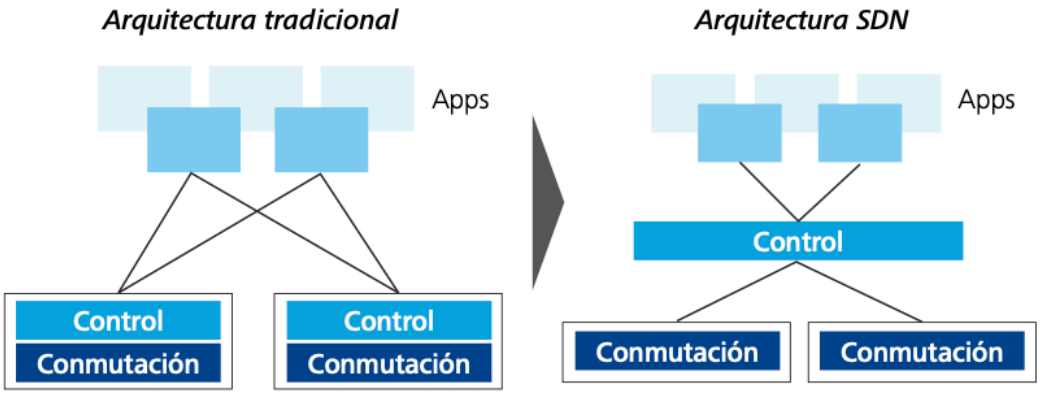


Figura 11: Comparativa entre la arquitectura tradicional y la SDN

Fuente: Deloitte

A continuación, se muestran diversos puntos a comparar:

- Control: El primer punto que comparar es respecto al control de la red en sí. Las redes tradicionales están descentralizadas. Sin embargo, la complejidad de las redes de hoy en día y la cantidad de datos que se transportan a diario hacen difícil llevar a cabo funciones de seguridad, calidad de servicio o gestión, por lo que en SDN el control se ha centralizado con la figura del controlador. Pese a que en las redes tradicionales es posible llevar a cabo una automatización, esta se hace muy tediosa mientras que en SDN es suficiente una sola interfaz para todos los dispositivos. Como desventaja de la centralización, se encuentran los ataques de denegación de servicio al controlador o congestión de éste.
- Aplicaciones: Las aplicaciones que componen la red tradicional están basadas en un modelo cliente servidor. Sin embargo, en la actualidad las aplicaciones están distribuidas a lo largo de diversos servidores y máquinas virtuales, intercambiándose tráfico entre ellas de forma que se puede balancear la carga u optimizar los servicios.
- Rigidez: Las redes tradicionales son estáticas y es difícil aplicar políticas, sin embargo, la arquitectura SDN permite obtener una red dinámica. La virtualización de la red hace que la red sea muy flexible, a diferencia de la red tradicional.
- Ampliación de la red: La red tradicional tiene unos límites de ampliación que SDN no tiene ya que soporta mucho mejor la escalabilidad.

- Configuración: En un conmutador tradicional el administrador de la red debe configurar el dispositivo manualmente mediante la línea de comandos mientras que con SDN se puede programar el conmutador más fácilmente ya que el controlador SDN ofrece una interfaz de programación para configurar los conmutadores virtualmente.
- Tablas de flujo: La red tradicional tiene las tablas de flujo cerradas en los dispositivos mientras que en SDN las tablas de enrutamiento son abiertas.
- Investigación: Cada vez es más corta la investigación y desarrollo de las redes tradicionales. En las redes SDN la innovación es muy amplia y es que SDN todavía está en fase de implantación.
- Coste: La red tradicional tiene alto coste operacional y operativo. Las redes definidas por software tienen una elevada reducción de costos operativos y financieros.
- Fiabilidad: Los protocolos aplicados a SDN como OpenFlow son todavía muy jóvenes y se encuentran en proceso de pruebas y mejoras, aunque con el paso del tiempo tendrán más fiabilidad y serán más sólidos. Además, al ser tan innovador, muchos fabricantes todavía no los soportan. Sin embargo, los protocolos básicos de las redes tradicionales son utilizados en todo el mundo demostrando ser seguros y fiables en condiciones generales.
- Soporte: Los switches SDN sólo funcionan para la arquitectura SDN mientras que un switch normal trabaja independientemente del resto de la red.
- Análisis: En SDN todas las cabeceras de los paquetes deben ser analizadas para enrutarlos y poder aplicar la regla de flujo correspondiente.
- Tasa de envío: Por lo general, la tasa de transferencia de paquetes en las redes SDN es mayor a la de las redes tradicionales. Esto es debido a que los switches SDN consultan las tablas de flujo antes de transferir el tráfico. En caso de no saber a qué dirección enviar, preguntan al controlador y actúan según su respuesta. Mientras que, en las redes tradicionales, si los conmutadores no saben cómo tratar el tráfico, se envía solicitud de broadcast ARP a todos los nodos, lo que genera un retardo inicial en el envío de tramas.
- Ámbito de uso: Según el tamaño de la red, es más eficiente usar SDN u otra arquitectura, ya que, por ejemplo, en un mismo segmento de red, arquitecturas tradicionales como Ethernet permiten conseguir transmisiones más rápidas ya que no existe la consulta al controlador, como sí ocurre en SDN.
- Ancho de banda: El ancho de banda sufre cambios más drásticos en redes tradicionales como Ethernet mientras que SDN estabiliza el tiempo de envío de los paquetes de datos y es más eficiente en cuanto a ancho de banda.

A modo de resumen, podemos ver en la siguiente tabla las diferencias básicas entre SDN y el networking tradicional:

Redes definidas por software	Redes tradicionales
Instancia de control centralizada	Instancias de control específicas de los dispositivos

Separación del plano de control y el hardware	El control está integrado en el hardware
Se puede programar el plano de control	El plano de control depende del fabricante
Protocolos de código abierto	Protocolos específicos de los fabricantes
Acceso por software al plano de datos	Acceso al plano de datos en el hardware
Arquitectura flexible y escalable	Arquitectura estática y difícil de editar

Tabla 5: Diferencias básicas entre las redes SDN y las redes tradicionales

Capítulo 5. Ubicación en el mercado actual

Como hemos podido observar de manera teórica, la arquitectura SDN tiene muchísimas ventajas y beneficios. Es una tecnología innovadora que dará un salto cualitativo en las arquitecturas de red. La industria SDN ha pasado de 406 millones de dólares en 2013 a un mercado que el año que viene tendrá más de 13.800 millones de dólares, y según una encuesta de Network World de 2017, un 49% de profesionales de redes estaba considerando una implementación de SDN.

La Open Networking Foundation cuenta ya con más de 200 empresas colaboradoras.

Sin embargo, desde el punto de vista práctico, es todavía una arquitectura en evolución y todavía está en fase de desarrollo. Además, la gran mayoría de los fabricantes todavía no están implementando esta arquitectura debido a que no están abiertos a innovaciones que puedan alterar su mercado y estabilidad.

Pese a estos datos, en los últimos años, la comunidad SDN está dando el paso hacia las SDN y grandes empresas ya han adoptado SDN en algunos casos de uso como, por ejemplo:

- Ncira (comprada por VMware en 2013) está profundizando en la idea de llevar el firewall a la máquina virtual, en escenarios dónde la seguridad es vital. Hoy en día es un proveedor de estas redes, donde destaca su producto NSX SDN, basado en SDN.
- La empresa Nuage, perteneciente al grupo Alcatel, está llevando las SDN a la WAN con el objetivo de ayudar a los operadores de telecomunicaciones a adaptar sus redes privadas de clientes a un entorno dinámico de provisión de servicios. Esto se conoce como SD-WAN (Wide-Wide-Networking), y permite utilizar una plataforma de gestión de software para controlar el acceso a las oficinas remotas o sucursales de una empresa. De manera que una compañía puede tener diversos tipos de conexiones de red a sus sucursales con una alta disponibilidad y priorización de tráfico.
- La compañía HP desde octubre de 2014 ha apostado con su App Store por el uso de SDN en entornos de campus. Esta App Store es una plataforma combinada con servicios de consultoría y apoyo que permite a sus clientes descubrir la tecnología de las redes SDN. Además, permite adaptar de forma dinámica la QoS de sus aplicaciones.
- La empresa BT lanzó en 2017 una aplicación basada en la tecnología SDN llamada BWoD (Bandwidth on demand) que es un método de comunicación que ofrece la capacidad de acomodar las demandas de tráfico.
- Intel está adoptando las redes definidas por software para habilitar el aprovisionamiento a pedido de redes y servicios de red.
- Muchas compañías como Juniper, Arista o Cisco Application Centric Infraestructure están adaptando ofertas para sus SDN.
- IBM ha comenzado a ofrecer servicios algunos servicios enfocados a las redes definidas por software como una consultoría SDN, SD-WAN híbrido, una IMI (Integrated Managed Infraestrcuture) para SDN o SDN para centros de datos.

Todo esto nos lleva a pensar que la arquitectura SDN, pese a que actualmente está poco desplegada, poco a poco irá aumentando su influencia.

Capítulo 6. Seguridad en las redes SDN

Las ventajas de SDN en diversos escenarios y redes de backbone ya han sido probados con éxito. Sin embargo, todavía quedan grandes desafíos para poder implementar una red de operador a gran escala utilizando SDN. Dentro de todos estos desafíos, hay un área clave, la seguridad SDN.

La centralización y programabilidad de una SDN también presenta desafíos de seguridad como los ya nombrados ataques de denegación de servicio (DoS) o el abuso de confianza entre aplicaciones de programación abierta y el controlador. Teniendo en cuenta los problemas específicos de la seguridad SDN, podemos identificar los desafíos asociados a cada capa de la arquitectura SDN: aplicación, control y plano de datos, y las respectivas interfaces.

Por ejemplo, el protocolo OpenFlow describe el uso de la seguridad en la capa de transporte (TLS) con una autenticación mutua entre el controlador o controladores y los conmutadores SDN. Sin embargo, esta función de seguridad es opcional y no está especificada en el estándar TLS. La falta de uso de TLS podría llevar a la inserción de reglas fraudulentas o la modificación de estas con los problemas que pueden derivar.

Otro tipo de ataque frecuente consiste en el escaneo de puertos para descubrir puntos vulnerables de la red. SDN permite una defensa contra estos ataques con el uso de direcciones virtuales IP mediante el uso de un controlador OpenFlow que permite administrar un conjunto de direcciones IP virtuales, que asigna a los hosts dentro de la red, ocultando las direcciones IP reales del mundo exterior. Es lo que se conoce como ciberseguridad adaptativa.

Como podemos ver, la arquitectura SDN se puede aprovechar para mejorar la seguridad de la red mediante la implementación de un sistema de monitorización, análisis y respuesta de seguridad altamente reactivo, gracias al papel que desempeña la figura del controlador. El análisis de tráfico o métodos de detección de anomalías generan datos que pueden transferirse al controlador central y las aplicaciones pueden ser ejecutadas en el controlador para analizarlas. Con base a ese análisis, se pueden aplicar reglas de seguridad mediante la implementación de las tablas de flujo. La combinación de la vista global o la programabilidad de la red permite la implementación de sistemas de detección de intrusiones (IDS) y los sistemas de prevención de intrusiones (IPS).

Otra medida de seguridad es la implementación de sistemas intermedios como la integración de middle-boxes de seguridad que permiten redirigir el tráfico de la red. Así destaca la arquitectura Slick, que propone un controlador centralizado que se encarga de instalar y migrar funciones en cuadros intermedios personalizados. Las aplicaciones pueden dirigir al controlador Slick para que instale las funciones necesarias para enrutar flujos particulares en función de los requisitos de seguridad.

La arquitectura FlowTags propone el uso de sistemas intermedios mínimamente modificados que interactúan con un controlador a través de una interfaz de programación de aplicaciones (API) FlowTags. En los encabezados de los paquetes se integran diagramas de flujo que permite proporcionar un seguimiento del flujo y controlar el tráfico de los paquetes etiquetados. Por contrapartida, esta arquitectura necesita tener las políticas predefinidas, por lo que no es dinámica.

Estas medidas de introducción de elementos intermedios tienen como desventaja que pueden penalizar el rendimiento cuando se desvía el tráfico de la red a través de ellos.

Otro aspecto que ofrece seguridad en las redes SDN es que estas redes son completamente cerradas con tráfico encriptado, por lo que el negocio queda menos expuesto a ataques y brechas de seguridad que las soluciones de red tradicionales. Es por ello que muchas empresas están invirtiendo en SDN con el objetivo de conseguir beneficios cruciales relacionados con la ciberseguridad.

Capítulo 7. Experimento y resultados

En este capítulo se pretende aplicar a la práctica los conocimientos teóricos aprendidos durante este proyecto sobre las redes definidas por software mediante el desarrollo de diferentes entornos virtuales basados en la arquitectura SDN en los que se mostraran las diferentes aplicaciones que ofrece el controlador Ryu.

En concreto, las aplicaciones Ryu a mostrar en esta tesis son:

- Firewall: Se simulará una red SDN dónde el controlador implementará un Firewall en el conmutador y añadirá diferentes reglas ACL para permitir o bloquear el tráfico entre hosts.
- Spanning Tree: Se simulará una red SDN basada en Spanning Tree de modo que el controlador aplicará las técnicas necesarias para un correcto uso de la red.
- Acciones de QoS: Se simulará una red SDN en la que se requerirán diferentes calidades de servicio según el enlace y el controlador mandará las instrucciones oportunas para que éstas se cumplan.
- Monitor de tráfico: Se mostrará un monitor de tráfico sencillo implementado en el controlador de Ryu que permite obtener parámetros estadísticos de la red, muy útiles a la hora de analizarlos con las nuevas tendencias de inteligencia artificial o deep learning.

Para este trabajo se ha utilizado el software libre Mininet, el cual se ha instalado en una Raspberry-Pi. Como controlador se ha decidido utilizar Ryu, que, como se ha comentado anteriormente, es un controlador SDN de código abierto basado en OpenFlow.

Como se describe en el resumen del proyecto, con las pruebas pertinentes sobre la red virtual SDN se quieren mostrar las posibilidades que ofrece la utilización del controlador Ryu en particular en cuanto a la gestión del tráfico y los dispositivos de la red.

Además, se utilizarán otras herramientas más conocidas como comandos en lenguaje Linux que nos permitirán analizar el tráfico y ver sus características.

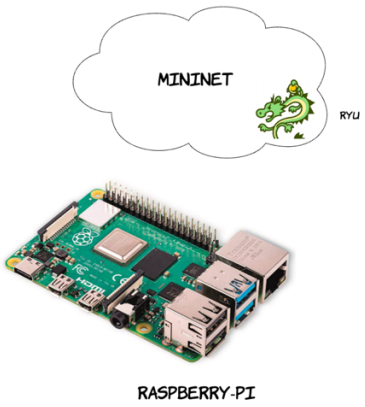


Figura 12: Estructura del experimento

Fuente: Propia

7.1 Herramientas utilizadas

A continuación, se van a detallar las tres herramientas básicas para realizar este proyecto: Una Raspberry-Pi dónde se ubicará nuestro proyecto, el software utilizado para la creación del entorno virtual que es Mininet, y su interfaz gráfica Miniedit, y un controlador SDN Ryu.

7.1.1 Raspberry-Pi

Pese a que este experimento se puede realizar en una máquina virtual con un sistema operativo con Linux, como por ejemplo Ubuntu, instalado en un ordenador personal, se ha decidido trabajar sobre un dispositivo que está adquiriendo mucha fama en los últimos años por todas las posibilidades que ofrece, y no es otro que la placa conmutadora Raspberry-Pi.

La Raspberry-Pi es un ordenador de placa reducida (SBC) de bajo coste que fue creado en el Reino Unido en 2012 con el objetivo de fomentar la docencia en informática en los colegios y universidades.

Pese a que el hardware es un producto de propiedad registrada, su software es libre, siendo el sistema operativo oficial Raspbian, una versión adaptada a la Raspberry de Debian. Aunque se pueden utilizar otros sistemas operativos como Ubuntu, Windows 10 o Fedora.

No se incluyen elementos periféricos como el teclado, el ratón, la carcasa o una pantalla, aunque se pueden adquirir algunos accesorios oficiales y no oficiales como complementos para la Raspberry.

Desde el primer modelo que salió a la venta (Raspberry-Pi 1 Model A) en 2012 hasta el último que fue lanzado en junio de 2019 (Raspberry-Pi 4 model B) ha habido otros cinco modelos. En todas sus versiones podemos encontrar un procesador Broadcom, memoria RAM, GPU, puertos USB, HDMI, Ethernet, 40 pines GPIO y un conector para cámara. No se incluye memoria, pero se puede utilizar una tarjeta SD en su primera versión y una MicroSD en las siguientes.

Más tarde aparece la posibilidad de la conectividad inalámbrica mediante Wi-Fi o Bluetooth.

Figura 13: Logo de la marca Raspberry-Pi

Fuente: raspberrypi.org

Para este proyecto se ha utilizado una Raspberry-Pi 4 Model B de 1Gb de RAM. Algunas de las novedades que incluye este modelo respecto a los anteriores son la aparición de dos puertos microHDMI que sustituyen a los conectores HDMI de tamaño completo, lo

que le permite manejar dos pantallas a la vez con una resolución de 4k a 60 Hz. Como conector de alimentación aparece el USB Tipo C, que sustituye al Micro USB y permite suministrar 5V y 3A para una potencia total de 15 W.

Destaca también la utilización de USBs 3.0, así como la desaparición de la limitación del puerto Ethernet a 300 Mbps por lo que permite la utilización del Gigabit Ethernet. El procesador Broadcom nuevo es tres veces más eficiente que el anterior, lo que le permite trabajar a 1,5 Ghz. Por último, la Raspberry Pi 4 ofrece, al igual que su predecesora, la posibilidad de conectarse inalámbricamente mediante algunas tecnologías como Wi-Fi 2,4/5Ghz, Bluetooth 5.0 y BLE.

Su precio se encuentra entorno a los 40 y los 60€, según si queremos la opción de 1, 2 o 4 Gb de RAM.

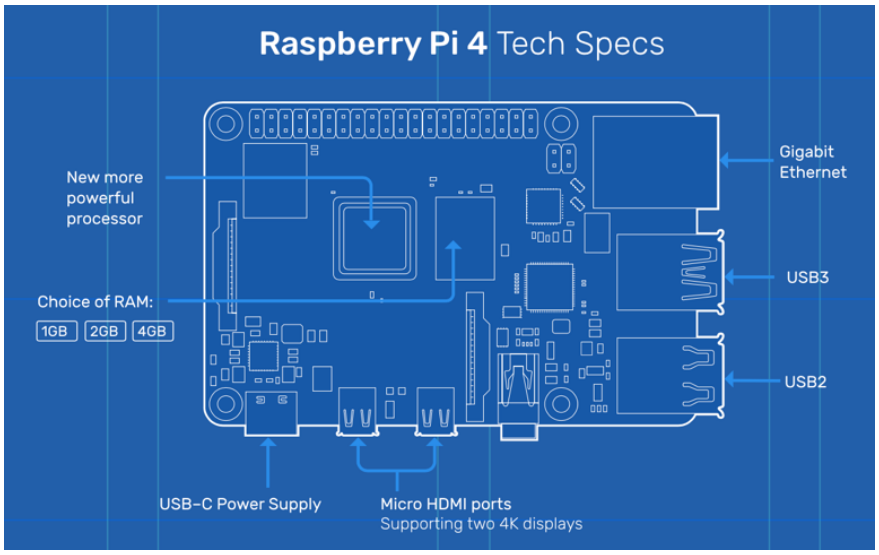


Figura 14: Especificaciones tecnológicas de la Raspberry-Pi 4

Fuente: raspberrypi.org

Para este proyecto se ha utilizado un teclado y ratón externos. Así como una televisión para poder proyectar en pantalla la Raspberry junto con un cable HDMI estándar y un adaptador de HDMI a microHDMI. Como memoria hemos utilizado una tarjeta microSD SanDisk de 16Gb a la cual hemos insertado la imagen del sistema operativo Raspbian.

Primero de todo hemos descargado de la página oficial de Raspberry (raspberrypi.org) la imagen Raspbian Buster, seleccionando la versión completa que incluía el escritorio y software recomendado.

Para poder grabar la imagen en la tarjeta microSD hemos utilizado la herramienta balenaEtcher:

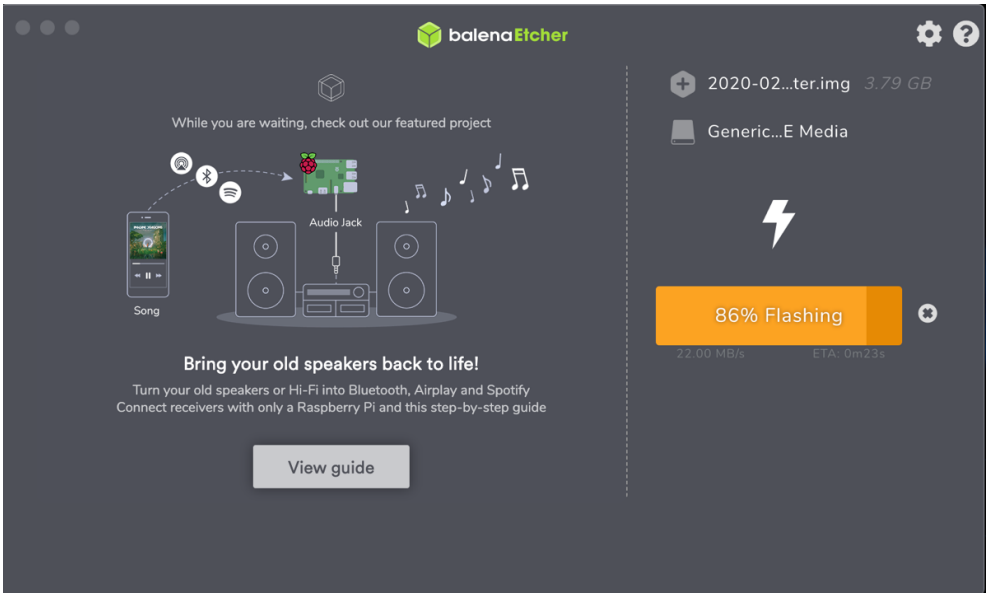


Figura 15: Proceso de grabación del sistema operativo en la microSD

Fuente: Elaboración propia

Una vez se ha descargado la imagen en la tarjeta microSD, conectamos la tarjeta en la placa Raspberry. A continuación, conectamos la alimentación mediante el USB Tipo C, un ratón y un teclado mediante USB para poder controlar la Raspberry y conectamos la TV con un cable HDMI y un adaptador para que pueda conectarse al puerto microHDMI.

Automáticamente, la Raspberry empieza la configuración del sistema operativo Raspbian, simplemente tendremos que seleccionar el idioma en el que deseamos tener el sistema y conectarnos a internet. En este caso no hemos utilizado el cable Ethernet ya que la Raspberry-Pi 4 cuenta con Wi-Fi. Una vez establece la conexión con la red doméstica inicia el proceso de actualización del sistema operativo. Cuando la actualización finaliza se reinicia la Raspberry.

Como último paso, se recomienda cambiar la contraseña predeterminada que nos viene en Raspbian y que es "Raspberry". Esta contraseña será la utilizada cuando queramos realizar comandos en Linux como administrador (comando "sudo").

El proceso total no dura más de 15 minutos y, una vez completado, ya se puede disfrutar del funcionamiento de la Raspberry. Si el sistema no se hubiera actualizado automáticamente, se puede actualizar desde el terminal mediante los siguientes comandos:

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

7.1.2 Mininet y Miniedit

Mininet es un emulador de código abierto que permite crear topologías de red virtuales compuestas por equipos hosts con Linux, switches y controladores basados en la arquitectura de redes definidas por software y el protocolo OpenFlow, lo que nos permite experimentar con la tecnología SDN. Para interactuar con la red se puede utilizar la línea de comandos o bien la interfaz gráfica Miniedit, de la que hablaremos más adelante.

Pese a que hay otros emuladores como EstiNet o Ns-3, se ha decidido trabajar con Mininet debido a los beneficios que ofrece tales como sencillez de implementación, capacidad de personalización de las topologías o la emulación de un entorno real donde se puede programar la transmisión de los paquetes y analizar las capacidades de los enlaces.

Centrándonos en la instalación de Mininet, pese a que son instrucciones específicas para instalar la aplicación en una computadora personal, en este trabajo se ha realizado sobre la Raspberry ya que también es posible y requiere exactamente los mismos comandos.

Según la página oficial de Mininet, mininet.org, existen tres maneras de instalar este software en el sistema operativo: instalación de una máquina virtual Mininet, instalación nativa desde el código fuente y la instalación mediante paquetes. En esta tesis se ha decidido utilizar la segunda opción.

Sin embargo, antes de todo necesitaremos instalar GitHub en nuestro sistema operativo ya que deberemos clonar un repositorio que se encuentra en GitHub a una localización local. Para ello:

```
$ sudo apt-get install git
```

Y, ahora sí, ya Podemos instalar el código fuente:

```
$ git clone git://github.com/mininet/mininet
```

Este código se guardará en un directorio local llamado Mininet y contiene la estructura de archivos del proyecto. A continuación, el usuario se debe ubicar en la carpeta *mininet* y descargar la versión que desee de Mininet:

```
$ cd mininet
$ git tag #muestra la lista con las versiones disponibles
1.0.0
2.0.0
2.1.0
2.1.0p1
2.1.0p2
.
# etc.
$ git checkout -b "versión que deseemos"
```

Personalmente recomiendo no elegir ninguna versión en particular y descargar la rama máster que contiene la última versión disponible ya que eligiendo otras versiones como la versión 2.2.1 me ha dado problemas tanto en una VM con Ubuntu en mi ordenador personal como en Raspbian con la Raspberry-Pi 4. Para descargar el código desde la rama máster:

```
$ git checkout #Sin elegir versión
```

Y, una vez realizado este paso, procedemos a la ejecución del script de instalación que se proporciona en la carpeta *mininet*. Para ello accedemos a la carpeta *util*, dentro de *mininet.*

```
$ cd ~/mininet/util/install.sh -a
```

Donde el comando -a es la opción que nos permite instalar todas las dependencias de la VM de Mininet como OpenvSwitch, Wireshark o el controlador POX.

Una vez la instalación se ha completado, debe salir el mensaje “enjoy mininet!” que nos indica que el script se ha ejecutado con éxito:

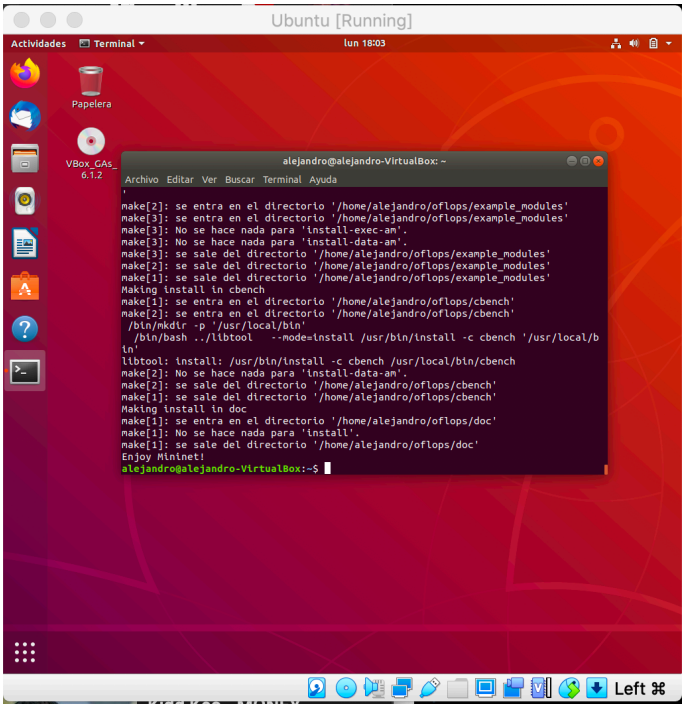


Figura 16: Mensaje de éxito una vez se ha ejecutado el script de instalación

Fuente: Propia, realizada durante la instalación de Mininet en una VM con Ubuntu.

Podemos comprobar que la aplicación funciona correctamente mediante el siguiente comando:

```
$ sudo mn –test pingall
```

El cual crea una red básica con un controlador (c0), dos hosts (h1 y h2) y un switch (s1) y, a continuación, inicia el switch y el controlador y realiza una prueba haciendo ping desde h1 a h2 y viceversa. Esta es una de las ventajas de Mininet, la creación de una red básica con la utilización de un solo comando: *sudo mn*.

Si no queremos trabajar desde la línea de comandos y preferimos utilizar Mininet mediante la interfaz gráfica MiniEdit, pese a que es menos potente, se debe de seguir los siguientes pasos:

El script MiniEdit está ubicado en la carpeta *examples*, dentro de la carpeta *mininet*. Para ejecutar el script, ejecutamos desde la línea de comandos:

```
$ cd ~/mininet/examples
$ sudo python miniedit.py
```

El script se debe ejecutar con la versión 3 de Python, pero por defecto, la Raspberry utiliza Pyhton2 pese a que también tiene instalada la versión posterior. El problema también ocurre si tenemos instalado Ubuntu 18.04 en una máquina virtual en nuestro ordenador personal. Si queremos evitar el error basta con ejecutar el siguiente comando:

```
$ sudo python3 miniedit.py
```

O, si se desea, se puede cambiar la opción por defecto a Pyhton3 creado un alias:

```
$ sudo python=python3
```

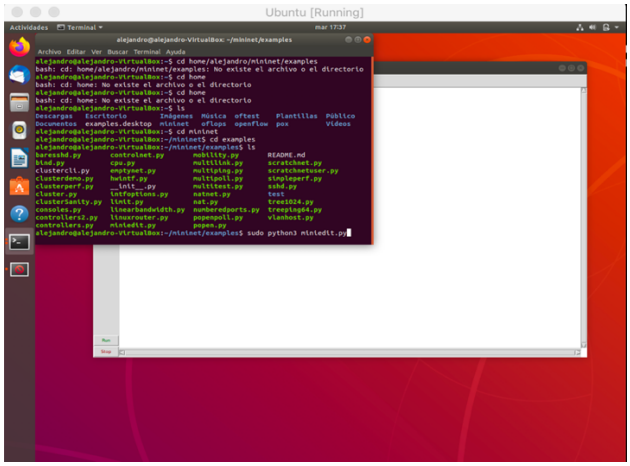


Figura 17: Proceso para utilizar MiniEdit

Fuente: Propia realizada durante la ejecución de Miniedit en una VM con Ubuntu.

La interfaz gráfica permite crear fácilmente una topología como la siguiente:

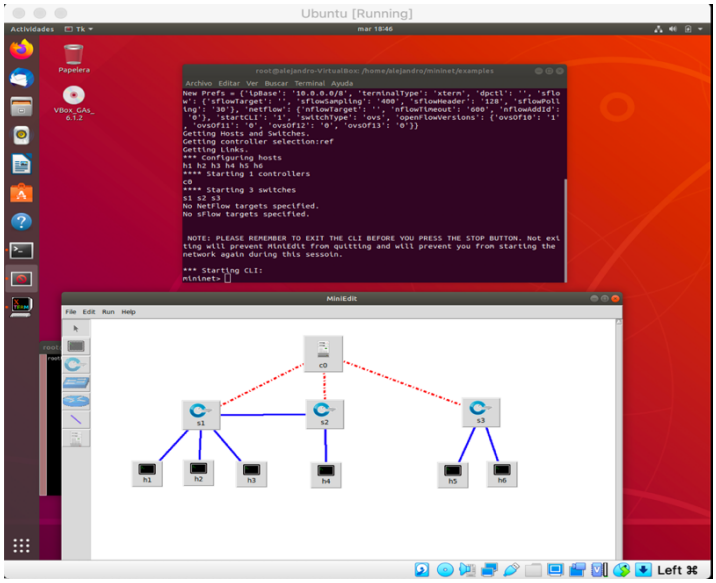


Figura 18: Creación de una topología con MiniEdit

Fuente: Propia realizada durante la ejecución de Miniedit en una VM con Ubuntu.

Como se puede ver en los iconos que tenemos en la parte izquierda de la ventana de Miniedit, tenemos diversas herramientas para crear una red SDN:

- Flecha: Permite seleccionar los nodos y cambiar su ubicación.
- Host: Introduce en la red hosts configurables.
- Switch: Permite crear un Open vSwitch configurable.
- Switch tradicional: Introduce conmutadores Ethernet independientes del controlador SDN y no son configurables.

- Router tradicional: Router básico independiente del controlador.
- Enlace: Conecta los distintos elementos de la red. Estos enlaces también son configurables.
- Controlador: Controlador configurable que sigue el estándar OpenFlow.

Además, podemos configurar la red desde la pestaña de preferencias de forma que podemos cambiar direcciones IP, los modelos de switch, los puertos o incluso el protocolo. Por supuesto, MiniEdit nos permite guardar o abrir nuestras redes o bien exportarlas como archivo Python ejecutable.

Sin embargo, como se ha comentado anteriormente, la interfaz gráfica MiniEdit es menos potente que la línea de comandos, por lo que es también aconsejable conocer algunos comandos básicos de Mininet para poder programar un entorno virtual.

El comando más importante de todos posiblemente sea el ya nombrado *sudo mn* por todo lo que nos permite hacer. Su estructura es la siguiente:

```
$ sudo mn --[Opción 1]=[Parámetro 1],[Argumentos] --[Opción n]=[Parámetro n],[Argumentos] …
```

Los parámetros dependen de la opción elegida, algunas de las opciones disponibles que ofrece Mininet son:

- *--h* o *--help*: Permite ver todas las opciones disponibles que tiene el comando *sudo mn*
- *--switch=[*parámetro*]*: Por defecto crea un Open vSwitch pero puede ser también *ivs* (IVSSwitch), *ovsbr* (OVSBridge) o *ovsk* (Open vSwitch en modo kernel), entre otros.
- *--host=[*parámetro*]*: Permite limitar el ancho de banda de un host virtual.
- *--controller=[*parámetro*]*: Por defecto crea un controlador compatible con OpenFlow. Mediante *none* se puede deshabilitar. También permite utilizar otros controladores con *nox* (controlador NOX), *ovsc* (controlador de prueba Open vSwitch), *ryu* (controlador Ryu) o *remote* (controlador externo a Mininet). Si no se especifica la dirección IP del controlador y el puerto, se utiliza por defecto la dirección 127.0.0.1 y el puerto 6633.
- *--link=[*parámetro*]*: Permite modificar los valores de un enlace como el ancho de banda o la latencia. Algunos parámetros para usar son *default* (por defecto), o bien si se desea personalizar los valores se utiliza el parámetro *tc, bw=[*Ancho de banda*],delay=[*Tiempo en ms*],loss=[*porcentaje de pérdidas*]*.
- *--topo=[*parámetro*]*= Se utiliza para cambiar el tamaño y tipo de la topología utilizada. Se puede utilizar *linear* (switches conectados en serie), *minimal* (topología de dos hosts y un switch), *single* (topología de un switch y N hosts), o *tree* (topología en árbol), entre otros.
- *--test=[*parámetro*]*: Se utiliza para realizar diversas acciones sobre la red. El parámetro para usar puede ser *cli* (para iniciar la simulación y utilizar la línea de comandos), *none* (empieza y finaliza la prueba), *build* (comenieza la prueba e imprime en pantalla el tiempo en ejecución de la simulación), *pingall* (realiza pings entre todos los hosts para comprobar la conectividad), *iperf* (mide el máximo ancho de banda entre dos hosts, las pérdidas y la latencia utilizando TCP) o *all* (durante la simulación realiza el *pingall* y el *iperf*).

- *--ipbase[*parámetro*]*: Permite elegir el espacio de direcciones utilizado por la red que por defecto es 10.0.0.0/8.
- *--verbosity[*parámetro*]*: Este comando se utiliza para depurar errores ya que Mininet clasifica la información obtenida según si es warning, info, debug…
- *-c*: Permite eliminar la topología creada y salir.

Existen muchos más parámetros que permiten modificar parámetros y dispositivos de la red o realizar otras configuraciones ya que Mininet es una herramienta muy potente que nos permite multitud de acciones. A modo de resumen se ha creado esta tabla con algunos de los comandos más utilizados para crear estructuras:

Mininet		Opción		Parámetro		Argumento
sudo mn	--	help			,	
		switch		default		
				ivs		
				ovs		
				ovsbr		stp=[1|0]
				ovsl		
				user		
				lxbr		stp=[1|0]
		host		cfs		
				rt		
				default		
				none		
				nox		
				ovsc		
				lvs		ip=[IP], port=[PUERTO]
		controller		ryu		
				default		
		link		tc		bw=[BW], delay=[TIME], loss=[%]
				linear		k=[SW], n=[HOST]
				minimal		
				single		k=[SW], n=[HOST]
				reversed		
				tree		k=[HOST]
		topo		torus		k=[HOST]
		clean				depth=[ALTURA], fanout=[RAMAS]
		custom		<fichero.py>		x=[N], t=[N]
				cli		
				none		
				build		
				pingpair		

sudo mn	--			pingfall	,	
				perf		
				iperfudp		
		test		all		
		xterms				
		ipbase		[IP]/[MASK]		
		mac				
		arp				
		verbosity		critical		
				error		
				warming		
				info		
				debug		
				output		
		inamespace				
		listenport		[PUERTO]		
		nolistenport				
		nat				
		version				

Tabla 6: Listado de parámetros de *sudo mn*

Fuente: Propia

A parte de todas estas opciones que nos permite realizar el comando *sudo mn*, tenemos muchos otros comandos para la línea de comandos (CLI) de Mininet que podemos encontrar si ejecutamos *help* en la CLI. Algunos de los comandos que podemos encontrar son:

- EOF: Finaliza la simulación de Mininet.
- dpctl: Permite gestionar el conmutador con acciones como añadir flujos o cambiar la configuración.
- dump: Se muestra información detallada de la red como tipos de dispositivos, nombre, puertos, direcciones IP…
- exit: Finaliza la simulación y cierra Mininet.
- iperf: Mide la velocidad de conexión entre dos hosts.
- link: Establece o elimina un enlace entre dos hosts.
- net: Muestra los enlaces y puertos utilizados en ellos.
- nodes: Muestra los nodos de la red.
- ports: Muestra los puertos e interfaces de cada switch de la red.
- quit: Finaliza la emulación.
- source: Permite leer comandos Mininet desde un archivo.
- switch: Inicia o para un switch.
- time: Muestra el tiempo que tarda en ejecutarse un comando de Mininet. Su sintaxis es *time [comando]*.
- xterm: Permite abrir un nuevo terminal para un nodo en particular.

En la siguiente tabla podemos ver a modo de resumen la lista de todos los comandos CLI que ofrece Mininet:

Comando	Argumento	Descripción
EOF		Cierra la simulación
exit		Cierra la simulación
quit		Cierra la simulación
help		Muestra la lista de comandos
dump		Muestra información más precisa de la red
net		Muestra información de los enlaces
intfs		Muestra información de los interfaces
nodes		Muestra los nodos en uso
ports		Muestra los puertos en uso
time	[COMANDO]	Tiempo que se está ejecutando la simulación
switch	[SW][start|stop]	Para habilitar o no un switch
links		Reporte de enlaces operativos
link	[NODO1][NODO2]	Permite habilitar o no un enlace
noecho	[HOST][CMD args]	Ejecuta comandos Shell en los hosts
sh	[CMD args]	Ejecuta comandos Shell en el host anfitrión
source	<file>	Lee comandos Mininet de un fichero
pingall		Realiza pings entre todos los nodos
pingallfull		Realiza pings a todos los nodos y muestra información
pingpair		Prueba conexión entre dos hosts

pingpairfull		Prueba conexión entre dos hosts y muestra información
iperf	[HOST1][HOST2]	Ancho de banda en TCP
iperudp	[BW][HOST1][HOST2]	Ancho de banda en UDP
px	[PYTHON]	Ejecuta declaraciones Python
py	[OBJETO.FUNCION()]	Ejecuta expresiones Python
xterm	[HOSTn]…	Abre CLIs
x	[HOST][CMD args]	Crea un túnel X11
gterm	[HOSTn]…	Abre un CLI GUI
dpctl	[COMANDO][args]	Ejecuta funciones dpctl

Tabla 7: Listado de comandos de Mininet

Fuente: Propia

7.1.3 Controlador Ryu

Como ya hemos visto en otros apartados, existen numerosos tipos de controladores, tanto públicos como de fabricante, que utilizan o no el protocolo OpenFlow. En esta tesis se ha decidido utilizar el controlador Ryu y mostrar qué permite realizar. El motivo de su elección es:

- Porque es de software libre.
- Utiliza, entre otros, el protocolo OpenFlow, que es el más extendido dentro de las redes SDN. Soporta además todas sus versiones, desde la v1.0 hasta la v1.5.
- Se programa en lenguaje Python y es compatible con la plataforma Linux.
- Tiene una API muy completa y muy bien documentada en su página web: osrg.github.io.
- Es compatible con el software de virtualización de Mininet.

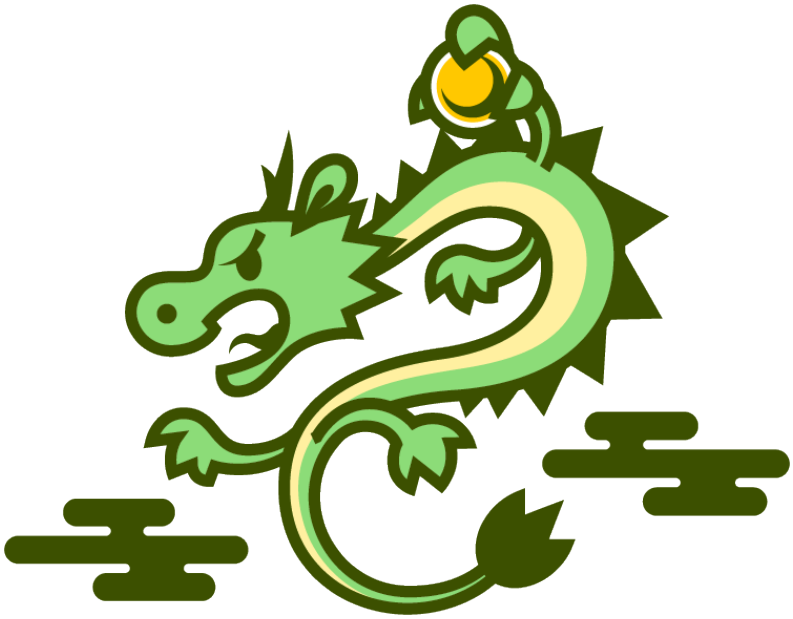


Figura 19: Logo de Ryu

Fuente: osrg.github.io

Cabe destacar que Ryu no es exactamente un controlador si no un Component-based SDN framework, que es un entorno de trabajo que permite mediante software implementar redes SDN. La utilización de Ryu permite mediante la invocación de un conjunto de aplicaciones manejar eventos de red, analizar cualquier solicitud de cambio y reaccionar a los cambios de red instalando nuevos flujos, si es necesario. Algunos rasgos que ofrece Ryu son:

- Capacidad de escuchar eventos asíncronos y observarlos.
- Capacidad de analizar paquetes entrantes y enviarlos en la red.
- Capacidad de crear y enviar mensajes OpenFlow.
- Agilidad para crear infraestructuras SDN y flexibilidad en la API Northbound.

Para entender la arquitectura de Ryu es necesario saber que es una estructura formada por diversos componentes que son las aplicaciones. Ryu ofrece diversas aplicaciones, pero se pueden modificar las aplicaciones a utilizar o implementar nuevas.

Algunos de los elementos que podemos encontrar en Ryu son las librerías con protocolos como Netconf, vrrp, xFlow, OVSDB JSON o Netflow entre otros, así como aplicaciones Ryu como OF REST, Firewall o Topology Viewer. Hay también aplicaciones que permiten la interoperabilidad entre switches como L2 Switch o Open vSwitchEndPoint así como aplicaciones que permiten integrarse con otros componentes de la red como IDS u OpenStack Quantum.

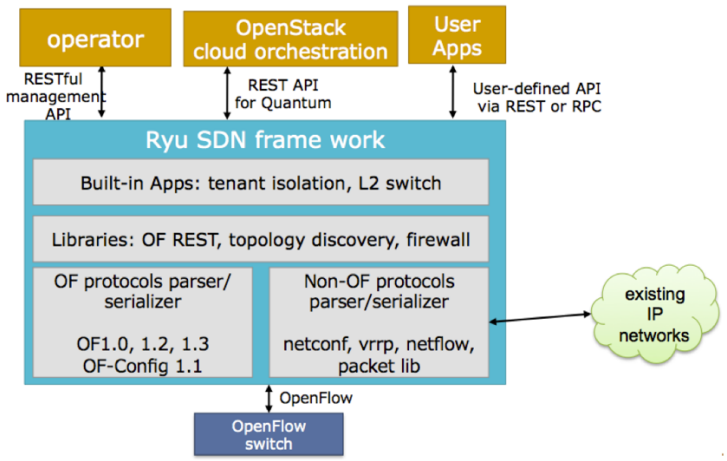


Figura 20: Arquitectura del controlador Ryu

Fuente: Universidad de Oregon

Como se puede observar, Ryu ofrece multitud de opciones. Ofrece grandes posibilidades y a diferencia de otros controladores permite implementar muchos módulos fácilmente. El módulo Firewall es parte importante de la seguridad de una red al controlar los flujos de entrada y salida entre diferentes hosts. El módulo Intrusion Detection System (IDS) también permiten multitud de mejoras en la seguridad de las redes virtuales.

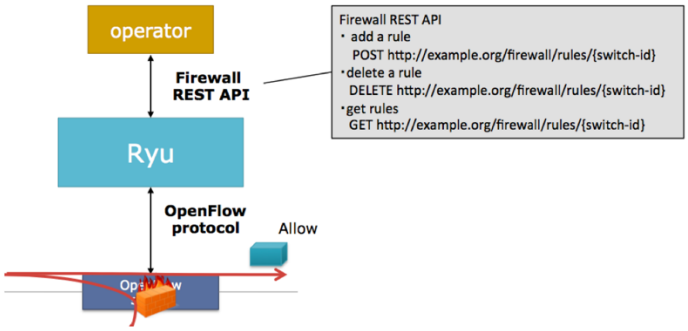


Figura 21: Funcionamiento del módulo de Ryu Firewall

Fuente: Universidad de Oregon

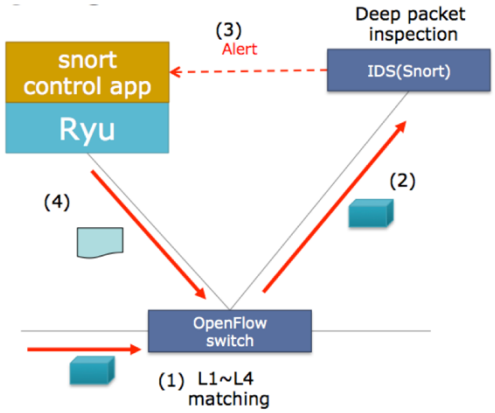


Figura 22: Funcionamiento del módulo de Ryu IDS

Fuente: Universidad de Oregon

Para poder utilizar Ryu se requiere tener instalado Mininet y Python. Como hemos visto anteriormente, se ha procedido a la instalación del sistema operativo Raspbian en la Raspberry-Pi que utiliza Mininet, así como a la instalación de Mininet.

Ryu se puede instalar desde la línea de comandos de dos formas:

1. Desde el administrador de paquetes Python, es la forma más sencilla:

```
$ pip install ryu
```

2. Desde el código fuente:

```
$ git clone git://github.com/osrg/ryu.git
$ cd ryu
$ python ./setup.py install
```

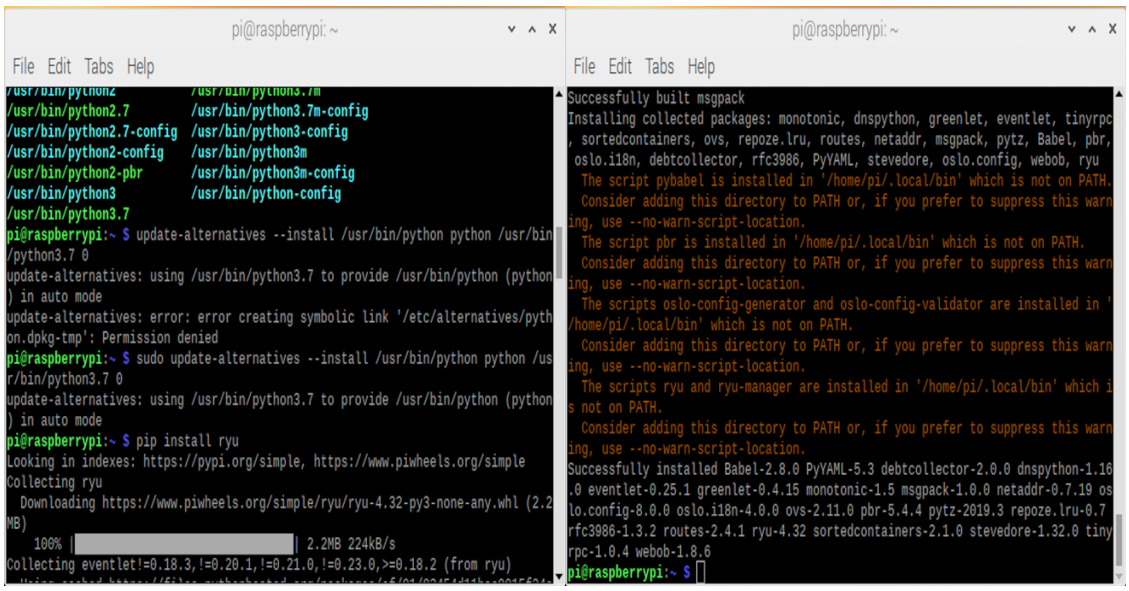


Figura 23: Instalación Ryu

Fuente: Propia realizada durante la ejecución de Ryu en la Raspberry-Pi.

Una vez instalado, podemos comprobar su funcionamiento mediante la ejecución de una aplicación con Ryu:

```
$ cd ryu
$ ryu-manager –verbose ryu.app.example_switch_13.py
```

Como resultado nos tiene que devolver por pantalla un mensaje parecido al siguiente:

```
loading app ryu.app.example_switch_13
loading app ryu.controller.ofp_nadler
instantiating app ryu.app.example_switch_13 of
ExampleSwitch13
.
.
```

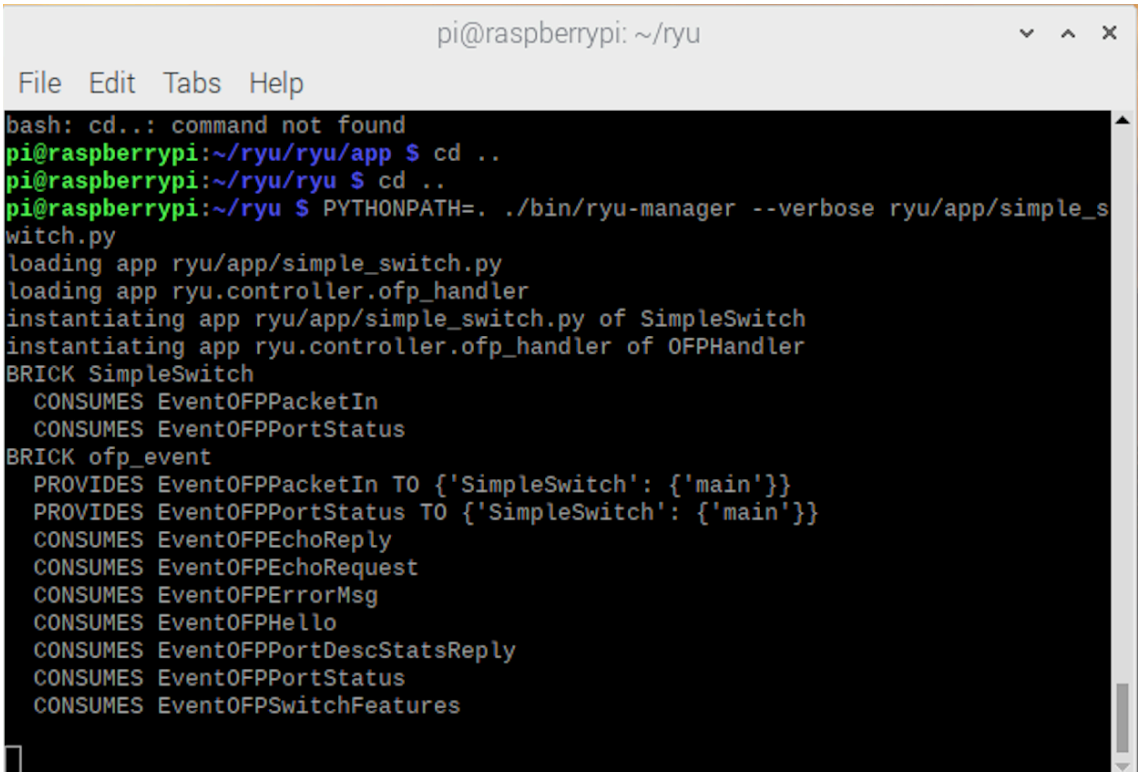


Figura 24: Comprobación instalación Ryu

Fuente: Propia realizada durante la ejecución de Ryu en la Raspberry-Pi.

7.2 Análisis práctico

El controlador Ryu viene instalado junto con diferentes archivos Pyhton ejecutables que implementan diferentes aplicaciones muy útiles a la hora de configurar y gestionar redes SDN. En concreto, podemos encontrar la implementación de un Switch Hub, de un Monitor de Tráfico básico, agregación de links, el protocolo Spanning Tree, uso de un Firewall, de un Router, de herramientas de QoS y de una herramienta de testeo.

En esta tesis se ha decidido implementar cuatro de estos ejemplos, en concreto son el Firewall, el Spanning Tree, las herramientas de QoS y el Monitor de Tráfico

Para cada aplicación, se creará un entorno simple SDN en el que se implementará el controlador Ryu y, desde este, se ejecutará el archivo Pyhton en cuestión. Para la simulación, se generará tráfico de red o bien se enviarán pings entre diferentes hosts de manera que se pueda comprobar la correcta utilización de cada aplicación.

7.2.1 Firewall

La primera aplicación que se va a implementar se trata de un firewall. Un firewall o cortafuegos es un dispositivo hardware o software diseñado para permitir, limitar, cifrar o descifrar el tráfico que circula en una red.

Para permitir o no el tráfico de un origen a un destino se utilizan reglas ACL que permiten al usuario indicar el tráfico que desea bloquear o permitir.

Hasta ahora la implementación de las reglas del firewall generalmente tiene lugar en routers o switches específicos y el administrador de la red debe configurar manualmente éstos. En este experimento se va a mostrar como el administrador de la red sólo crea las reglas en el controlador, sin necesidad de ir dispositivo a dispositivo añadiendo las reglas.

El entorno de simulación a implementar es el siguiente:

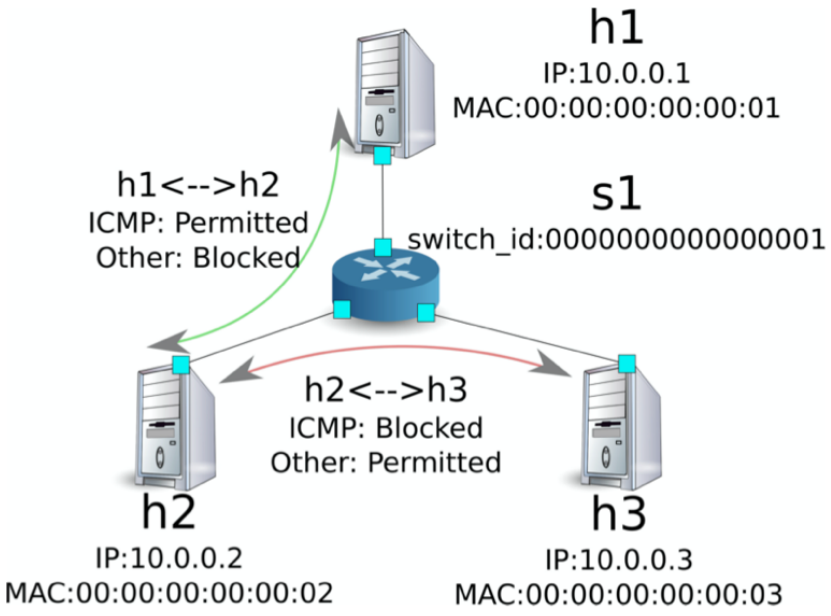


Figura 25: Entorno de simulación a implementar para el Firewall

Fuente: Documentación de Ryu

Para ello, se debe utilizar el siguiente comando en el terminal para crear la red en Mininet:

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

Indica que queremos 3 hosts, direcciones MAC por defecto, uso del switch ovsk, uso de un controlador remoto y la instrucción -x quivale a xterm y permite mostrar los terminales de cada dispositivo. Por defecto Mininet asigna a los hosts las direcciones IPv4 10.0.0.1 para el primero, 10.0.0.2 para el segundo etcétera.

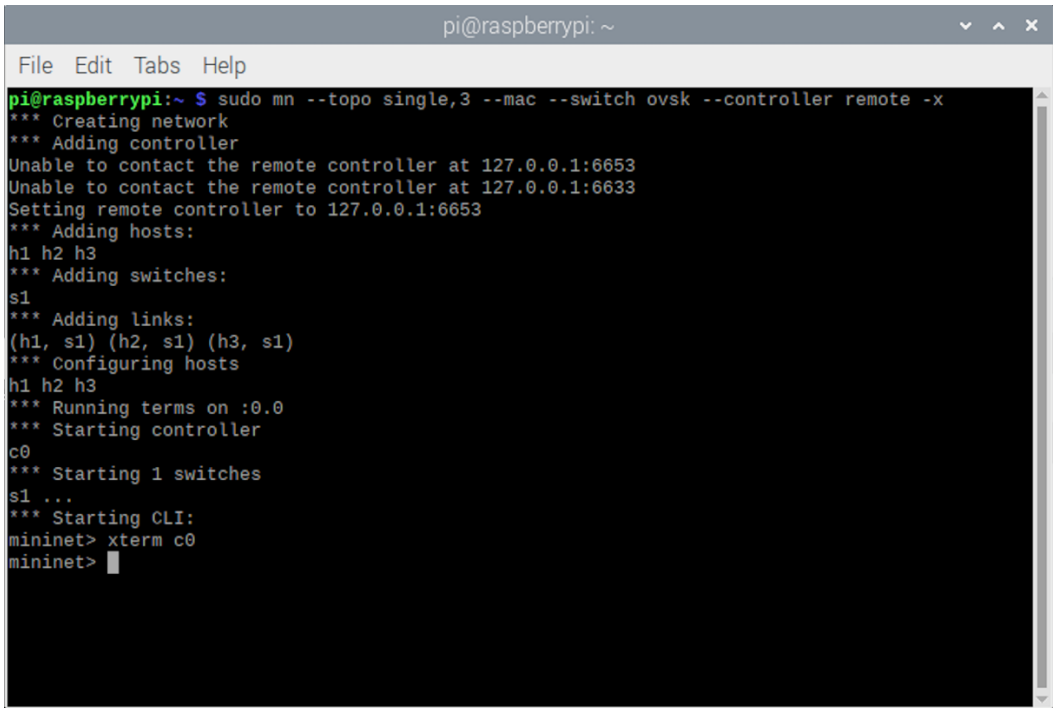


Figura 26: Creación de la topología en Mininet

Seguidamente, se elige la versión OpenFlow que utilizará el switch s1 (root):

```
$ ovs-vsctl set Bridge s1 protocols=OpenFlow13
```

A continuación, se selecciona como controlador remoto Ryu y se ejecuta el archivo Pyhton donde se encuentra el firewall en el controller c0 (root):

```
$ ryu-manager ryu.app.rest_firewall
```

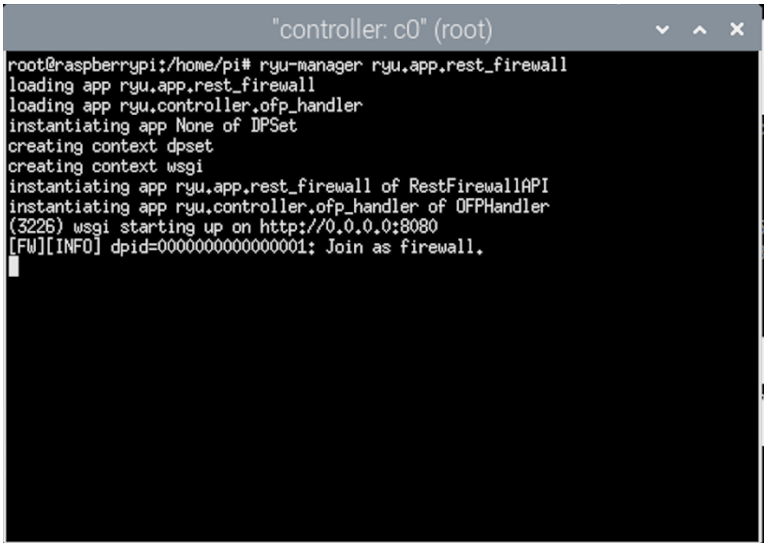


Figura 27: Elección de Ryu y del Firewall

Una vez se ha iniciado el Firewall, por defecto se deshabilita toda la comunicación por lo que hay que volver a habilitarla en el Node c0 (root)

```
$ curl -X PUT
http://localhost:8080/firewall/module/enable/000000000
0000001
.
.
$ curl http://localhost:8080/firewall/module/status
```

Ahora, ya se puede simular la red. Primero se probará un ping entre el host 1 y 2 para ver que efectivamente es bloqueado ya que las reglas todavía no se han establecido:

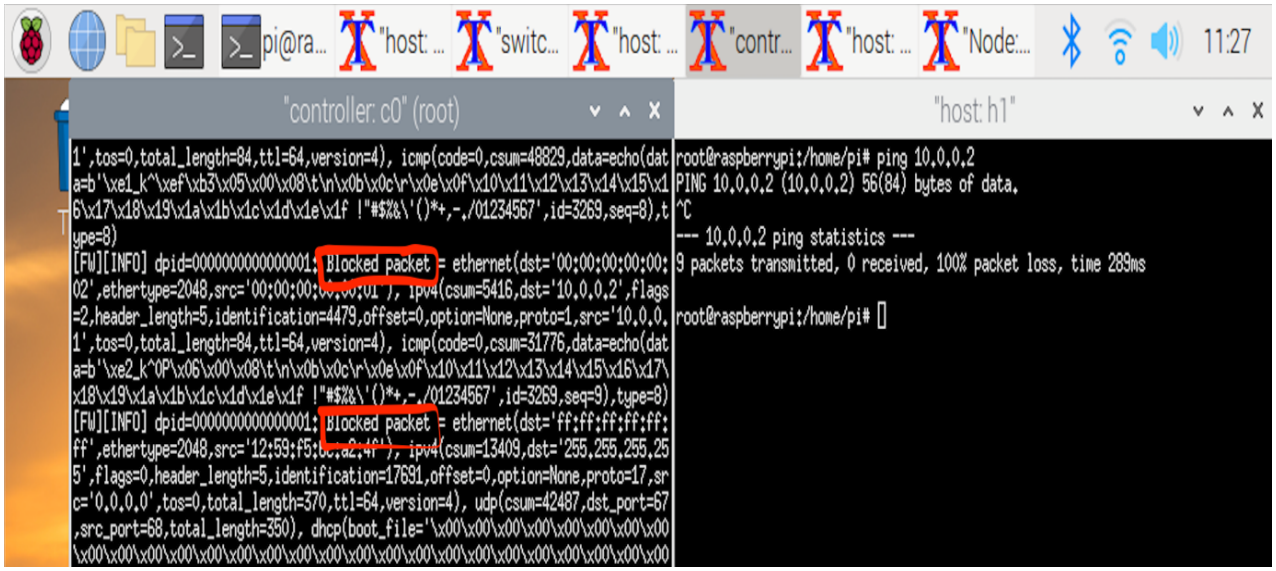


Figura 28: Ping bloqueado entre el host 1 y el host 2

Ahora se añade una regla que permita realizar pings entre el host 1 y el host2, la regla debe añadirse para los dos sentidos.

Origen	Destino	Protocolo	Permiso	ID
10.0.0.1/32	10.0.0.2/32	ICMP	Allow	1
10.0.0.2/32	10.0.0.1/32	ICMP	Allow	2

Tabla 8: Reglas de firewall para permitir ping entre los hosts 1 y 2

Para añadirlas, se ejecuta el siguiente comando en el Node c0 (root):

```
$ curl -X POST -d '{"nw_src": "10.0.0.1/32", "nw_dst":
"10.0.0.2/32", "nw_proto": "ICMP"}'
http://localhost:8080/firewall/rules/0000000000000001
.
.
$ curl -X POST -d '{"nw_src": "10.0.0.2/32", "nw_dst":
"10.0.0.1/32", "nw_proto": "ICMP"}'
http://localhost:8080/firewall/rules/0000000000000001
```

Y agregamos las reglas como entradas de flujo del conmutador switch s1 (root):

```
$ ovs-ofctl -O openflow13 dump-flows s1
```

A continuación, añadimos reglas para permitir todos los paquetes IPv4. Esto incluye también los pings, por lo que habrá que escribir las pertinentes reglas para bloquear los pings.

Origen	Destino	Protocolo	Permiso	ID
10.0.0.2/32	10.0.0.3/32	any	Allow	3
10.0.0.3/32	10.0.0.2/32	any	Allow	4

Tabla 9: Reglas de firewall para permitir paquetes entre los hosts 2 y 3

```
$ curl -X POST -d '{"nw_src": "10.0.0.2/32", "nw_dst":
"10.0.0.3/32"}'
http://localhost:8080/firewall/rules/0000000000000001
.
.
$ curl -X POST -d '{"nw_src": "10.0.0.3/32", "nw_dst":
"10.0.0.2/32"}'
http://localhost:8080/firewall/rules/0000000000000001
```

Y registramos las reglas como entradas de flujo en el conmutador, con la misma instrucción que se ha utilizado previamente.

Para bloquear los pings entre el host 3 y el 2, debemos establecer un valor de prioridad mayor que 1, que es el valor por defecto de la prioridad.

Prioridad	Origen	Destino	Protocolo	Permiso	ID
10	10.0.0.2/32	10.0.0.3/32	ICMP	Block	5
10	10.0.0.3/32	10.0.0.2/32	ICMP	Block	6

Tabla 10: Reglas de firewall para bloquear pings entre los hosts 2 y 3

```
$ curl -X POST -d '{"nw_src": "10.0.0.2/32", "nw_dst":
"10.0.0.3/32", "nw_proto": "ICMP", "actions": "DENY",
"priority": "10"}'
http://localhost:8080/firewall/rules/0000000000000001
.
.
```

```
$ curl -X POST -d '{"nw_src": "10.0.0.3/32", "nw_dst":
"10.0.0.2/32", "nw_proto": "ICMP", "actions": "DENY",
"priority":"10"}'
http://localhost:8080/firewall/rules/0000000000000001
```

Y volvemos a registrar las reglas como entradas de flujo del conmutador.

Para confirmar las reglas que se han establecido podemos ejecutar en el terminal del controlador Node c0 (root):

```
$ curl
http://localhost:8080/firewall/rules/0000000000000001
```

```
root@raspberrypi:/home/pi# curl http://localhost:8080/firewall/rules/0000000000
000001
[{"switch_id": "0000000000000001", "access_control_list": [{"rules": [{"rule_id"
: 1, "priority": 1, "dl_type": "IPv4", "nw_src": "10.0.0.1", "nw_dst": "10.0.0.2
", "nw_proto": "ICMP", "actions": "ALLOW"}, {"rule_id": 2, "priority": 1, "dl_ty
pe": "IPv4", "nw_src": "10.0.0.2", "nw_dst": "10.0.0.1", "nw_proto": "ICMP", "ac
tions": "ALLOW"}, {"rule_id": 5, "priority": 10, "dl_type": "IPv4", "nw_src": "1
0.0.0.2", "nw_dst": "10.0.0.3", "nw_proto": "ICMP", "actions": "DENY"}, {"rule_i
d": 6, "priority": 10, "dl_type": "IPv4", "nw_src": "10.0.0.3", "nw_dst": "10.0.
0.2", "nw_proto": "ICMP", "actions": "DENY"}, {"rule_id": 3, "priority": 1, "dl_
type": "IPv4", "nw_src": "10.0.0.2", "nw_dst": "10.0.0.3", "actions": "ALLOW"},
{"rule_id": 4, "priority": 1, "dl_type": "IPv4", "nw_src": "10.0.0.3", "nw_dst":
root@raspberrypi:/home/pi#
```

Figura 29: Confirmación de reglas de firewall

Ahora sí debería permitirse la comunicación mediante pings entre el h1 y h2:

```
root@raspberrypi:/home/pi# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1.65 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.189 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.188 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.183 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 49ms
rtt min/avg/max/mdev = 0.183/0.551/1.647/0.633 ms
root@raspberrypi:/home/pi#
```

Figura 30: Confirmación de habilitación de pings entre h1 y h2

Por otro lado, pese a que se permite la comunicación entre el h2 y h3, hemos denegado el permiso a los pings entre estos hosts por lo que si intentamos realizar un ping entre ellos la conexión fallará:

Figura 31: Bloqueo de pings entre h2 y h3

Una vez visto como se comporta la red cuando se han creado diversas reglas de firewall, vamos a ver como se eliminan. Para ello vamos a implementar la siguiente estructura de la red:

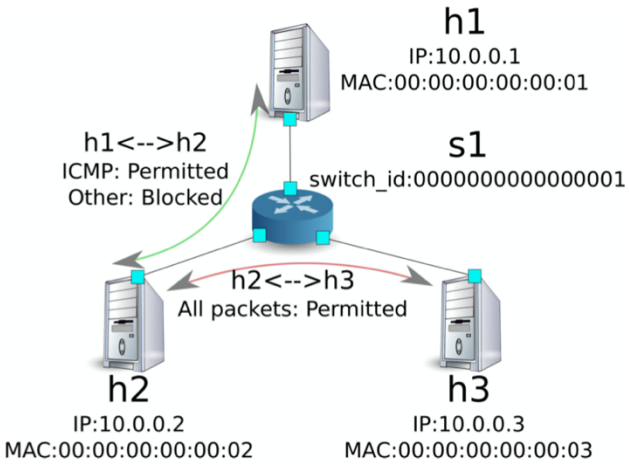


Figura 32: Entorno de simulación 2 a implementar para el Firewall

Fuente: Documentación de Ryu

Para implementar este entorno es necesario eliminar las últimas dos reglas que habían sido creadas, las cuales bloqueaban el tráfico ICMP entre los hosts 2 y 3. Los IDs de estas reglas son el 5 y el 6. Para eliminarlas debemos escribir los siguientes comandos en el controlador Node c0 (root):

```
$ curl -X DELETE -d '{"rule_id": "5"}'
http://localhost:8080/firewall/rules/0000000000000001
.
.
$ curl -X DELETE -d '{"rule_id": "6"}'
http://localhost:8080/firewall/rules/0000000000000001
```

Ahora sí debería estar permitido el tráfico ICMP entre ambos hosts:

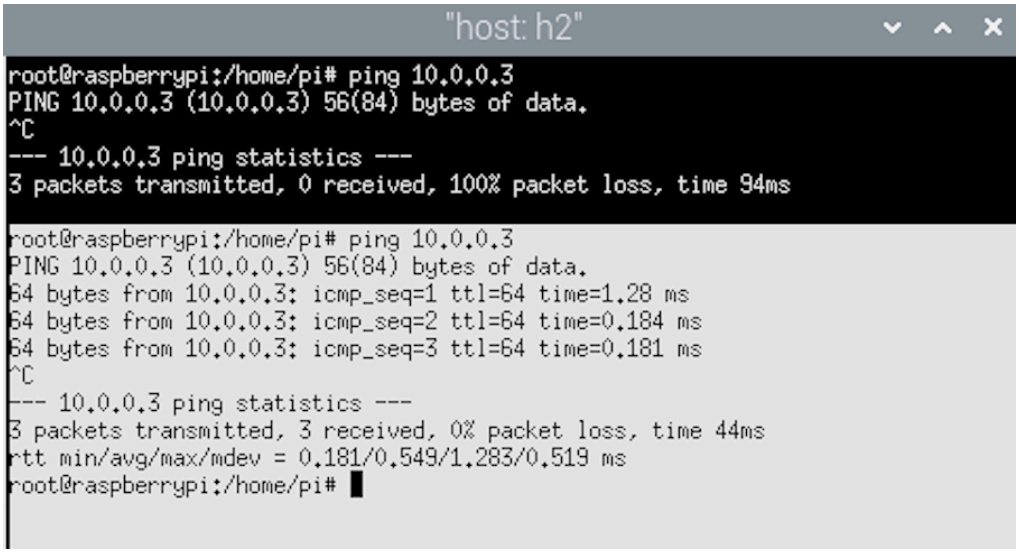


Figura 33: Ping correcto entre h2 y h3

Como podemos observar, SDN nos ha permitido trabajar de manera remota y centrándonos en la figura del controlador, sin la necesidad de trabajar específicamente desde el conmutador para añadir las reglas de firewall. Quizás en un entorno de trabajo local no supone un gran coste temporal, pero en una corporación con un elevado número de routers o conmutadores, el tiempo ahorrado es mucho mayor si se tiene establecida una arquitectura definida por software.

7.2.2 Spanning Tree

La siguiente aplicación que veremos implementada con Ryu es el protocolo Spanning Tree. El Spanning Tree (STP: 802.1D) es un protocolo de capa 2 que permite suprimir la aparición de bucles en topologías de red debido a la redundancia que se provoca en sus enlaces. Además, permite asegurar la transmisión de la red recalculando automáticamente la ruta en caso de fallo de un enlace.

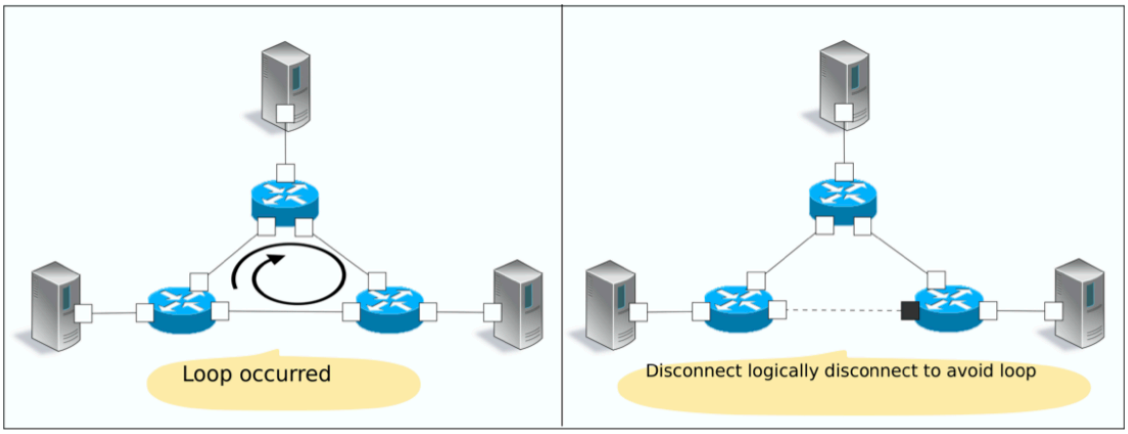


Figura 34: Actuación del Spanning Tree frente a un bucle

Fuente: Documentación de Ryu

En STP, los paquetes BPDU (Bridge Protocol Data Unit) son intercambiados entre puentes para decidir por qué ruta debe ser transferida la información. Para ello es necesario:

- Elegir el puente raíz, que es el que tiene el menor ID y será el encargado de transferir los paquetes BPDU originales.
- Decidir el rol de los puertos según el coste que tiene cada puerto para llegar al puente raíz.
    - Puerto Raíz: Puerto que tiene el mejor coste entre puentes para alcanzar al puente raíz.
    - Puerto Designado: Son los puertos que tienen un pequeño coste para llegar al puente raíz de cada enlace. Todos los puertos del puente raíz son designados.
    - Puerto No Designado: No se transfiere información por ellos.

Para poder comprobar el funcionamiento del Spanning Tree en redes SDN con el controlador Ryu se ha utilizado la siguiente topología:

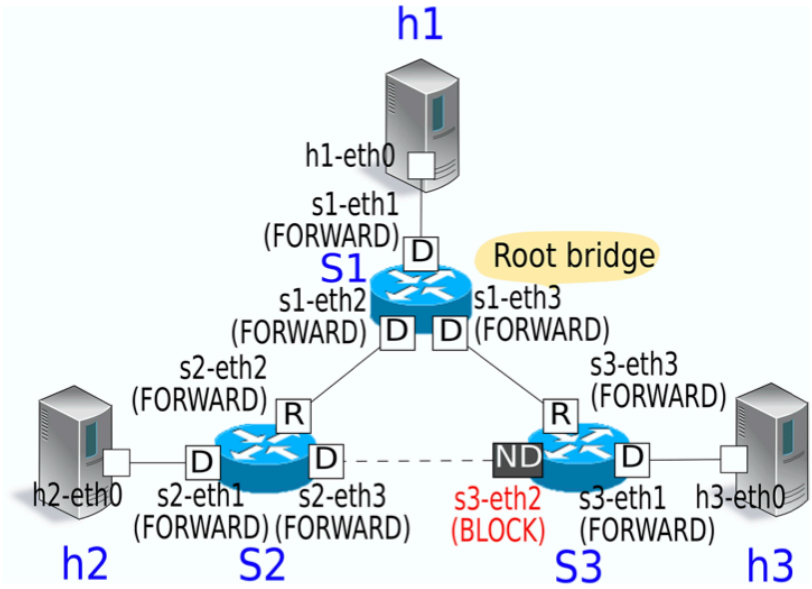


Figura 35: Topología a implementar para el Spanning Tree

Fuente: Documentación de Ryu

Para ello implementamos en el terminal:

```
$ curl -O https://raw.githubusercontent.com/osrg/ryu-book/master/sources/spanning_tree.py
$ sudo ./spanning_tree.py
```

```
pi@raspberrypi:~ $ sudo python spanning_tree.py
Unable to contact the remote controller at 127.0.0.1:6633
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s3-eth1
s1 lo:  s1-eth1:h1-eth0 s1-eth2:s2-eth2 s1-eth3:s3-eth3
s2 lo:  s2-eth1:h2-eth0 s2-eth2:s1-eth2 s2-eth3:s3-eth2
s3 lo:  s3-eth1:h3-eth0 s3-eth2:s2-eth3 s3-eth3:s1-eth3
c0
mininet>
```

Figura 36: Creación de la topología

Lo siguiente es establecer la versión OpenFlow en lo switches s1, s2 y s3:

```
$ ovs-vsctl set Bridge s1 protocols=OpenFlow13
$ ovs-vsctl set Bridge s2 protocols=OpenFlow13
$ ovs-vsctl set Bridge s3 protocols=OpenFlow13
```

Ahora, desde el controlador cargamos el controlador Ryu y ejecutamos el archivo Python para establecer el protocolo STP:

```
$ ryu-manager ryu.app.simple_switch_stp13
```

Una vez se ha completado la conexión entre cada switch OpenFlow y el controlador, se intercambian los paquetes BPDU y se selecciona el puente raíz, los roles de los puertos y los estados de estos últimos:

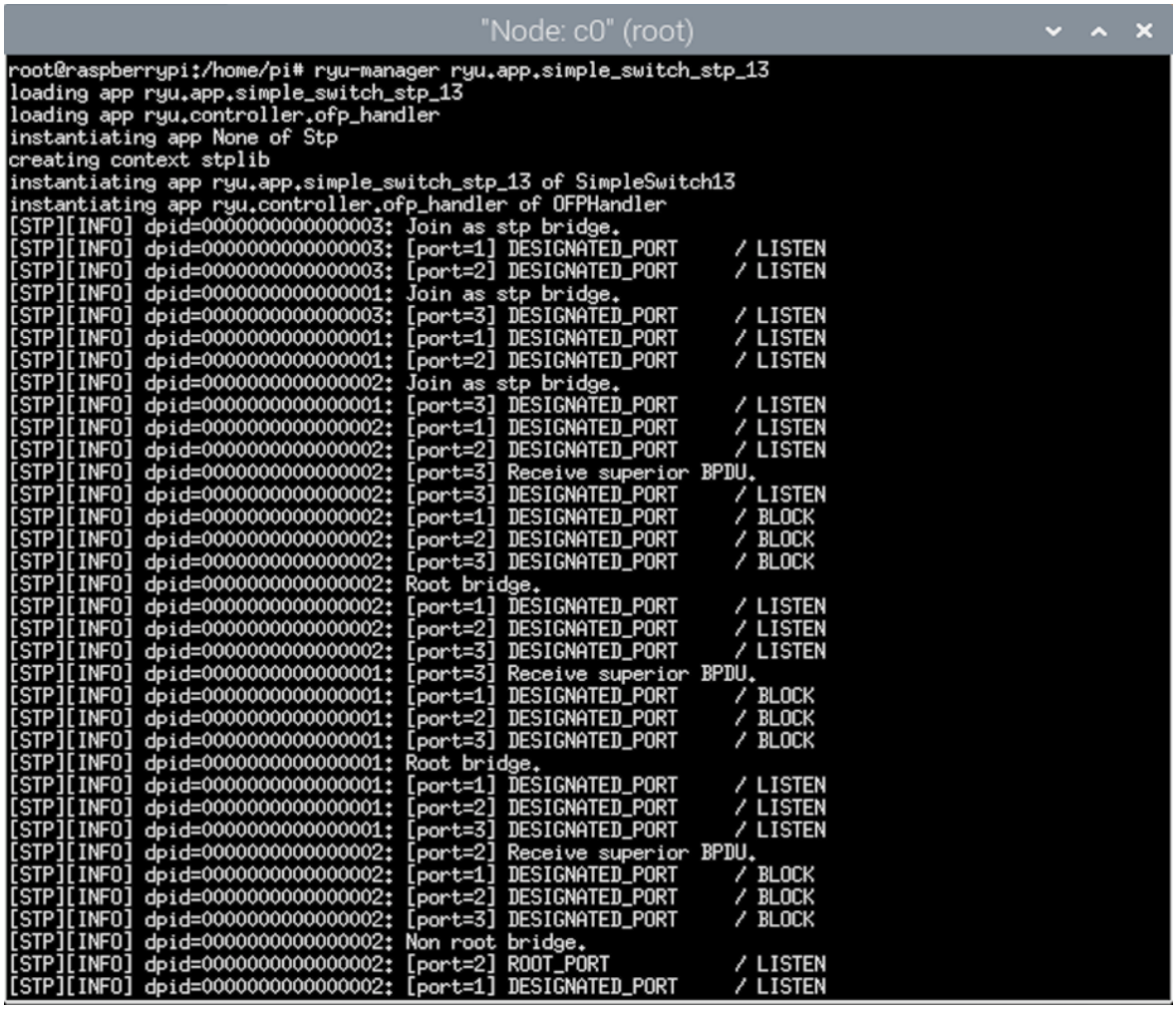


Figura 37: Establecimiento del Spanning Tree

Para comprobar que no se producen bucles, vamos a enviar ping entre h1 y h2, pero antes ejecutamos la herramienta *tcpdump* que permite analizar el tráfico que circula por la red. Así pues, desde cada switch ejecutamos:

```
$ tcpdump -i s1-eh2 arp
$ tcpdump -i s2-eh2 arp
$ tcpdump -i s3-eh2 arp
```

Y en la consola dónde hemos creado la topología realizamos pings entre h1 y h2. Como resultado, podremos ver en la salida del *tcpdump* de cada switch como no se ha producido ningún bucle en los mensajes ARP:

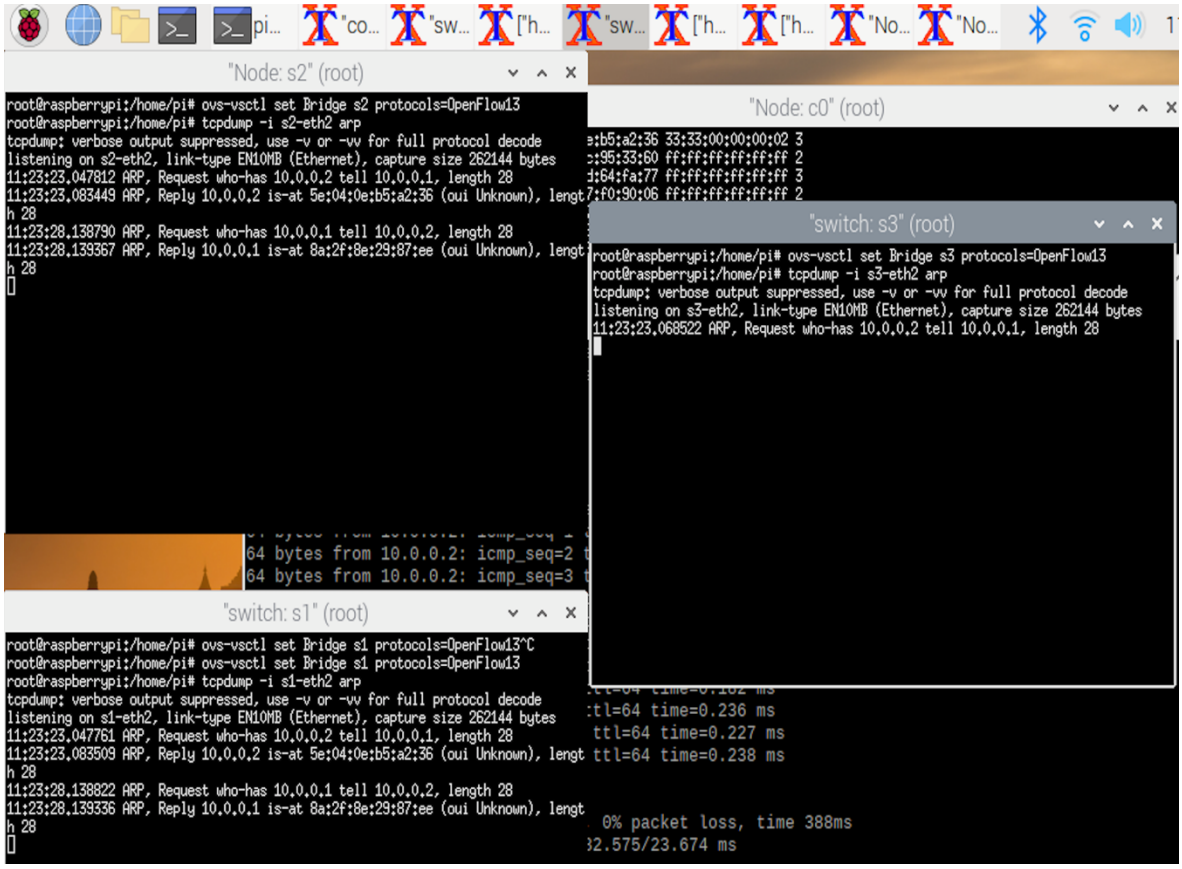


Figura 38: Confirmación del funcionamiento del protocolo STP

Seguidamente, procedemos a observar cómo actúa el protocolo cuando se detecta un fallo. De forma que la topología de red que tendremos es la siguiente:

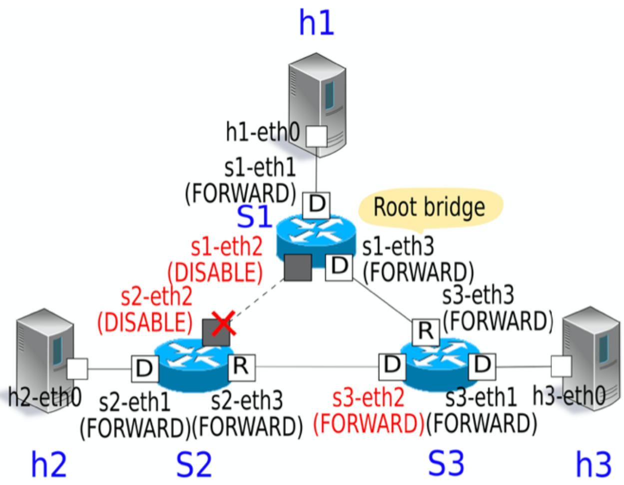


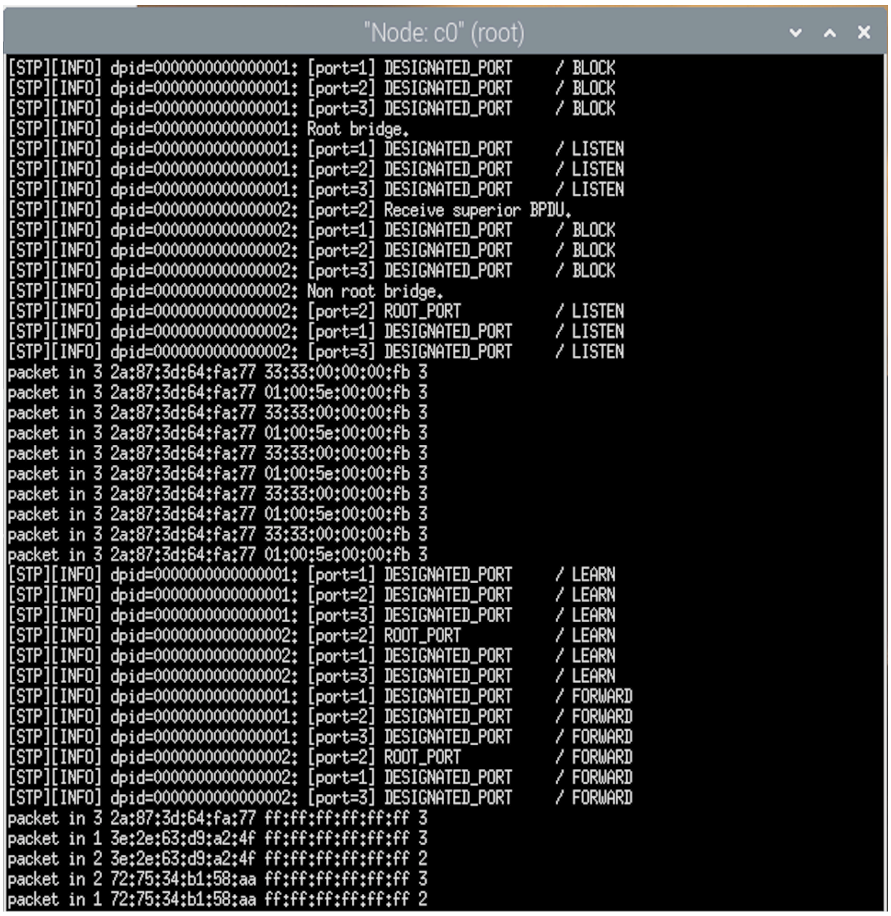
Figura 39: Topología del escenario 2

Fuente: Documentación de Ryu

Para ello escribimos en el terminal del switch Node s2 (root):

```
$ ifconfig s2-eth2 down
```

Y se puede observar en el controlador como se ejecuta la recalculación de STP:



"Node: c0" (root)

```
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: Root bridge.
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=2] Receive superior BPDU.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: Non root bridge.
[STP][INFO] dpid=0000000000000002: [port=2] ROOT_PORT           / LISTEN
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / LISTEN
packet in 3 2a:87:3d:64:fa:77 33:33:00:00:00:fb 3
packet in 3 2a:87:3d:64:fa:77 01:00:5e:00:00:fb 3
packet in 3 2a:87:3d:64:fa:77 33:33:00:00:00:fb 3
packet in 3 2a:87:3d:64:fa:77 01:00:5e:00:00:fb 3
packet in 3 2a:87:3d:64:fa:77 33:33:00:00:00:fb 3
packet in 3 2a:87:3d:64:fa:77 01:00:5e:00:00:fb 3
packet in 3 2a:87:3d:64:fa:77 33:33:00:00:00:fb 3
packet in 3 2a:87:3d:64:fa:77 01:00:5e:00:00:fb 3
packet in 3 2a:87:3d:64:fa:77 33:33:00:00:00:fb 3
packet in 3 2a:87:3d:64:fa:77 01:00:5e:00:00:fb 3
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000002: [port=2] ROOT_PORT           / LEARN
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000002: [port=2] ROOT_PORT           / FORWARD
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / FORWARD
packet in 3 2a:87:3d:64:fa:77 ff:ff:ff:ff:ff:ff 3
packet in 1 3e:2e:63:d9:a2:4f ff:ff:ff:ff:ff:ff 3
packet in 2 3e:2e:63:d9:a2:4f ff:ff:ff:ff:ff:ff 2
packet in 2 72:75:34:b1:58:aa ff:ff:ff:ff:ff:ff 3
packet in 1 72:75:34:b1:58:aa ff:ff:ff:ff:ff:ff 2
```

Figura 40: Recalculación de la ruta

Como hemos podido observar, el establecimiento de protocolos de red como el Spanning Tree es posible en arquitecturas definidas por software. Además, permite actuar desde el controlador sin necesidad de establecer el protocolo individualmente en cada conmutador. Ryu es quien calcula la asignación de puentes y el rol de los puertos y, en caso de fallo, el controlador Ryu sabe que está utilizando el protocolo STP y actúa en función a ello recalculando las rutas y asignando los roles oportunos a los puertos.

7.2.3 Herramientas de QoS

Las Qos (Calidad de Servicio – Quality of Service) es una tecnología que permite transferir datos en función de la prioridad basada en el tipo de datos o en la reserva del ancho de banda de la red para una comunicación en particular con el fin de comunicarse con un ancho de banda constante, así como según otras propiedades de la red como la tasa de errores, el rendimiento, el retraso en la transmisión, jitter, disponibilidad etc. Para poder aplicar QoS en una red en necesario implementar mecanismos como la priorización de tráfico o la garantía de un ancho de banda mínimo.

La aplicación de QoS es un requisito básico para poder implantar servicios interactivos.

Para aplicar QoS en una red SDN se ha decidido implementar una topología de red sencilla basada en dos hosts, un switch y el controlador Ryu. El mecanismo de calidad de servicio a implementar es la garantía de un ancho de banda mínimo.

Primero de todo implementamos en la línea de comandos la siguiente instrucción para crear la topología:

```
$ sudo mn –mac –switch ovsk –controller remote -x
```

A continuación, establecemos la versión de OpenFlow a utilizar y establecemos el puerto de escucha 6632 para acceder a OVSDB (Open vSwitch Database) en el switch s1 (root):

```
$ ovs-vsctl set Bridge s1 protocols=OpenFlow13
$ ovs-vsctl set-manager ptcp:6632
```

Seguidamente, iniciamos el controlador Ryu y las aplicaciones de QoS y switch de Ryu en el controlador controller c0 (root):

```
$ ryu-manager ryu.app.rest_qos
ryu.app.qos_simple_switch_13 ryu.app.rest_conf_switch
```

Figura 41: Establecimiento de switch que implementa QoS

Queremos establecer el siguiente mecanismo para garantizar un ancho de banda mínimo:

ID de cola	Velocidad máx.	Velocidad mín.
0	500Kbps	-
1	1Mbps	800Kbps

Tabla 11: Ancho de banda a garantizar

Primero de todo establecemos en el controlador c0 (root) la dirección de OVSDB para poder acceder:

```
$ curl -X PUT -d '"tcp:127.0.0.1:6632"'
http://localhost:8080/v1.0/conf/switches/0000000000000
001/ovsdb_addr
```

Y establecemos también los parámetros de la cola:

```
$ curl -X POST -d '{"port_name": "s1-eth1", "type":
"linux-htb", "max_rate": "1000000", "queues":
[{"max_rate": "500000"}, {"min_rate": "800000"}]}'
http://localhost:8080/qos/queue/0000000000000001
```

Para configurar la calidad de servicio instalamos la siguiente entrada de flujo al switch, desde el controlador c0 (root):

Prioridad	Dirección destino	Puerto destino	Protocolo	ID de cola	ID QoS
1	10.0.0.1	5002	UDP	1	1

Tabla 12: Entrada de flujo

```
$ curl -X POST -d '{"match": {"nw_dst": "10.0.0.1",
"nw_proto": "UDP", "tcp_dst": "5002"},
"actions":{"queue": "1"}}'
http://localhost:8080/qos/rules/0000000000000001
```

Para poder comprobar la configuración del switch podemos ejecutar desde el controlador el siguiente comando:

```
$ curl -X GET
http://localhost:8080/qos/rules/0000000000000001
```

De forma que obtenemos el siguiente login:

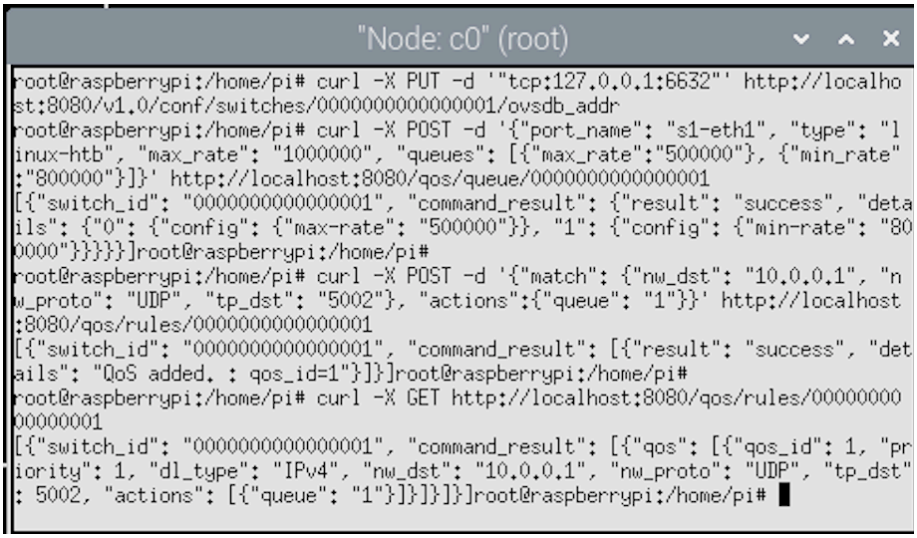


Figura 42: Configuración del switch

Para comprobar que el mecanismo de calidad de servicio funciona correctamente, vamos a medir el ancho de banda utilizando la herramienta *iperf*. El host 1 (servidor) escucha en el puerto 5001 y 5002 con el protocolo UDP. El host 2 (cliente) envía tráfico UDP de 1Mbps por el puerto 5001 y 5002 a h1.

Para ello abrimos un nuevo terminal para h1 y h2:

```
mininet > xterm h1
mininet > xterm h2
```

Y ejecutamos *iperf* en los terminales h1(1) (root), h1(2) root, h2(1) root y h2(2) (root):

```
$ iperf -s -u -i 1 -p 5001

$ iperf -s -u -i 1 -p 5002

$ iperf -c 10.0.0.1 -p 5001 -u -b 1M

$ iperf -c 10.0.0.1 -p 5002 -u -b 1M
```

Las opciones de *iperf* que estamos ejecutando significan que *iperf* actúa en modo servidor (-s), usa UDP en vez de TCP (-u), en intervalos de 1 segundo (-i 1) y escucha por el puerto 5001/5002 (-p 5001) en el caso del host 1.

En el caso del h2, se ejecuta *iperf* en modo cliente (-c), utiliza el puerto 5001/5002 (-p 5001), utilizando UDP (-u) y enviando tráfico de 1Mbps (-b 1M).

Observando ahora la información de las líneas de consola de los hosts tendríamos que observar el correcto funcionamiento de forma que se cumplieran los parámetros establecidos en la tabla 11, por lo que para el puerto 5001 el tráfico garantizado no tendría que superar los 500Kbps y en el puerto 5002 se debería de garantizar 800BKbps de ancho de banda:

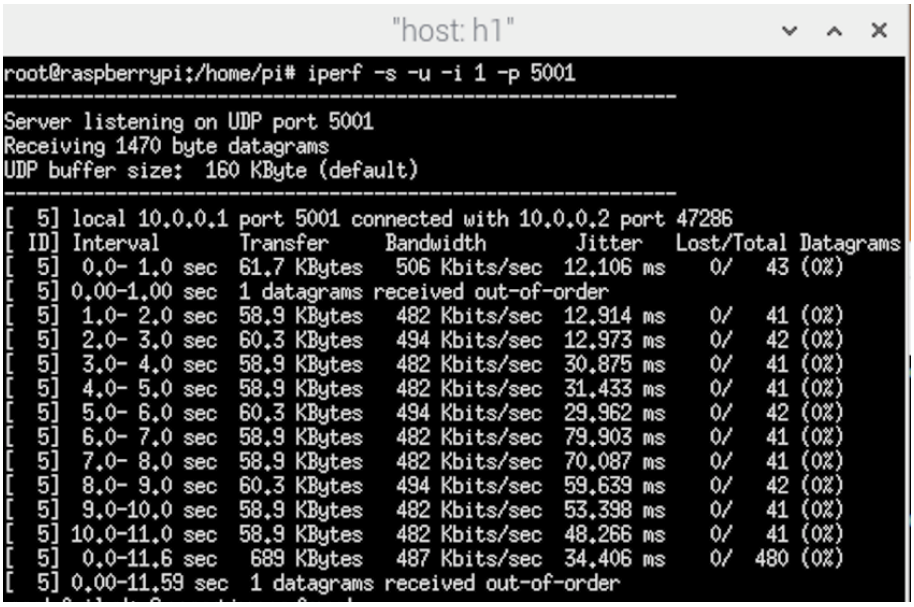


Figura 43: El puerto 5001 muestra que no se superan los 500 Kbps

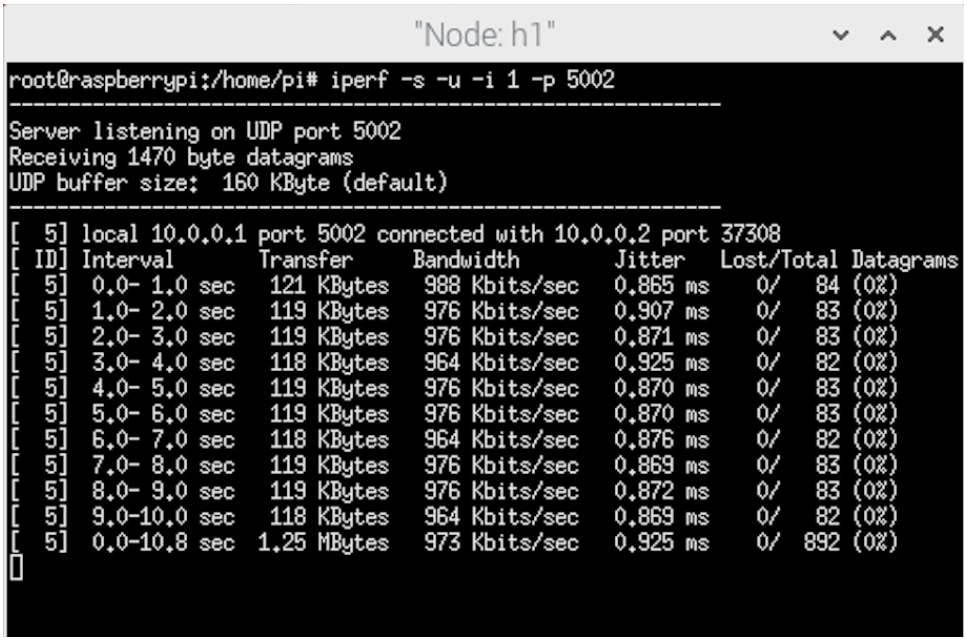


Figura 44: El puerto 5002 tiene garantizados 800 Kbps de ancho de banda

Como se puede observar, el mecanismo de calidad de servicio utilizando funciona perfectamente en redes SDN. Por lo que la utilización de este tipo de arquitecturas puede ser también aplicada a ámbitos donde se requieran cumplir una serie de requisitos para satisfacer a los clientes.

Además, la utilización de SDN ha permitido que mediante el controlador Ryu se pueda configurar cualquier puerto del switch y adaptarlo automáticamente a las necesidades deseadas, sin tener que trabajar directamente con el conmutador.

7.2.4 Monitor de tráfico

Las redes se han convertido en las infraestructuras básicas de muchos servicios y empresas, los cuales requieren de una red estable y que actúe con normalidad. Sin embargo, los problemas siempre están presentes.

Cuando ocurre un fallo en la red, se debe identificar la causa y restaurar la operación lo más rápido posible. Por ello, la detección de errores suele estar muy ligada a comportamientos anormales que ocurren en la red. Para saber si está ocurriendo alguna anomalía, primero de todo se debe conocer perfectamente como se comporta la red de manera regular.

Es por ello por lo que el monitoreo constante del estado de una red es esencial para la operación continua y segura de los servicios o negocios utilizados en la red. El simple hecho de monitorear la información de tráfico no proporciona garantía perfecta, pero en esta sección se describe como utilizar OpenFlow y el controlador Ryu para adquirir información estadística para un switch.

Para ello, debemos ejecutar Mininet creando una topología sencilla de tres hosts y un conmutador y establecer la versión de OpenFlow de la misma manera que se ha realizado en los anteriores ejemplos.

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

Y en el switch s1 (root):

```
$ ovs-vsctl set Bridge s1 protocols=OpenFlow13
```

Seguidamente, ejecutamos el monitor de tráfico y establecemos Ryu como controlador:

```
$ ryu-manager –verbose ryu.app.simple_monitor_13
```

En este momento, la entrada de flujo y el contador de cada puerto es 0 ya que todavía no se ha transferido tráfico. La tabla de flujo al estar vacía no se muestra.

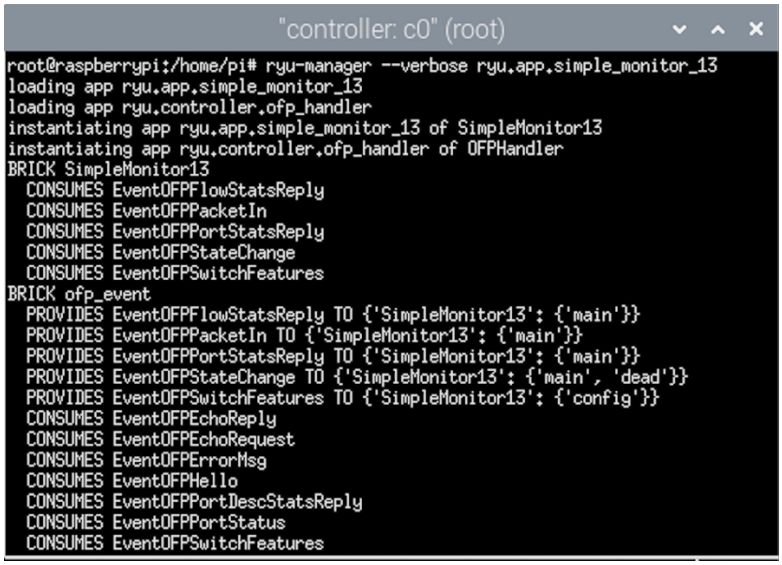


Figura 45: Inicio de Ryu y del monitor de tráfico

Para que se muestre la tabla de flujo, ejecutamos un ping entre el host 1 y el host 2:

```
$ ping -c1 10.0.0.2
```

El paquete se transfiere y es registrado en la tabla de flujo, cambiando la información estadística:

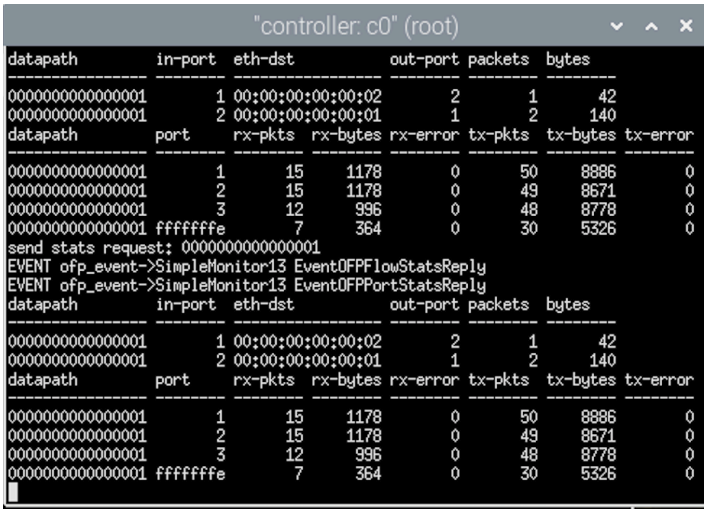


Figura 46: Monitoreo del tráfico en la tabla de flujo

De acuerdo con la información estadística de la tabla de flujo, el tráfico que coincide con el flujo del puerto 1 de recepción se registra como un paquete de 42 bytes. En el puerto de recepción 2 vemos registrados dos paquetes, 140 bytes.

Según la información estadística del puerto, el recuento de paquetes de recepción (rx-pkts) del puerto 1 es de 15 paquetes, y el recuento de bytes en la recepción (rx-bytes) es de 1178 bytes. De la misma forma, podemos ver el recuento de paquetes y de bytes en la recepción en los puertos 2 y 3.

Como se puede observar, las cifras no coinciden entre la información estadística de la entrada de flujo y de puerto. Esto es debido a que la información estadística de la entrada de flujo es la información de los paquetes que coinciden con la entrada y fueron transferidos.

Como hemos podido observar, el monitor de tráfico de Ryu es bastante sencillo, pero nos permite obtener importantes parámetros de tráfico como:

- El número de paquetes y bytes entrantes y salientes, así como el puerto por el que entran o salen.

- Si hay algún error en la transmisión o recepción, cuenta el número de errores que ha habido.

Pese a que no da información de ancho de banda, podría modificarse el script de Ryu que permite utilizar el monitor de tráfico para obtener el tiempo en el que se ha actualizado la tabla de flujo, de forma que obteniendo los bytes transmitidos/recibidos de dos tiempos t1 y t2, podrían restarse los bytes y dividir por (t2-t1) de forma que se obtendría fácilmente el ancho de banda que se ha utilizado.

Por lo cual, en redes SDN, Ryu nos permite realizar mecanismos de monitoreo de tráfico para poder medir diversos parámetros del tráfico de una red.

Capítulo 8. Conclusiones y futuros trabajos

En este proyecto, hemos podido aprender en detalle en qué consiste el Software-defined Networking (SDN) y qué beneficios puede proporcionar a la situación tecnológica que existe actualmente. Hemos visto como las redes actuales se están quedando obsoletas y cada vez les cuesta más soportar la gran y creciente demanda de los servicios de telecomunicaciones actuales, y cómo SDN puede ser una gran herramienta para hacer frente al increíble futuro que viene en los próximos años.

Pese a que es una arquitectura relativamente nueva, los conceptos teóricos están claramente definidos: Así, hemos podido conocer en detalle qué capas y componentes forman parte de la arquitectura SDN, así como la forma de trabajo que tiene el protocolo más extendido, OpenFlow. Sin embargo, el desafío sigue siendo la aplicación práctica de esta arquitectura. Y es que, al ser un protocolo tan joven, todavía faltan muchas horas de investigación, así como procesos de estandarización para que se pueda implementar SDN en la gran mayoría de empresas. Pese a ello ya son más de 100 las empresas que actualmente están utilizando SDN como tecnología y en el futuro, serán muchísimas más. Por ello se anima a las empresas a adoptar estas tecnologías e invertir en el desarrollo e investigación de esta arquitectura.

En el presente documento también se ha hecho hincapié en las grandes ventajas que ofrece trabajar con SDN mediante la simulación práctica de algunas aplicaciones muy utilizadas por las corporaciones, como un Firewall, mecanismos de Calidad de Servicio, el protocolo Spanning Tree y un monitor de tráfico. Para la creación de las topologías de red SDN de estas aplicaciones se ha utilizado el software de código abierto Mininet, que permite crear, configurar y gestionar entornos SDN e incluye multitud de herramientas para poder hacer mediciones o generar tráfico, entre otras. Como controlador SDN se ha elegido un controlador OpenFlow de código abierto llamado Ryu, el cual es muy potente y permite implementar de manera muy simple multitud de aplicaciones, y es que los cuatro ejemplos prácticos que hemos implementado se han ejecutado desde este controlador.

El entorno de trabajo utilizado ha sido una Raspberry-Pi Model 4. Se ha decidido utilizar esta herramienta en vez de una máquina virtual debido a la gran fama que ha obtenido en los últimos años en el mundo de la tecnología, y es que permite tanto a alumnos como a profesionales trabajar de una manera práctica y diferente en multitud de ámbitos de telecomunicaciones o electrónica. Pese a que su uso va muy ligado a la robótica, en este proyecto hemos realizado un ejemplo del uso de una Raspberry-Pi para aplicaciones telemáticas. Así pues, en este trabajo se explica también como configurar desde cero una Raspberry-Pi y cómo instalar paso a paso las herramientas necesarias para poder implementar redes SDN.

Además, debido a que no se requiere un hardware y software específico, se anima a las universidades a implementar prácticas de laboratorio con estas herramientas, ya sea sobre una Raspberry o sobre una máquina virtual, ya que permitirán al alumnado conocer las redes definidas por software, las cuales sin duda serán objeto de estudio en la docencia debido a la gran importancia que obtendrán en la próxima década.

En definitiva, mediante este TFG podemos concluir que hemos cumplido los objetivos previstos, debido a que hemos aprendido detalladamente tanto de forma teórica como práctica en qué consisten las redes SDN y cómo funcionan. De la misma forma hemos aprendido en detalle el funcionamiento del controlador Ryu. Además, la implementación

del proyecto sobre una Raspberry nos ha permitido conocer esta plataforma, qué características tiene y qué nos puede ofrecer.

Como hemos comentado anteriormente, la tecnología SDN es joven y todavía queda mucho por implementarse y desarrollar en el futuro:

- Pese a que en este TFG se ha profundizado en el controlador Ryu, el desarrollo de redes SDN con otros controladores permitiría conocer mejor el protocolo OpenFlow y entender mejor el enfoque que quieren adoptar otras empresas. Por lo cual una posible línea de trabajo futura sería implementar los mismos escenarios de este proyecto, pero con diferentes controladores para así saber que ventajas y desventajas tienen unos frente a otros y conocer mejor el funcionamiento de OpenFlow en general.
- Como se ha comentado, la ventaja del uso de SDN se aprecia más en entornos corporativos cuyas redes constan de decenas de elementos. Sin embargo, en el presente documento se han diseñado topologías simples de uno o dos conmutadores y tres o cuatro hosts. Sería interesante pues implementar las distintas aplicaciones en redes mucho más grandes y complejas para poder ver como se comportaría SDN en una situación semejante a la realidad.
- El estudio de SDN sería muy beneficioso para los estudiantes de Telecomunicación, Telemática o Informática por lo que el desarrollo de guiones de prácticas de laboratorio con el uso de SDN permitiría a los alumnos conocer una de las tecnologías de red más importantes en el futuro, y es que, hoy en día, la docencia acerca de este tema es casi nula.
- Actualmente todavía no hay ningún protocolo estandarizado, ni siquiera OpenFlow, por lo que avanzar hacia una estandarización global permitiría poder trabajar con diferentes fabricantes manteniéndose la compatibilidad.
- Mejorar la seguridad de las redes SDN es una acción vital para asegurar el futuro de esta tecnología. Ya sabemos que la ciberseguridad es un sector cada día más importante para las empresas, por lo que aplicar técnicas de seguridad a las redes SDN para solucionar amenazas como los ataques DoS al controlador debería ser una prioridad.
- La utilización de redes SDN va fuertemente ligada a sistemas operativos como Linux o Ubuntu debido a que los controladores más utilizados están programados en Python. Un desarrollo de SDN aplicado a sistemas operativos como Windows o Macintosh permitiría la expansión más rápida de este protocolo.

Capítulo 9. Bibliografía

[1] Redes definidas por software (SDN). (2020). Recuperado de https://www.cisco.com/c/es_mx/solutions/software-defined-networking/overview.html
[2] Rubio, J. H. (2017, 9 septiembre). ¿Qué es SDN? Recuperado de https://www.ciena.com.mx/insights/what-is/What-is-SDN_es_LA.html+
[3] Capillas Diosdado, M. (2017, 16 mayo). SDN en las redes: el origen de una transformación. Recuperado de https://empresas.blogthinkbig.com/sdn-en-las-redes-el-origen-de-una-transformacion/
[4] Colaboradores de Wikipedia. (2020, 2 abril). Redes definidas por software - Wikipedia, la enciclopedia libre. Recuperado de https://es.wikipedia.org/wiki/Redes_definidas_por_software

[5] Millán Tejedor, R. J. (2016). Qué es... SDN (Software-Defined Networking). Recuperado de https://www.ramonmillan.com/tutoriales/softwaredefinednetworking.php
[6] IONOS España S.L.U. (2019, 12 junio). Software defined network. Recuperado de https://www.ionos.es/digitalguide/servidores/know-how/software-defined-network/
[7] Navarro, M. (2017, 1 junio). ¿Qué aportarán las nuevas redes? Byte, p. 1. Recuperado de https://revistabyte.es/tema-de-portada-byte-ti/aportaran-las-nuevas-redes/
[8] Ramiro, R. (2017, 6 diciembre). Seguridad en las redes definidas por Software (SDN) [Publicación en un blog]. Recuperado de https://ciberseguridad.blog/seguridad-en-las-redes-definidas-por-software-sdn/
[9] IT Digital Media Group. (2018, 20 julio). La mejora de la ciberseguridad impulsa el uso de herramientas SDN. Recuperado de https://www.itdigitalsecurity.es/endpoint/2018/07/la-mejora-de-la-ciberseguridad-impulsa-el-uso-de-herramientas-sdn

[10] Miller Ramírez Giraldo y Ana María López Echeverry (2018, agosto) "Redes de datos definidas por software" https://jci.uniautonoma.edu.co/2018/2018-7.pdf
[11] López, D. (2019, 11 abril). ¿Qué es la virtualización de red y por qué se habla de ello? [Publicación en un blog]. Recuperado de http://blog.orange.es/red/la-virtualizacion-red-se-habla-ello/

[12] Elementos en SDN. (2015) Área de Ingeniería Telemática de la Universidad Pública de Navarra https://www.tlm.unavarra.es/~daniel/docencia/rng/rng15_16/slides/Tema2-05-SDN.pdf

[13] Ryu SDN Framework Community. (2017). RYU SDN Framework. https://osrg.github.io/ryu-book/en/Ryubook.pdf

[14] Carlos Manuel Rodríguez Vergel y Caridad Anías Calderón (2014, enero) "Controladores SDN, elementos para su selección y evaluación" https://www.researchgate.net/publication/320711755_Controladores_SDN_elementos_para_su_seleccion_y_evaluacion

[15] Dean Pemberton, Andy Linton and Sam Russell. (2014) University of Oregon, Network Startup Resource Center. "Ryu OpenFlow Controller" https://nsrc.org/workshops/2014/nznog-sdn/raw-attachment/wiki/WikiStart/Ryu.pdf

[16] Open Networking Foundation (ONF) (2014, junio) https://www.ramonmillan.com/documentos/bibliografia/SDNArchitecture_ONF.pdf

[17] Fraile Herrera 2 septiembre, 2015, R. (2015, 2 septiembre). Casos de uso de SDN: La búsqueda continúa. Recuperado de https://empresas.blogthinkbig.com/casos-de-uso-de-sdn-la-busqueda-continua/
[18] Twain, E. (2018, 20 mayo). SDN Switch Vs. Non-SDN Switch. Recuperado de https://medium.com/@bilby_yang/sdn-switch-vs-non-sdn-switch-49decc32650
[19] Open Networking Foundation. (s.f). [Open Networking Foundation is an operator led consortium leveraging SDN, NFV and Cloud technologies to transform operator networks and business models]. Recuperado de https://www.opennetworking.org

[20] RFC 7426. (s.f) Software-Defined Networking (SDN) https://tools.ietf.org/html/rfc7426

[21] Aleksander Straunik. (s.f). Global Services BT. Bandwidth on demand. https://www.globalservices.bt.com/es/aboutus/news-press/bt-launches-bandwidth-on-demand

[22] Intel Corporation (2014, Mayo) https://www.intel.la/content/dam/www/public/lar/xl/es/documents/articles/adoptingsdnintheent-spa.pdf
[23] Raspberry Pi Foundation. (s.f). [Raspberry Pi 4 Model B – Raspberry Pi]. Recuperado de https://www.raspberrypi.org/products/raspberry-pi-4-model-b/
[24] Knet Solutions. (s.f). [knetsolutions - Sofware Defined Networking (SDN) Training and Academic Projects]. Recuperado de http://knetsolutions.in
[25] Mininet Team. (2020). [Mininet: An Instant Virtual Network on your Laptop (or other PC) - Mininet]. Recuperado de http://mininet.org
[26] Natarajan, S. (s.f). RYU Controller Tutorial | SDN Hub. Recuperado de http://sdnhub.org/tutorials/ryu/
[27] Colaboradores de Wikipedia. (2020, 1 abril). OpenFlow - Wikipedia, la enciclopedia libre. Recuperado de https://es.wikipedia.org/wiki/OpenFlow
[28] Market Watch. (2019, 5 junio). Software Defined Networking (SDN) Market Size to Hit US$ 130 Bn by 2022. Recuperado de https://www.marketwatch.com/press-release/software-defined-networking-sdn-market-size-to-hit-us-130-bn-by-2022-2019-06-05
[29] Cuenca Pérez, G., & Flores Marín, M. (s.f). Redes definidas por software: Solución para servicios portadores del Ecuador. INVESTIGATIO, (6), 41 - 63. https://doi.org/10.31095/irr.v0i6.23
[30] Carlos Lester Dueñas Santos, Yanko Antonio Marín Muro, & Héctor Cruz-Enriquez (2017, Abril) "SDN Network vs. Traditional Network" https://www.researchgate.net/publication/319097382_SDN_Network_vs_Traditional_Network
[31] Felix Álvarez Paliza, & Yanko Antonio Marín Muro. (2016, diciembre) "Plataforma de pruebas para evaluar el desempeño de las redes definidas por software en el protocolo OpenFlow"
https://www.researchgate.net/publication/312187600_PLATAFORMA_DE_PRUEBAS_PARA_EVALUAR_EL_DESEMPENO_DE_LAS_REDES_DEFINIDAS_POR_SOFTWARE_BASADAS_EN_EL_PROTOCOLO_OPENFLOW
[32] David Fernández Cambronero (s.f) "SDN en redes de campus" https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=2ahUKEwiwo7G2msfoAhV9AGMBHSBFD3sQFjAAegQIAhAB&url=http%3A%2F%2Fwww.rediris.es%2Fdifusion%2Feventos%2Fponencias%2F%3Fid%3Dfrc2015-frc-ses-infrII-a5b3c1.pdf&usg=AOvVaw1jSUkFQIohKr5zvUBdaFiL

[33] W. Stallings, F. Agboma, and S. Jelassi, (2015, Octubre) "Foundations of modern networking: SDN, NFV, QoE, IoT, and Cloud." https://learning.oreilly.com/library/view/foundations-of-modern/9780134175478/

[34] Kuzniar, Maciej, Peresini, Peter, and Kostic, Dejan, "What You Need to Know About SDN Flow Tables," in Passive And Active Measurement (Pam 2015), 2015, vol. 8995, pp. 347–359. http://kth.diva-portal.org/smash/get/diva2:785946/FULLTEXT01.pdf

[35] Ramiro Augusto Ríos Paredes, (2016) "Conceptualización de SDN y NFV," Maskay, vol. 6, no. 1, pp. 29–34. https://journal.espe.edu.ec/ojs/index.php/maskay/article/view/163/305

[36] K. Benzekki, A. El Fergougui, and A. Elbelrhiti Elalaoui, (2016) "Software-defined networking (SDN): a survey," Security and Communication Networks, vol. 9, no. 18, pp. 5803–5833. https://onlinelibrary.wiley.com/doi/full/10.1002/sec.1737

[37] Servicios de red para redes definidas por software. (s. f.). Recuperado 13 de abril de 2020, de https://www.ibm.com/es-es/services/network/software-defined