



UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería del Software

Trabajo Fin de Grado

Estudio de despliegue de redes *SDN* utilizando
dispositivos Raspberry Pi

José Andrés Paredes Arribas

Junio, 2023



UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería
del Software

Trabajo Fin de Grado

Estudio de despliegue de redes *SDN* utilizando
dispositivos Raspberry Pi

Autor: José Andrés Paredes Arribas

Tutor: Jaime Galán Jiménez

Tribunal Calificador

Presidente: Lorenzo Martínez Bravo

Secretario: Rafael Martín Espada

Vocal: Mar Ávila Vegas

Índice

RESUMEN	8
1. INTRODUCCIÓN	9
2. OBJETIVOS	11
3. ANTECEDENTES.....	12
3.1 INTRODUCCIÓN	12
3.2 ARQUITECTURA <i>SDN</i>	13
3.2.1 <i>Capa de aplicación</i>	15
3.2.2 <i>Capa de control</i>	15
3.2.3 <i>Capa de infraestructura</i>	16
3.3 COMPONENTES DE UNA RED <i>SDN</i>	17
3.3.1 <i>Aplicación SDN</i>	18
3.3.2 <i>API ascendente</i>	19
3.3.3 <i>Controlador</i>	20
3.3.4 <i>API descendente</i>	24
3.4 VENTAJAS DE USAR REDES <i>SDN</i>	25
3.5 OPENFLOW	27
3.5.1 <i>Conmutador OpenFlow</i>	29
3.5.2 <i>Arquitectura convencional vs Arquitectura OpenFlow</i>	29
4. METODOLOGÍA.....	31
4.1 PLANIFICACIÓN	31
4.2 INSTALACIÓN Y PREPARACIÓN DE LAS HERRAMIENTAS EMPLEADAS	32
4.2.1 <i>Raspberry Pi</i>	32
4.2.2 <i>RealVNC</i>	35
4.2.3 <i>Mininet</i>	37
4.2.4 <i>Controlador POX</i>	43
4.3 TIEMPO INVERTIDO EN EL PROYECTO	46
5. PRUEBAS REALIZADAS Y RESULTADOS	48

5.1 CONEXIÓN ENTRE LAS RASPBERRY PI.....	50
5.2 CORTAFUEGOS	53
5.3 SPANNING TREE PROTOCOL	59
5.4 PROBLEMAS DURANTE EL DESARROLLO DEL PROYECTO.....	65
6. CONCLUSIONES	67
7. BIBLIOGRAFÍA	70
8. GLOSARIO	73
9. ANEXOS.....	74

Índice de figuras

Figura 1: Plano de control y datos Redes SDN.....	13
Figura 2: Capas de la arquitectura SDN	14
Figura 3: Apartados de una red SDN.....	18
Figura 4: API Ascendente	20
Figura 5: API Descendente.....	24
Figura 6: Plano con los apartados de la Raspberry Pi 4	32
Figura 7: Instalador del Sistema Operativo para Raspberry Pi	34
Figura 8: Interfaz VNC Viewer	35
Figura 9: Activación VNC en Raspberry Pi 4	36
Figura 10: Interfaz VNC Server en Raspberry Pi 4	37
Figura 11: Instalación exitosa de Mininet.....	39
Figura 12: Comprobación del correcto funcionamiento de Mininet	40
Figura 13: MiniEdit	41
Figura 14: Logo POX	43
Figura 15: Ejecución de controlador POX eel	45
Figura 16: Error al levantar el controlador.....	45
Figura 17: Gráfico de horas	46
Figura 18: Representación de la conexión de las Raspberry Pi	48
Figura 19: Raspberry Pi utilizadas para el proyecto.....	49
Figura 20: Ubicación del controlador SDN en la topología	50
Figura 21: Ubicación del túnel GRE-TAP en la topología	51
Figura 22: Topología para probar la conexión entre Raspberry.....	52
Figura 23: Topología cortafuegos	54
Figura 24: Ejecución del controlador POX con reglas de Firewall	55
Figura 25: Topología con un cortafuegos entre H11 y H21	55
Figura 26: Captura Wireshark Raspberry1 h11	57
Figura 27: Captura Wireshark Raspberry2 h21	57
Figura 28: Captura Wireshark Raspberry2 h23	58
Figura 29: Captura Wireshark Raspberry2 h23	59
Figura 30: Topología Spanning Tree Protocol	60
Figura 31: Captura del controlador en la prueba STP.....	61

Figura 32: Capturas de los enlaces cortados en la topología 1	62
Figura 33: Capturas de los enlaces cortados en la topología 2	62
Figura 34: Árbol resultante del STP	63
Figura 35: Mensaje indicando cambio en los enlaces.....	64
Figura 36: Captura de los enlaces cortados tras desactivar un enlace	64
Figura 37: Árbol resultante del STP tras la modificación	65

Índice de tablas

Tabla 1: Comparaciones entre controladores SDN.....	23
Tabla 2: Diferencias entre redes SDN y redes tradicionales.....	27

Resumen

Este proyecto pretende desplegar y evaluar una red *SDN* empleando dispositivos Raspberry Pi. Estas redes permiten una mayor flexibilidad y programabilidad en la gestión de la red, lo que implica una mejora en la eficiencia y el rendimiento de la red respecto a las redes convencionales, al mismo tiempo que se reduce la complejidad de la tarea del mantenimiento de la red. Además, se pretende comprobar la efectividad de los mecanismos de Calidad de Servicio (Quality of Services, QoS) y cortafuegos implementados en la red *SDN* utilizando el controlador POX. Para ello, se explicará cómo preparar el entorno en el que se realizarán las pruebas, la instalación de las herramientas empleadas, preparación de las Raspberry Pi 4 y como realizar las diferentes pruebas con POX.

Pero antes de comenzar a hablar acerca de la instalación de los elementos y el modo de empleo del controlador, se procederá a explicar qué son las redes *SDN*, sus características, su funcionamiento y su situación actual respecto a las redes convencionales.

Las Raspberry Pi harán la función de conmutadores OpenFlow y se conectarán entre sí. Se realizaron diversas pruebas para comprobar el funcionamiento de la red, incluyendo la creación de flujos de tráfico utilizando diversas funciones de QoS como, por ejemplo, delimitar la conexión a un ancho de banda específico y la configuración de un cortafuegos para bloquear el tráfico no deseado.

1. Introducción

En los últimos años, las redes han evolucionado a pasos agigantados, y con ellas, la forma en que se gestionan y controlan. Tradicionalmente, las redes han sido construidas mediante dispositivos especializados, como enrutadores y conmutadores, los cuales tienen una lógica de funcionamiento muy específica y compleja ya que los enrutadores convencionales disponen tanto del apartado de los datos como el apartado del control de la red. Esta complejidad puede dificultar la configuración, el mantenimiento y la resolución de problemas en estas redes.

Es aquí donde las redes definidas por software (Software-Defined Networking, *SDN*) entran en juego. Las redes *SDN* están ganando cada vez más importancia debido a su capacidad para simplificar y automatizar la gestión de redes, al tiempo que aumentan su flexibilidad y escalabilidad. Esto se consigue gracias a la separación de la capa de control de la red de la capa de datos, lo que permite que el control de la red sea programable y centralizado.

Con una red *SDN*, es posible crear y aplicar políticas de red de manera más rápida y flexible, lo que puede tener un gran impacto en la eficiencia y el rendimiento de una red. Además, la programabilidad de una red *SDN* permite que las aplicaciones y servicios se adapten y evolucionen de una forma más rápida de lo que se consigue con las redes convencionales, lo que se traduce en una mayor agilidad y capacidad de innovación.

Para este trabajo, se ha querido comprobar las cualidades de este tipo de redes utilizando varias Raspberry Pi 4 a modo de nodos *SDN* para crear una red a pequeña escala utilizando la plataforma Mininet. Mininet es una herramienta muy útil para emular redes y permite probar diferentes soluciones de manera fácil y rápida creando topologías diversas y permitiendo utilizar todas las cualidades que ofrece el protocolo OpenFlow, que es el utilizado por la mayoría de los controladores de redes *SDN*.

En este proyecto se ha optado por utilizar el controlador SDN conocido como POX. Es un software libre de código abierto que implementa el protocolo OpenFlow, que este a su vez se encarga de comunicar el controlador con los dispositivos de la red, permitiendo que las reglas de enrutamiento y otras políticas de red sean programadas y ejecutadas de manera centralizada.

La utilización de Raspberry Pi como si fuesen componentes de una red SDN permite probar diferentes escenarios y operaciones de forma sencilla y con un bajo costo de hardware, ya que los propios componentes como enrutadores o conmutadores *SDN* tienen un elevado coste. Además, la programabilidad de la red permite crear y modificar diferentes políticas de red de manera rápida y eficiente. Esto permite que la red se adapte a las necesidades específicas de cada situación, ya sea en términos de tráfico, seguridad o cualquier otro aspecto de la red.

En este trabajo, se pondrán a prueba las funcionalidades del controlador POX en diferentes escenarios, como por ejemplo la implementación de un cortafuegos entre diferentes dispositivos de la red, políticas de enrutamiento y la gestión del tráfico de red.

2. Objetivos

El objetivo principal que se pretende alcanzar en este proyecto es el de poder utilizar varias Raspberry Pi a modo de nodos en una red *SDN* para poder crear una red a pequeña escala y poder realizar pruebas con esa red. Para conseguir ese objetivo principal, se ha dividido en varios objetivos que enumero a continuación:

1. Analizar la información disponible sobre las redes *SDN*, sus cualidades y diferencias respecto a las redes IP tradicionales, comentar las diversas opciones de controladores disponibles más populares a día de hoy y más concretamente probar las capacidades del controlador *SDN* POX.
2. Analizar la viabilidad sobre la creación de redes *SDN* en dispositivos Raspberry Pi y evaluar las capacidades de este tipo de redes emulando redes a pequeña escala dentro de estos dispositivos e implementar varias aplicaciones utilizando el controlador POX.
3. Crear topologías de red para probar diferentes casos o funcionalidades del controlador POX mediante el uso de la herramienta Mininet que permitan conectar varias Raspberry Pi entre sí y así comprobar la conectividad entre las Raspberry Pi, garantizando que los dispositivos puedan comunicarse de manera eficiente siguiendo protocolos OpenFlow y realizar varias pruebas con las redes creadas mediante Mininet como son: Configurar un cortafuegos para bloquear el tráfico no deseado o evitar que se generen bucles en la red para evitar la pérdida de paquetes de datos que viajen por la red.
4. Documentar el proceso de implementación y las pruebas realizadas en un informe técnico que se pueda utilizar como referencia para futuros proyectos de redes *SDN* con Raspberry.

3. Antecedentes

3.1 Introducción

Las Redes Definidas por Software son una forma de manejar las redes mediante el uso de controladores por software o mediante interfaces de programación de aplicaciones (Application Programming Interface, *API*) conectado al hardware que compone la red para así dirigir el tráfico de la red.

Son un enfoque moderno en el diseño de redes que busca mejorar la flexibilidad, la escalabilidad y la capacidad de programación de la red. En una red *SDN*, el control de la red se centraliza en un controlador de red, mientras que el reenvío de paquetes se realiza en dispositivos de red físicos en una capa separada del plano de control.

La centralización del control en una red *SDN* permite a los administradores de redes tener una vista completa y unificada de toda la red, lo que les permite tomar decisiones más informadas y amoldar la configuración de la red en tiempo real en función de las necesidades que requiera la red en un momento determinado. Además, la capacidad de programar la lógica de red en un controlador de red centralizado permite una mayor automatización y personalización de la red.

La arquitectura de una red *SDN* se divide en tres componentes principales: el plano de control, el plano de datos y el plano de administración. El plano de control es el componente central que se encarga de la toma de decisiones de reenvío de paquetes y de la programación de la lógica de red. El plano de datos es el componente que se encarga del reenvío físico de paquetes. El plano de administración se encarga de la configuración, monitorización y mantenimiento de la red. En una red *SDN*, los dispositivos de red físicos, como los conmutadores y enrutadores, actúan como dispositivos de reenvío, que simplemente reenvían los paquetes a través de la red siguiendo las decisiones de reenvío tomadas por el plano de control centralizado. El plano de control se comunica con los dispositivos de reenvío

a través de un protocolo estandarizado llamado OpenFlow, que permite al controlador de red programar las tablas de reenvío de los dispositivos de reenvío.

3.2 Arquitectura SDN

La arquitectura SDN se basa en separar el plano de control del plano de datos de la red, permitiendo una mayor flexibilidad a la hora de programar y gestionar las redes. Esta arquitectura se divide en tres partes principales que son el *plano de gestión*, *plano de control* y *plano de datos*.

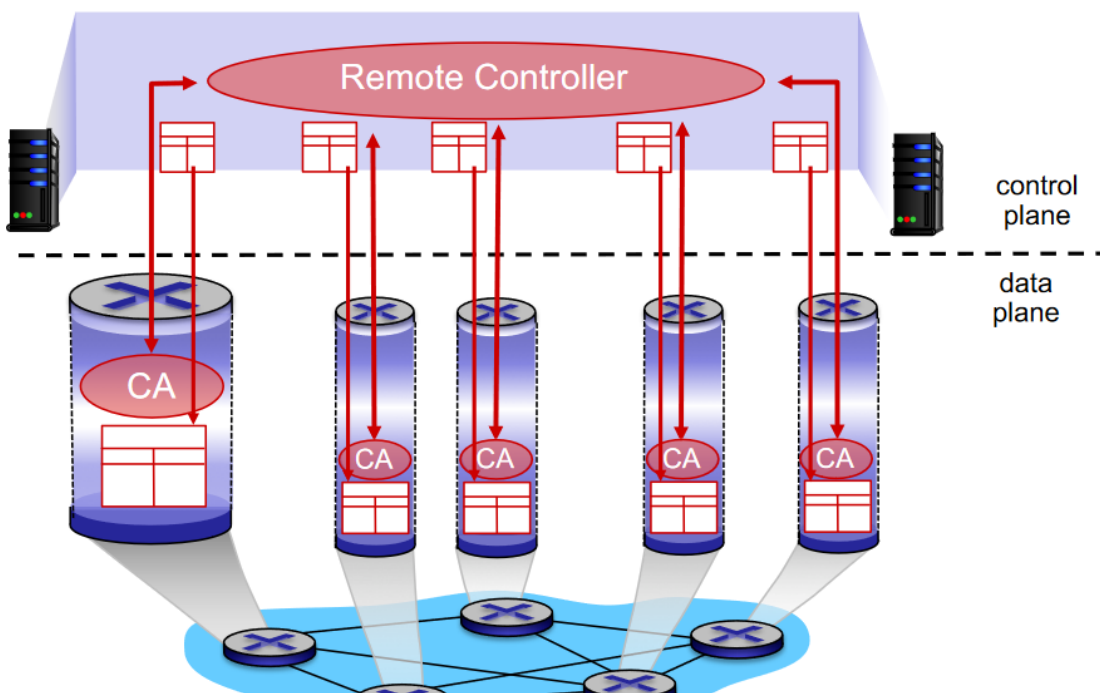


Figura 1: Plano de control y datos Redes SDN

Fuente: Redes de computadoras. Un enfoque descendente, 7 Ed [13]

El **plano de gestión** es la interfaz de administración de la red que permite a los administradores de red configurar y monitorizar el tráfico de la red SDN. El plano de gestión se implementa en el controlador SDN y se accede a través de una *API* ascendente y con ella se puede desarrollar una herramienta que cumpla con las necesidades de los administradores de la red.

El **plano de control** es el componente central de la red *SDN*, encargada de tomar las decisiones de envío de paquetes y es la parte de la red que maneja la lógica de control de la red. En esta, se implementa un controlador centralizado que se comunica con los dispositivos de red a través de una *API* descendente y, a través del protocolo OpenFlow, programa las tablas de direccionamiento de los dispositivos de la red.

El **plano de datos** es la parte de la red que maneja el tráfico de red real y está compuesta por los dispositivos de red, como conmutadores y enrutadores. En una red *SDN*, los dispositivos de red están programados por el controlador a través de la *API* descendente y siguen las decisiones tomadas por el plano de control centralizado.

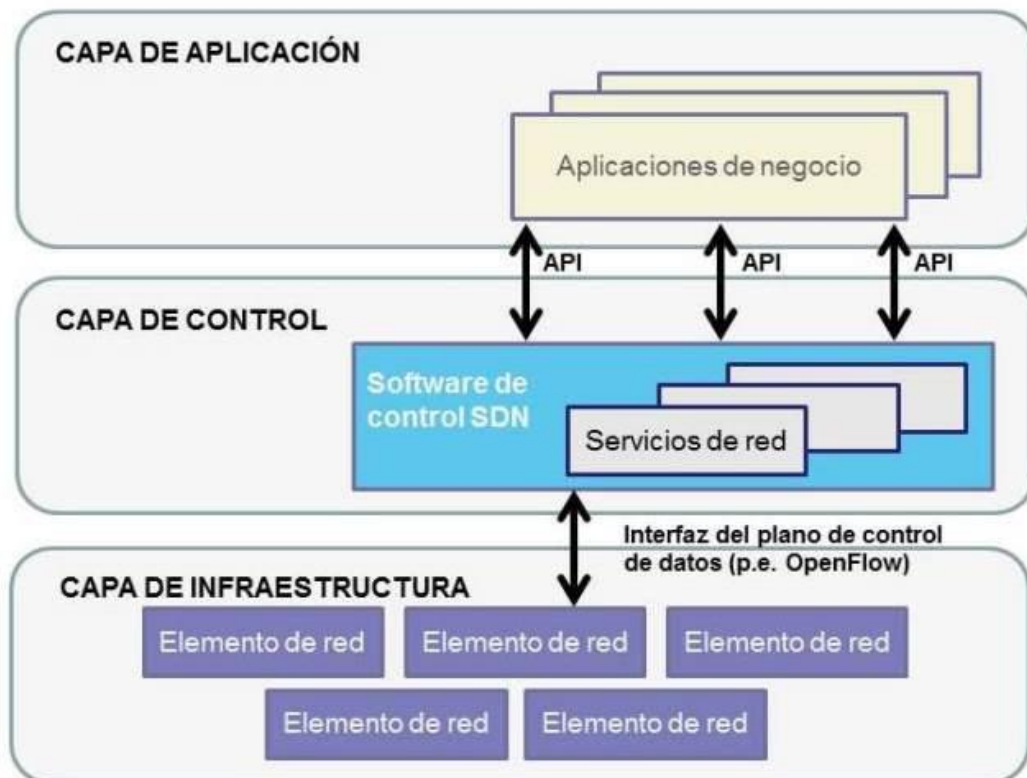


Figura 2: Capas de la arquitectura SDN

Fuente: "SDN: el futuro de las redes inteligentes" [1]

3.2.1 Capa de aplicación

Esta es la capa superior de la pila de protocolos de red donde se implementan las aplicaciones que se encargan de utilizar y gestionar la red. Se encarga de proporcionar servicios y aplicaciones específicas para satisfacer las necesidades de los usuarios finales.

La capa de aplicación se comunica con el controlador a través de una interfaz de programación de aplicaciones estandarizada. Esto permite a los desarrolladores de aplicaciones personalizar y automatizar la gestión de las redes para sus necesidades específicas.

Las aplicaciones que se pueden implementar en el nivel de aplicación de *SDN* incluyen seguridad de red, gestión de ancho de banda para controlar las congestiones en la red, automatización de tareas de gestión de la red, configuración, mantenimiento y análisis de tráfico.

3.2.2 Capa de control

Es la capa intermedia de la pila de protocolos de red y es la responsable de la gestión y control de la red. En este nivel, se encuentra el controlador de red, que es el componente clave de la arquitectura *SDN*.

El controlador de red es un software programable que se ejecuta en un servidor y actúa como el cerebro de la red *SDN*, permitiendo que haya una gestión centralizada. Es el responsable de recibir y procesar los mensajes de los dispositivos de la red, como enrutadores y conmutadores, y de enviar instrucciones a los dispositivos para configurar la red de acuerdo con las políticas definidas por el usuario.

También es responsable de detectar los dispositivos de red en la capa de infraestructura de la red. Para hacer esto, utiliza protocolos de descubrimiento de dispositivos, como OpenFlow, para comunicarse con los dispositivos de red y recopilar información sobre sus capacidades y estado.

Proporciona una vista global de la red y es capaz de tomar decisiones sobre el enrutamiento de manera centralizada, en lugar de delegar estas

decisiones a los conmutadores de red individuales. Esto permite una mayor flexibilidad y control sobre la red, ya que las políticas de red pueden ser definidas y modificadas de manera más rápida y sencilla en el controlador de red en lugar de tener que modificar todos los componentes de la red uno a uno para adecuarse a las necesidades de la red.

Además, el controlador de red puede proporcionar una interfaz programable a las aplicaciones de nivel superior, lo que permite que las aplicaciones controlen directamente la configuración de la red a través del controlador.

3.2.3 Capa de infraestructura

La capa de infraestructura, en la arquitectura *SDN*, hace referencia a la capa inferior de la pila de protocolos de red compuesta por los dispositivos físicos de red, como son los conmutadores, enrutadores y otros dispositivos de red, que proporcionan conectividad física para el transporte de datos y control de la red.

Los dispositivos de red son responsables de la recepción y transmisión de paquetes de datos a través de la red. Por lo que los dispositivos deben ser capaces de analizar los paquetes entrantes y seguir las pautas de enrutamiento marcadas por el controlador en la capa superior. También pueden ser configurados para priorizar, por ejemplo, el tráfico de vídeo sobre otros tipos de tráfico en la red.

En las redes *SDN*, los dispositivos de red en la capa de infraestructura son diferentes de los dispositivos de red tradicionales. Estos dispositivos, llamados conmutadores OpenFlow, son capaces de comunicarse con el controlador de red a través de un protocolo de comunicación entre el controlador y los dispositivos de red llamado OpenFlow. Mediante este protocolo, proporcionan al controlador información acerca del ancho de banda, el rendimiento de red y otros aspectos que sean importantes de la red.

Además, los dispositivos de red pueden ser programados para cambiar su comportamiento según las políticas definidas por el controlador de red. Esto

significa que los dispositivos de red pueden modificarse de manera dinámica y adaptarse a los cambios en la red de manera más rápida y sencilla que en las redes tradicionales.

3.3 Componentes de una Red *SDN*

Procedo a explicar brevemente los diferentes componentes de una red *SDN*, de los que luego hablaré más en profundidad.

1. Aplicación *SDN*: Son una amplia variedad de herramientas que, mediante las *API* mencionadas anteriormente, permiten a los usuarios comunicarse con el controlador para marcarle las pautas que debe seguir la red y a su vez el controlador proporcionarle toda la información disponible de la red.
2. Interfaces de programación de aplicaciones: las *API* posibilitan la interacción entre el controlador y las aplicaciones que se ejecutan en la red. Las aplicaciones pueden utilizar las *API* para programar la red y gestionarla de manera centralizada.
3. Controlador: el controlador es el componente central de una red *SDN* y es responsable de la gestión centralizada y la programación de la red. El controlador se comunica con los dispositivos de red a través del protocolo OpenFlow y es responsable de tomar decisiones sobre cómo se enrutan los paquetes de datos a través de la red.
4. Conmutadores *SDN*: A diferencia de los conmutadores convencionales, estos toman decisiones de enrutamiento de forma independiente pero siempre siguiendo las pautas indicadas por el controlador mediante un protocolo de comunicación estándar como es, por ejemplo, OpenFlow. De los conmutadores *SDN* hablaré en más profundidad en el apartado de OpenFlow.

Ahora, se procede a explicar los distintos componentes siguiendo el orden mostrado en la Figura 3 comenzado desde el nivel de aplicación y terminando con la explicación de OpenFlow y los conmutadores *SDN*.

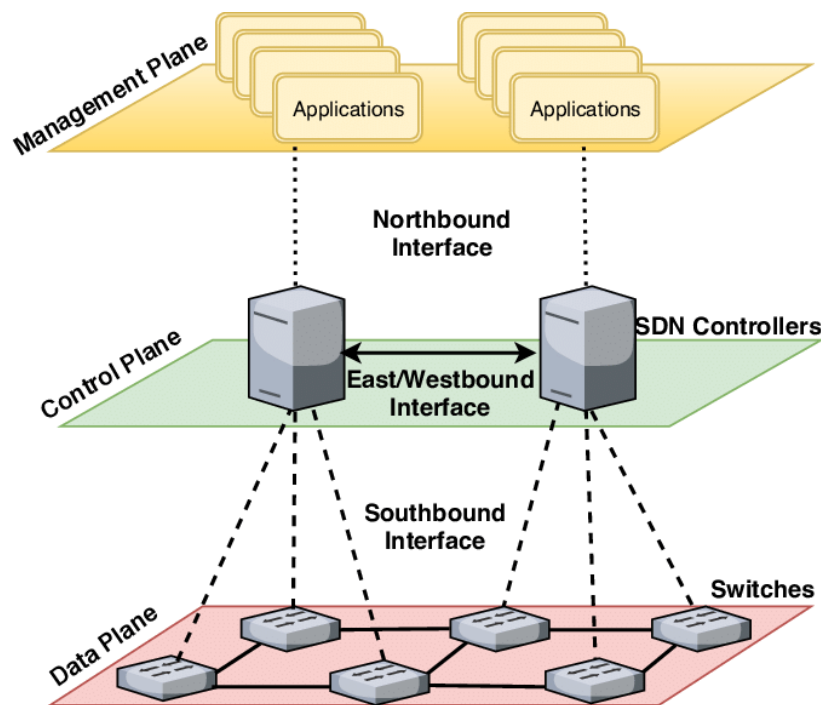


Figura 3: Apartados de una red SDN

Fuente: "A Comprehensive Survey of Interface Protocols for Software Defined Networks" [2]

3.3.1 Aplicación SDN

Son programas informáticos que se ejecutan en la capa de aplicación y utilizan la *API* ascendente para interactuar con el controlador de red y poder manipular la red de una forma que permite que resulte más sencillo programar, gestionar la red y recabar información sobre las capas inferiores. Pueden ser desarrolladas por proveedores de red, terceros o incluso por los propios administradores de red, lo que les da la capacidad de personalizar la red para adaptarse a sus necesidades específicas.

Una única aplicación SDN puede llamar a otros servicios externos y organizar cualquier número de controladores SDN. Estas aplicaciones pueden actuar de varias formas:

- Un servidor que proporciona información acerca del estado de la red. Se dedica a obtener la información de la red como, por ejemplo, el ancho de banda empleado, si hay congestión en algún punto de la red, etc. para así poder enviar esa información a otras aplicaciones que desempeñen la labor de gestora de las redes.
- Una aplicación que se encargue de administrar las políticas de gestión de la red. Utilizando la información obtenida de algún servidor que proporcione información acerca del estado de la red, esta aplicación permite manipular la red enviando las políticas de enrutamiento, el uso de los recursos, políticas de seguridad que debe implementar el controlador en la red.
- Hay algunas aplicaciones que realizan los dos apartados anteriores a la vez.

3.3.2 *API* ascendente

La *API* ascendente, también denominada *Northbound API*, es un conjunto de interfaces de programación de aplicaciones que permiten que las aplicaciones de la capa de aplicación se comuniquen con la capa de control. Permite que las aplicaciones externas como, por ejemplo, el software de gestión de red; herramientas de análisis de tráfico; etc. se comuniquen con el controlador de red en el nivel de controlador de *SDN*. De esta forma, las aplicaciones pueden acceder a la información sobre la red y utilizarla para implementar políticas y servicios específicos de la red.

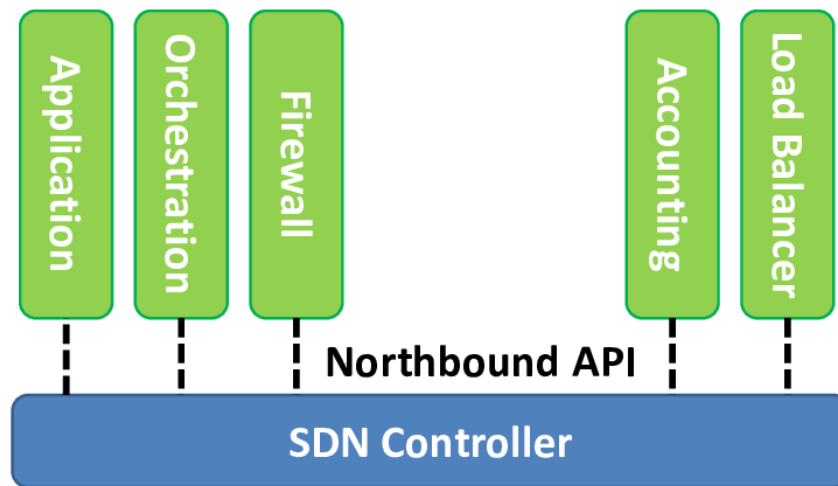


Figura 4: API Ascendente

Fuente: “Programmable Networks – From Software Defined Radio to Software Defined Networking” [3]

Las API ascendentes suelen ser diseñadas con la intención de ser independientes de un controlador de red concreto, permitiendo que cualquier aplicación ajena al controlador específico que se esté utilizando se puedan comunicar con una alta variedad de controladores de red mediante algunas de las API más utilizadas como son *REST* (Representational State Transfer), *NETCONF* (Network Configuration Protocol) o *Java API*.

3.3.3 Controlador

El controlador es el cerebro de la red. Maneja y controla el tráfico de la red de manera centralizada desde la capa intermedia de las redes SDN. Interactúa con los dispositivos de red, como conmutadores y enrutadores, a través de la API descendente, permitiendo la configuración y el control de la red de una manera más cómoda, sencilla y permitiendo que sea más sencilla la programación de la misma.

Las tareas fundamentales que desempeña un controlador son la obtención de información acerca del estado de la red, la toma de decisiones acerca de cómo se deben enrutar los paquetes, la configuración de los diferentes dispositivos de la red y su descubrimiento. Algunos controladores emplean

herramientas como bases de datos locales o un coordinador que permita manejar las acciones de varios controladores en redes de mayor tamaño. También dispone de interfaces para hacer más cómoda la integración de software de terceros o la propia implementación de la aplicación desarrollada para controlar la red.

A día de hoy, ha aumentado el número de controladores *SDN* disponibles. Cada uno tiene sus propias características. Algunos de los más extendidos y de código abierto en el mercado son los siguientes:

NOX: Fue el primer controlador de OpenFlow que se desarrolló en el mercado. Como lenguaje solo dispone de versión en C++ y su versión base solo trabaja con la versión de OpenFlow 1.0, pero puede llegar a manejar las versiones posteriores de OpenFlow hasta la 1.3. [4]

POX: Está desarrollado con Python. Originalmente funcionaba únicamente como un controlador OpenFlow, pero ahora puede utilizarse también como un conmutador de OpenFlow y para desarrollar software. Trabaja con la versión OpenFlow 1.0 y tiene soporte para extensiones de Nicira y vSwitch. Puede funcionar en muchas plataformas, pero en la que dispone de más capacidades es en los sistemas operativos con base Linux. [5]

Ryu: Está desarrollado con Python. Es un software libre que emplea Apache para funcionar y utiliza las licencias de Apache 2.0. Tiene una *API* bien definida que permite a los desarrolladores crear de forma sencilla aplicaciones de gestión y control de redes. Dispone de varios protocolos para gestionar dispositivos de red, como OpenFlow, desde su versión 1.0 hasta la 1.5 y algunas extensiones, Netconf y OF-config entre otros. [6]

Floodlight: Creado por una comunidad en la que la mayoría de los desarrolladores provienen de Big Switch Networking (una empresa dedicada a servicios y datos almacenados en la nube) y emplea OpenFlow para controlar el tráfico en la red *SDN*. Dispone de dos entornos gráficos, uno web y otro de Java. Aunque ahora el proyecto se encuentra desactualizado, teniendo su última actualización en mayo de 2021. [7]

Beacon: Es un controlador de código abierto realizado en Java. Es el que se utilizó como base para la creación de Floodlight y OpenDayLight y también para la investigación y la enseñanza de las redes *SDN*. Destaca por ser multiplataforma y emplear una estructura, conocida como Iniciativa de Puerta de Enlace de Servicios Abiertos (Open Services Gateway Initiative, *OSGI*), que permite configurar las aplicaciones sin necesidad de desconectar los conmutadores de la propia red. [8]

Cherry: Es un controlador OpenFlow escrito con Go y soporta las versiones de OpenFlow 1.0 y 1.3. No está centrado en que cualquier usuario pueda utilizarlo, sino más bien para un proveedor de servicios técnicos para *SDN*. [9]

OpenDayLight: Es un controlador creado por la Fundación Linux en 2013. Está escrito en Java, ya que junto con Floodlight, este controlador deriva de Beacon. Dispone de una *API REST* y dispone de Interfaz de usuario para las versiones de Oxygen (ODL v8) y anteriores. Soporta una gran cantidad de protocolos de red como por ejemplo OpenFlow como *API* descendente. [10]

En la Tabla 1 se encuentran resumidas las características que diferencian a estos controladores:

	Soporte OpenFlow	Lenguaje	Interfaz Gráfica	Plataformas soportadas	Código abierto	API REST
NOX	1.0	C++, Python	No	Linux	Sí	Solo Python
POX	1.0	Python	No	Linux, Windows, Mac	Sí	Sí
RYU	1.0 - 1.5	Python	Sí	Linux	Sí	Sí
Floodlight	1.0 - 1.4	Java	Web	Linux, Windows, Mac	Sí	Sí
Beacon	1.0	Java	Web	Linux, Windows, Mac, Android	Sí	Sí
Cherry	1.0, 1.3	Go	No	Docker	Sí	Sí
ODL	1.0, 1.3.2	Java	Hasta Oxygen	Linux, Windows, Mac	Sí	Sí

Tabla 1: Comparaciones entre controladores SDN

Según las herramientas que se vayan a emplear, la empresa o el tamaño de la red que se debe gestionar es recomendable un controlador u otro. NOX tiene una curva de aprendizaje bastante elevada y a día de hoy se encuentra en desuso, es recomendable comenzar con POX, que tiene una mayor comunidad de desarrolladores, lo que permite poder encontrar bastante información por internet y es bastante flexible a la hora de utilizarlo. Ryu tiene bastantes similitudes con el controlador POX en lo que respecta a flexibilidad, pero este dispone de mayor compatibilidad con diferentes protocolos más allá de OpenFlow. Floodlight tiene una gran comunidad detrás, cuenta con bastantes recursos online y es recomendable para grandes proyectos. Beacon se encuentra en desuso y no tiene apoyo online. Cherry tiene un diseño modular que puede ayudar a la hora de crear proyectos pequeños o medianos. OpenDayLight tiene muchas opciones,

para alguien que comienza en las tecnologías SDN puede resultar muy compleja, sobre todo si se utilizan las últimas versiones, que no dispone de una interfaz visual que permita realizar modificaciones en la red sin requerir de utilizar comandos. Personalmente recomiendo POX debido a su sencillez de uso y en caso de utilizar la herramienta Mininet ya viene integrado en su funcionamiento, por lo que no requiere hacer muchos cambios una vez instalas Mininet.

3.3.4 API descendente

La API descendente, también conocida como Southbound API, es el conjunto de interfaces que permiten que la capa de control en la que se encuentra el controlador de red se comuniquen con los dispositivos de red en la capa de infraestructura. Es la que posibilita que el controlador de red envíe las instrucciones de configuración y enrutamiento de paquetes adecuadas a los dispositivos de red en la capa de infraestructura, como conmutadores y enrutadores, para implementar políticas de red definidas por el usuario y todo esto de forma dinámica, en lugar de delegar esas tareas a los dispositivos de la red individuales como se hace en las redes convencionales.

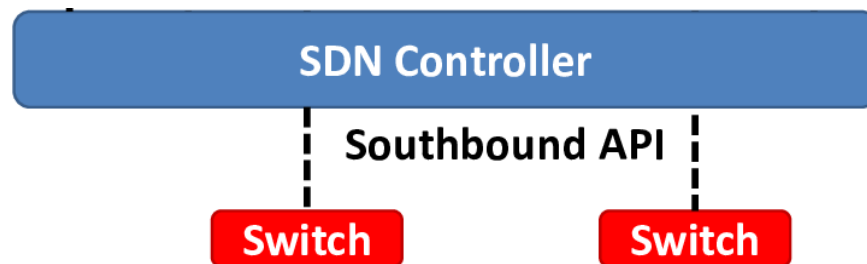


Figura 5: API Descendente

Fuente: "Programmable Networks – From Software Defined Radio to Software Defined Networking" [3]

Estos pueden variar según el protocolo de comunicación que se emplee entre el controlador y los dispositivos de la red. Uno de los protocolos de comunicación más habituales utilizados por las API descendentes en la

arquitectura *SDN* es OpenFlow, que permite que el controlador de red configure y controle la red enviando mensajes a los conmutadores OpenFlow. Otros protocolos que se utilizan para implementar la *API* descendente son *OVSDB* (Open vSwitch Database Management Protocol) y *NETCONF*.

3.4 Ventajas de usar redes *SDN*

Algunas de las ventajas a destacar que podemos encontrar al utilizar las redes *SDN* respecto a usar redes convencionales son las siguientes:

- **Arquitectura:** A diferencia de los controladores convencionales, que suelen tener una arquitectura basada en la distribución de la información, los controladores *SDN* disponen de una arquitectura centralizada, lo que facilita su gestión.
- **Mayor disponibilidad de las redes:** Permite ayudar a los administradores de las redes con las actualizaciones de la red, con la recuperación de la red en caso de que se produzca alguna caída o en caso de que se necesite modificar la red para poder cubrir las necesidades del negocio.
- **Reduce los costes:** Gracias a la gestión mediante software de las redes, no se requiere comprar tantos componentes que sean muy caros y permite utilizar con mayor eficiencia y eficacia los recursos de los que se dispone. Y al reducirse la complejidad que conlleva la administración y gestión de las redes, también se reducen tanto el gasto general causado por el mantenimiento como el coste de personal.
- **Personalización:** Las empresas pueden crear sus propias aplicaciones y herramientas personalizadas para conseguir mejorar el uso y gestión de sus redes y disponer de una interfaz propia para el control que se adecúe a sus necesidades. Esto no es posible con los controladores convencionales.

- **Control del tráfico:** Permite redirigir la información dentro de la red configurando las rutas que deben seguir los datos entre los distintos componentes de la red, administrar de forma dinámica el ancho de banda según las necesidades de cada recurso, poder administrar todos los recursos relacionados con la seguridad...
- **Generar analíticas:** Al disponer de una gestión centralizada, se puede obtener mucha información sobre la red y generar informes acerca de: quién accede a qué sitio, durante cuánto tiempo lo hace, con qué dispositivo se ha realizado la conexión, el tráfico que pueda llegar a tener una zona de la red...
- **Estandarización:** Al utilizar protocolos estándar, facilita las operaciones entre diferentes dispositivos y sistemas de red.
- **Separación entre control y datos:** La lógica de control al centralizarse en un controlador, no es necesario que se encuentre en cada dispositivo de la red.

En esta tabla se encuentran de forma resumida algunos de los puntos explicados anteriormente:

	Redes SDN	Redes tradicionales
Gestión	La gestión se encuentra centralizada	El control se encuentra distribuido
Visibilidad	Mayor visibilidad y control del tráfico de la red	Hay limitaciones en la visibilidad y el control del tráfico
Código abierto	Los protocolos disponen de código abierto	Los protocolos dependen de los fabricantes de los componentes
Escalabilidad	Mejor escalabilidad y capacidad de gestión de la red	Limitación en el número de dispositivos y cuanto mayor es la red más compleja es su gestión
Configuración	La configuración de los dispositivos de la red se puede realizar de forma automática	Requiere configurar manualmente los dispositivos de la red

Tabla 2: Diferencias entre redes SDN y redes tradicionales

3.5 OpenFlow

OpenFlow es un protocolo de comunicación de red que permite a los controladores SDN controlar el flujo de datos de la red. Es un protocolo abierto y estándar que permite que se programen las redes de una forma centralizada. Fue desarrollado por la Open Networking Foundation (ONF) como una iniciativa para crear un enfoque más flexible y programable para la gestión de redes.

Se utiliza para conectar el controlador SDN de la capa de control con los dispositivos de la capa de infraestructura, permitiendo que haya un nivel de

abstracción entre el control de la red y la propia red. A día de hoy, OpenFlow es de los protocolos más populares en redes *SDN* y es compatible con muchos hardware y softwares de red.

Funciona estableciendo un canal de comunicación entre el controlador *SDN* y los dispositivos de red. Este canal se utiliza para enviar y recibir mensajes que permiten al controlador tomar el control de las acciones de los dispositivos de red. El protocolo define un conjunto de reglas y acciones que se pueden aplicar a los paquetes de red en el plano de datos de la red. Estas reglas se pueden programar y cambiar dinámicamente para adaptarse a las necesidades de la red en un momento determinado.

A lo largo de los años, han ido saliendo varias versiones diferentes de OpenFlow, cada una añadiendo mejoras respecto a la anterior.

OpenFlow 1.0: Esta fue la primera versión de OpenFlow lanzada en 2008. Esta versión permitía la separación de la capa de control y la capa de datos, lo que permitía una mayor flexibilidad y control sobre la red.

1. OpenFlow 1.1: Lanzada en 2011 y agregó nuevas características como el soporte para múltiples tablas de flujo y una mayor capacidad para manejar paquetes de red.
2. OpenFlow 1.2: Lanzada en 2011, esta versión agregó soporte para grupos de acciones, lo que permitió a los controladores *SDN* manejar acciones de flujo en conjunto y ejecutarlas simultáneamente.
3. OpenFlow 1.3: Lanzada en 2012, añadió soporte para múltiples flujos en un solo paquete, lo que mejoró la eficiencia del procesamiento de paquetes.
4. OpenFlow 1.4: Lanzada en 2013, esta versión agregó soporte para la comunicación bidireccional entre los dispositivos de red y los controladores *SDN*, lo que permitió a los dispositivos de red enviar mensajes al controlador.

5. OpenFlow 1.5: Esta versión se lanzó en 2014 y agregó soporte para tablas de flujo de mayor tamaño y soporte mejorado para manejar paquetes IPv6.
6. OpenFlow 1.6: Lanzada en 2015, esta versión agregó soporte para la redirección de paquetes y el procesamiento de paquetes de manera más eficiente.
7. OpenFlow 1.7: Esta versión se lanzó en 2017 y agregó soporte para el control de acceso de flujo, lo que permitió a los controladores *SDN* restringir el acceso a flujos específicos de la red.

3.5.1 Conmutador OpenFlow

Un conmutador OpenFlow está diseñado para funcionar con el protocolo OpenFlow. Este protocolo permite que un controlador *SDN* controle el comportamiento de los conmutadores de red mediante la separación del plano de control del plano de datos.

En una arquitectura OpenFlow, los conmutadores se dividen en dos planos: el plano de control y el plano de datos. El plano de control es donde se ejecuta el software que implementa la lógica de control de la red. El plano de datos es donde se procesan los paquetes de datos que se transmiten a través de la red. El controlador puede programar los conmutadores OpenFlow para que realicen determinadas acciones en función de las políticas de la red. Por ejemplo, el controlador puede programar al conmutador para enrutar el tráfico de una manera específica, aplicar políticas de QoS o proporcionar seguridad mediante la aplicación de reglas de cortafuegos.

Los conmutadores OpenFlow son flexibles y programables, lo que les permite adaptarse a las necesidades específicas de la red.

3.5.2 Arquitectura convencional vs Arquitectura OpenFlow

La principal diferencia entre una arquitectura convencional de red y una arquitectura OpenFlow se basa en la forma en que se maneja el tráfico de la

red. En una arquitectura convencional, el manejo del tráfico se realiza en los dispositivos de red como conmutadores y enrutadores, mientras que en una arquitectura OpenFlow, el control del tráfico se encuentra centralizado.

En una arquitectura convencional, los conmutadores y enrutadores de la red son independientes y realizan la función de encaminamiento y control de tráfico en función de las reglas y políticas configuradas localmente en cada dispositivo. Esto puede dar lugar a una configuración compleja de la red, lo que dificulta la gestión y el control.

En una arquitectura OpenFlow, el controlador de red se encarga de manejar todo el tráfico de la red. Los conmutadores de la red, en lugar de tomar decisiones de manera autónoma, envían paquetes al controlador para su procesamiento y para determinar cómo deben ser encaminados. El controlador de red puede aplicar políticas de seguridad, QoS y enrutamiento de manera centralizada, lo que simplifica la configuración y el control de la red.

Otra diferencia importante es que la arquitectura OpenFlow es programable y puede ser adaptada a las necesidades específicas de la red. Esto significa que los desarrolladores pueden crear y personalizar aplicaciones para mejorar la gestión y el control de la red, lo que no es posible en una arquitectura convencional.

4. Metodología

4.1 Planificación

Para la realización de este proyecto, primero se han instalado y preparado las herramientas necesarias para ejecutar las redes *SDN* en los dispositivos Raspberry como se indica en [4.2 Instalación y preparación de las herramientas empleadas.](#)

La metodología que se ha empleado para este proyecto se basa en el ensayo y error. Se han planteado una gran variedad de pruebas o ideas a lo largo del desarrollo del proyecto y se han ido descartando o modificando hasta conseguir resultados óptimos y un correcto funcionamiento de lo que se pretendía realizar en este proyecto. Esta metodología ha permitido obtener un mayor conocimiento acerca del funcionamiento de la mayoría de los componentes y variables que rodean este proyecto desde las herramientas instaladas como las funciones internas del S.O., así como sus limitaciones a la hora de realizar algunas operaciones debido a que alguna herramienta no disponía de todas sus capacidades al no estar preparada para las pruebas que se pretendían realizar.

Una vez se han completado las instalaciones y habiéndose comprobado que todas las aplicaciones funcionan correctamente tras la configuración, se ha procedido a plantear una serie de pruebas para comprobar que la red funciona correctamente. Esta tarea se ha tenido que realizar teniendo en cuenta las limitaciones de una de las herramientas al estar preparada para funcionar en un único dispositivo en lugar de en una topología fraccionada en varias máquinas como se intenta comprobar en este proyecto.

Tras haber decidido cuáles serán las funcionalidades que se pondrán a prueba, se ha diseñado una batería de pruebas consistentes en crear diferentes topologías para llevar a cabo dichas pruebas. En la documentación se mostrará solo un entorno diferente para cada una de las

pruebas realizadas explicando lo que se pretende probar, su implementación, su ejecución y los resultados obtenidos.

4.2 Instalación y preparación de las herramientas empleadas

Para la realización de este proyecto que trata de crear redes *SDN* utilizando dispositivos Raspberry Pi 4, se han utilizado diversas herramientas necesarias para poder realizar las pruebas de la manera más eficiente y cómoda. En primer lugar, se ha llevado a cabo la instalación y configuración de las Raspberry Pi 4 que se han utilizado para actuar como conmutadores *SDN*. A continuación, se ha utilizado RealVNC para el acceso remoto a las Raspberry Pi desde el equipo principal y Mininet para emular las topologías de red deseadas. Por último, se ha empleado el controlador POX para programar el control de la red.

4.2.1 Raspberry Pi

Estas pruebas deberían funcionar en distintos dispositivos ya sean máquinas virtuales o diferentes ordenadores. Para la realización de este proyecto, se ha optado por utilizar 4 dispositivos Raspberry Pi 4 conectados entre sí.

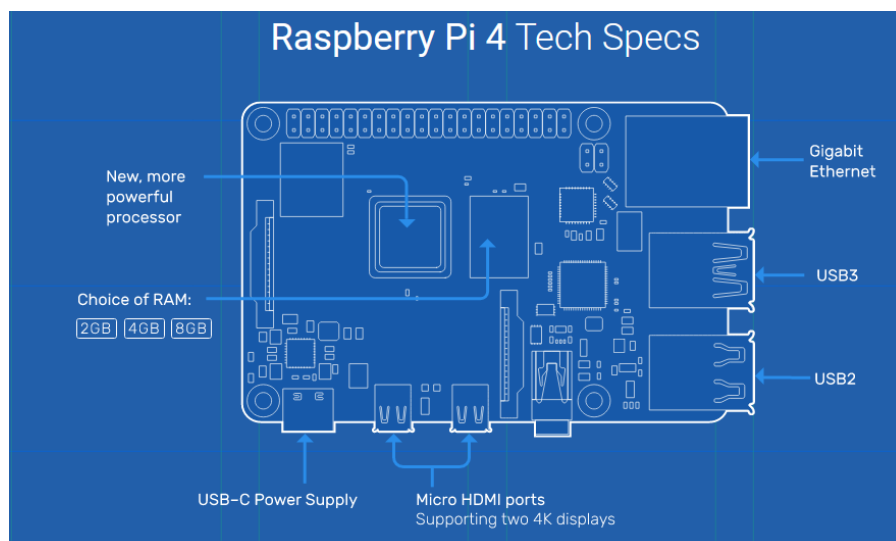


Figura 6: Plano con los apartados de la Raspberry Pi 4

Fuente: Especificaciones de Raspberry Pi 4 [11]

Las Raspberry Pi son unos ordenadores con una sola placa (Single Board Computer) con un coste bajo desarrollados en Reino Unido para fomentar la docencia en informática en diversas instituciones como colegios, institutos y universidades.

El hardware es un producto registrado, pero su software es completamente libre, pudiendo utilizar cualquier sistema operativo que se desee (Windows, Linux, Ubuntu, Fedora...), aunque el oficial es Raspberry Pi OS 32bits, que es una versión adaptada a partir de Debian. La versión 64b no dispone de tanto soporte como la versión de 32b, puede causar problemas a la hora de instalar librerías o programas desarrollados para Raspberry teniendo en mente la versión de 32b como puede suceder con la instalación de la herramienta Mininet, de la que se hablará más adelante.

A la hora de comprar una Raspberry Pi, no incluye periféricos como teclado, ratón o pantalla, aunque tienen disponibles algunos accesorios oficiales. Como dispositivo de salida de imagen utiliza *HDMI*, y no dispone de entradas clásicas para ratón y teclado por lo que se requiere de periféricos que se conecten mediante *USB*.

Para este proyecto se han utilizado 2 Raspberry Pi 4 modelo B con 4GB de memoria *RAM*, memoria *MicroSD* de 32GB. A la hora de instalar el sistema operativo y configurar las Raspberry Pi 4 se ha seguido el mismo procedimiento con las 2.

Para comenzar con el proceso de instalación, se accede a la página oficial (<https://www.raspberrypi.com>), en el apartado Software y se descarga el instalador Raspberry Pi Imager en la versión del Sistema Operativo (S.O. para abreviar de ahora en adelante) en el que se vaya a utilizar. Una vez se haya instalado la herramienta, se elige la imagen del S.O. que se vaya a instalar en la Raspberry, que para este proyecto se ha utilizado Raspberry Pi OS 32-bit, y seleccionamos el almacenamiento donde deseamos realizar la instalación del S.O. que en este caso se ha utilizado la memoria que

emplean las Raspberry Pi 4 por defecto que consiste en una tarjeta *MicroSD* extraíble.

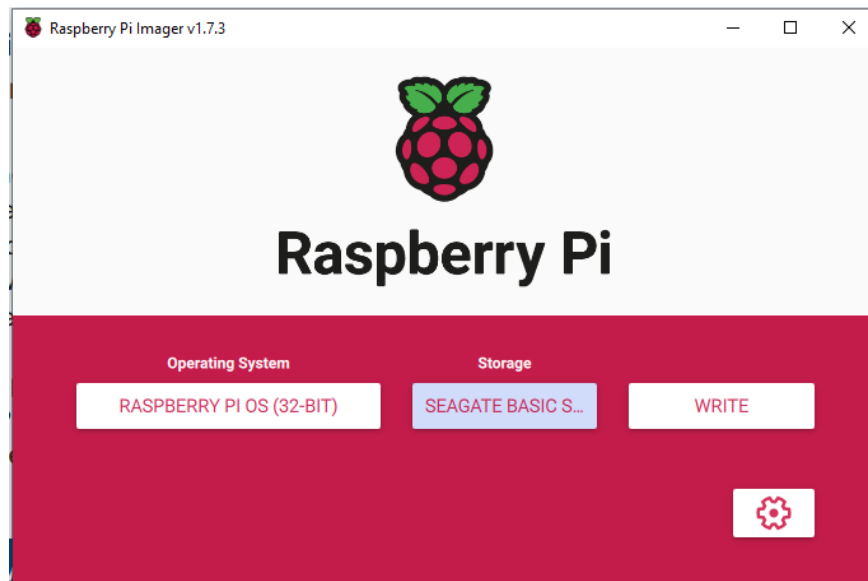


Figura 7: Instalador del Sistema Operativo para Raspberry Pi

Fuente: Propia

Una vez se haya terminado la instalación de la imagen en la memoria *MicroSD*, se inserta en la parte inferior de la Raspberry Pi. Para realizar la configuración inicial, hay que utilizar un ratón y un teclado conectados a los *USB* de la Raspberry Pi y un monitor con entrada *HDMI*. Se selecciona el idioma que se vaya a utilizar y luego para que el S.O. acabe de instalarse, hay que conectar la Raspberry a internet, ya sea por *Wi-Fi* o mediante cable Ethernet. Para este proyecto, se ha utilizado la conexión *Wi-Fi* por comodidad, pero ambos métodos funcionan correctamente.

Una vez finalizada la instalación, se reinicia la Raspberry y se solicita la contraseña que se utilizará para los comandos que requieran permisos de administrador. Y ya se puede utilizar la Raspberry Pi 4 y actualizarla de la misma forma que se puede actualizar cualquier sistema basado en Linux. En este proyecto, para poder utilizar de una forma más cómoda las Raspberry sin tener que estar cambiando los dispositivos periféricos constantemente,

se ha instalado otro programa llamado RealVNC del que se habla más adelante.

4.2.2 RealVNC

Este software será utilizado para controlar los diferentes dispositivos Raspberry Pi 4 mediante el uso de un escritorio remoto. Este programa permite controlar de manera simultánea hasta 5 dispositivos de forma gratuita sin necesidad de tener varios periféricos en uso o estar desconectando y conectando el teclado, ratón o pantalla.

Instalación de VNC Viewer

VNC Viewer es la parte de la herramienta que permite controlar el software VNC que se encuentra en Raspberry Pi. Para la instalación, se descarga el instalador que se encuentra en la página web oficial de RealVNC (<https://www.realvnc.com>) en el apartado de *Descargas > VNC Viewer*. Se debe descargar el instalador correspondiente al S.O. de la máquina que se esté utilizando, y para este proyecto, se ha instalado VNC Viewer en Windows. Una vez descargado e instalado, ya estará en funcionamiento la aplicación.

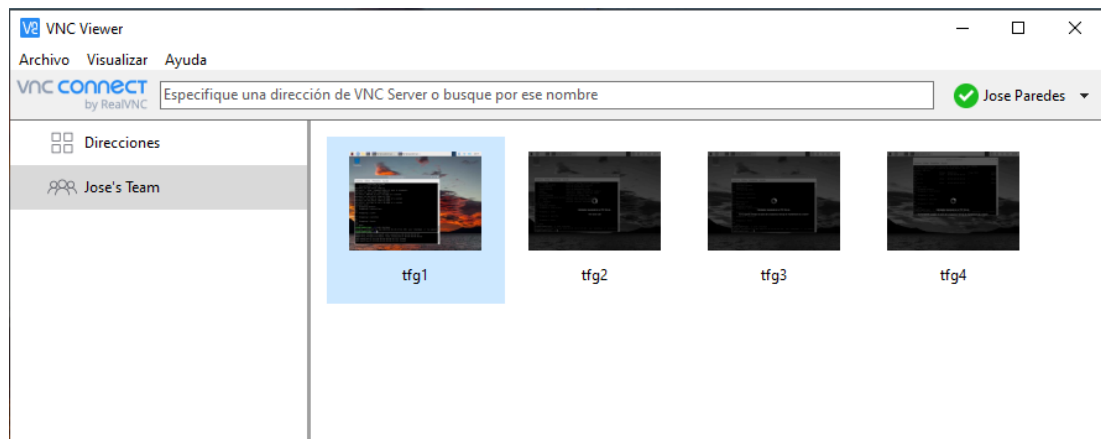


Figura 8: Interfaz VNC Viewer

Fuente: Propia

Instalación VNC Server

VNC Server es la herramienta que se encarga de enviar el flujo de información al Viewer y obtener los inputs que se realicen desde el dispositivo maestro. VNC viene instalado de forma nativa en el S.O. Raspbian, pero por defecto viene desactivado. Para habilitarlo, hay que ir a *Menú > Preferencias > Configuración de Raspberry Pi > Interfaces*.

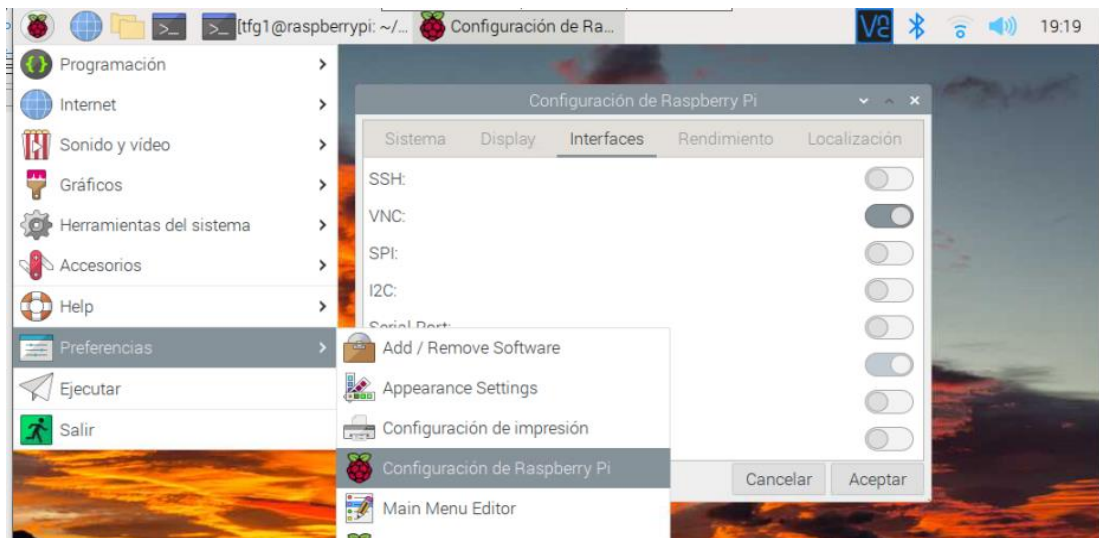


Figura 9: Activación VNC en Raspberry Pi 4

Fuente: Propia

Se busca en la lista la opción VNC y se activa. Una vez se haya hecho, el software ya se pondrá en funcionamiento y también en futuros arranques del dispositivo. Aparecerá un icono en la parte superior derecha indicando que se encuentra activo.

Una vez se inicie VNC Server, se puede conectar de diferentes formas: Mediante la creación de una cuenta o conectarse directamente mediante la IP del dispositivo. Para este proyecto, se ha creado una cuenta de RealVNC gratuita y para vincular los dispositivos con el dispositivo maestro.

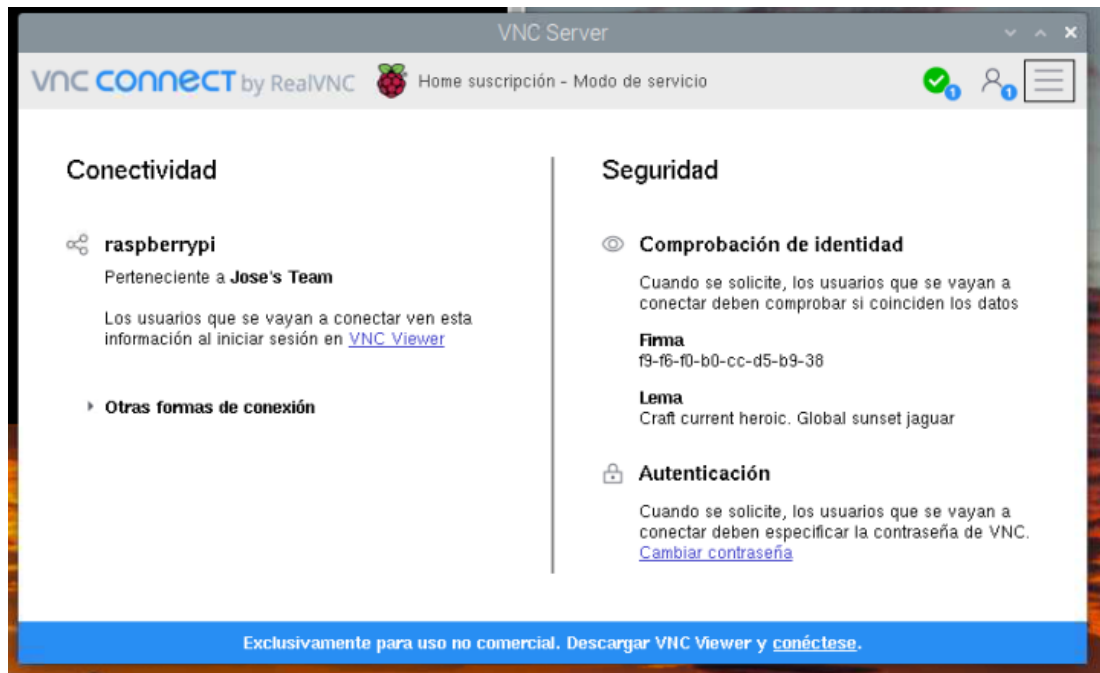


Figura 10: Interfaz VNC Server en Raspberry Pi 4

Fuente: Propia

4.2.3 Mininet

Antes de realizar las conexiones entre las diferentes Raspberry Pi, se ha optado por Mininet para realizar las pruebas con el controlador SDN para simular el control de diferentes dispositivos en una red antes de preparar toda la infraestructura con las Raspberry.

Para comenzar la instalación de Mininet, primero se debe instalar git mediante este comando

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

Con este comando se descarga el contenido que se encuentra en el Git de Mininet.

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

Hay que entrar en la carpeta que se ha creado con la descarga anterior y utilizar el comando “git tag” para obtener una lista de todas las versiones disponibles de Mininet. El comando “git checkout -b” es para seleccionar la versión que se desea utilizar.

```
$ cd mininet
$ git tag
1.0.0
2.0.0
2.1.0
2.1.0p1
2.1.0p2
2.2.0
2.2.1
2.2.2
2.3.0
2.3.0b1
.
# etc.
$ git checkout -b “versión que deseemos”
```

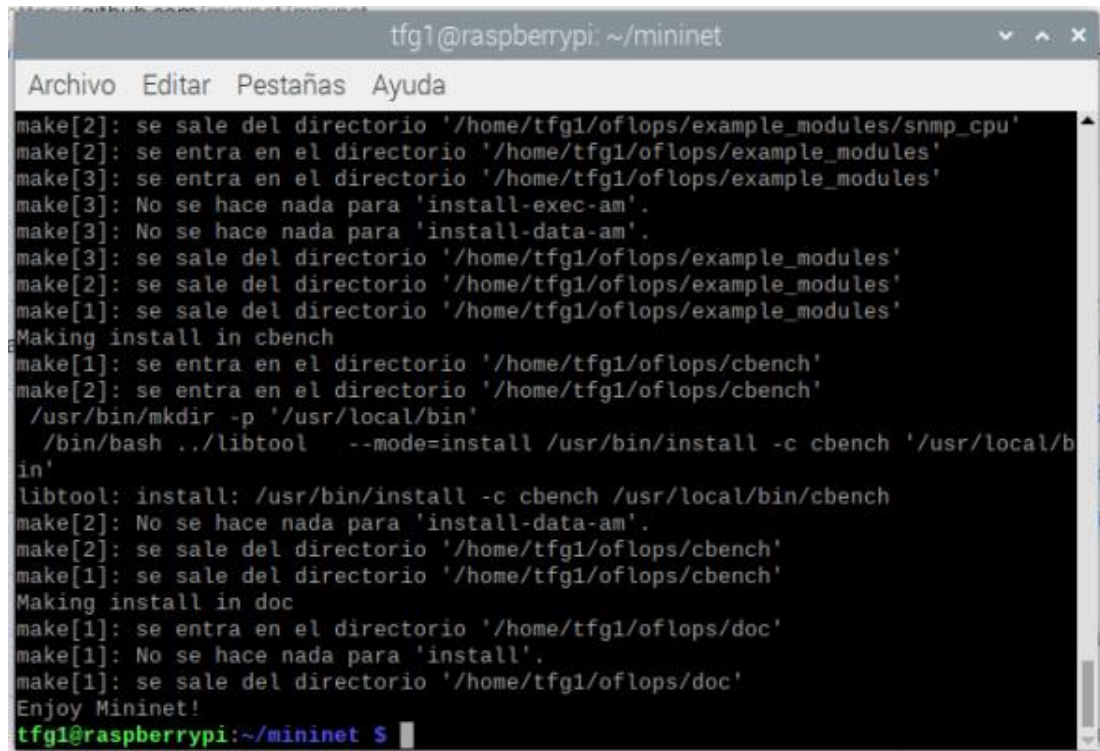
Para este proyecto, se ha utilizado el comando “git checkout” sin especificar una versión para que de esta forma se seleccione la última versión que se encuentra en la rama principal del proyecto.

```
$ git checkout
```

Una vez seleccionado, hay que acceder a la carpeta *mininet>util* y ahí llamar al script que se encarga de la instalación. Como parámetro extra se ha añadido el -a ya que esto hace que se instalen también todas las dependencias necesarias que no se encuentren en el sistema actualmente.

```
$ util/install.sh -a
```

Una vez se haya completado la instalación, aparecerá un mensaje que dice: “Enjoy Mininet!”



```
tfg1@raspberrypi: ~/mininet
Archivo  Editar  Pestañas  Ayuda
make[2]: se sale del directorio '/home/tfg1/oflops/example_modules/snmp_cpu'
make[2]: se entra en el directorio '/home/tfg1/oflops/example_modules'
make[3]: se entra en el directorio '/home/tfg1/oflops/example_modules'
make[3]: No se hace nada para 'install-exec-am'.
make[3]: No se hace nada para 'install-data-am'.
make[3]: se sale del directorio '/home/tfg1/oflops/example_modules'
make[2]: se sale del directorio '/home/tfg1/oflops/example_modules'
make[1]: se sale del directorio '/home/tfg1/oflops/example_modules'
Making install in cbench
make[1]: se entra en el directorio '/home/tfg1/oflops/cbench'
make[2]: se entra en el directorio '/home/tfg1/oflops/cbench'
/usr/bin/mkdir -p '/usr/local/bin'
/bin/bash ../libtool  --mode=install /usr/bin/install -c cbench '/usr/local/b
in'
libtool: install: /usr/bin/install -c cbench /usr/local/bin/cbench
make[2]: No se hace nada para 'install-data-am'.
make[2]: se sale del directorio '/home/tfg1/oflops/cbench'
make[1]: se sale del directorio '/home/tfg1/oflops/cbench'
Making install in doc
make[1]: se entra en el directorio '/home/tfg1/oflops/doc'
make[1]: No se hace nada para 'install'.
make[1]: se sale del directorio '/home/tfg1/oflops/doc'
Enjoy Mininet!
tfg1@raspberrypi:~/mininet $
```

Figura 11: Instalación exitosa de Mininet

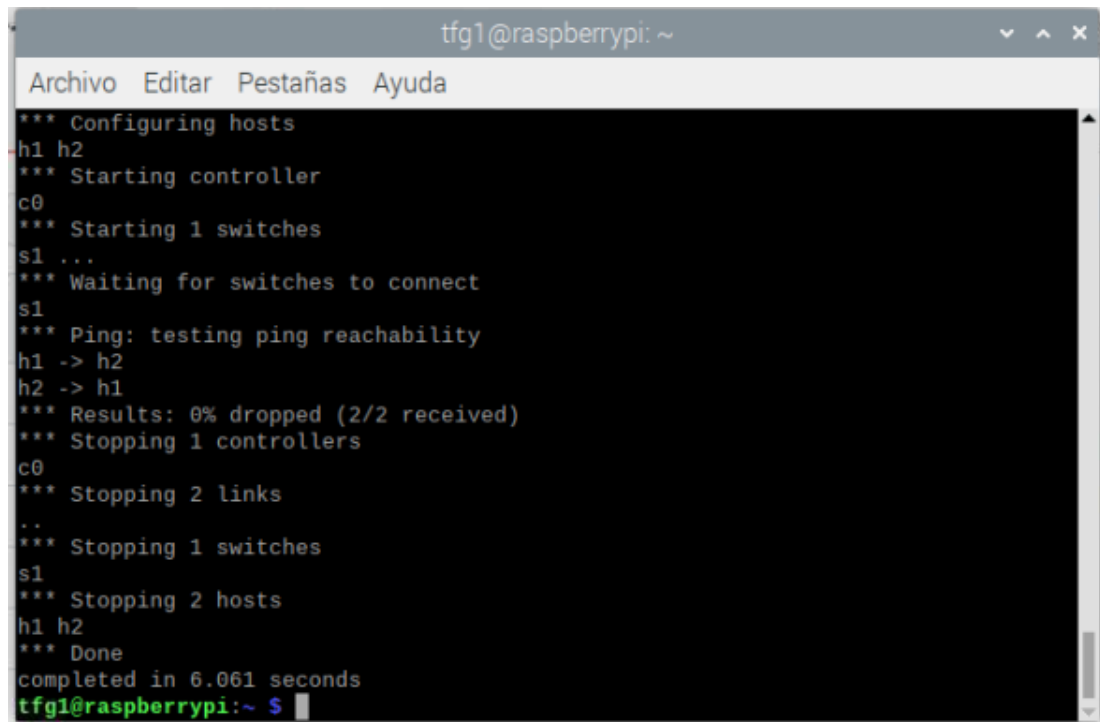
Fuente: Propia

Si se desea comprobar que la aplicación funciona y se ha instalado sin errores, se puede ejecutar este comando.

```
$ sudo mn --test pingall
```

Este comando crea una topología con los componentes básicos para funcionar: un controlador (c0), dos dispositivos (h1 y h2) y un conmutador (s1). Tras conectar los dispositivos al conmutador y el conmutador al controlador, inicia el controlador y realiza pruebas de ping de h1 a h2 y viceversa. Este ejemplo sirve para comprobar que la instalación de Mininet

se ha realizado con éxito y que es capaz de crear una topología básica y realizar funciones sencillas, aun no se han conectado las Raspberry entre sí.



```
tfg1@raspberrypi: ~
Archivo  Editar  Pestañas  Ayuda
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 6.061 seconds
tfg1@raspberrypi:~ $
```

Figura 12: Comprobación del correcto funcionamiento de Mininet

Fuente: Propia

También dispone de una interfaz gráfica llamada MiniEdit para crear las topologías si no se quiere trabajar con comandos, si resulta más sencillo crear la topología utilizando un apoyo visual, con editor de texto o si no se requiere de una topología distribuida como la que se está realizando para el proyecto.

El script se encuentra dentro de la carpeta *mininet>examples*. Para ejecutarlo, hay que escribir lo siguiente en la línea de comandos:

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

Esta herramienta permite crear de forma rápida una interfaz. Permite configurar componentes básicos de la red de una forma bastante sencilla.

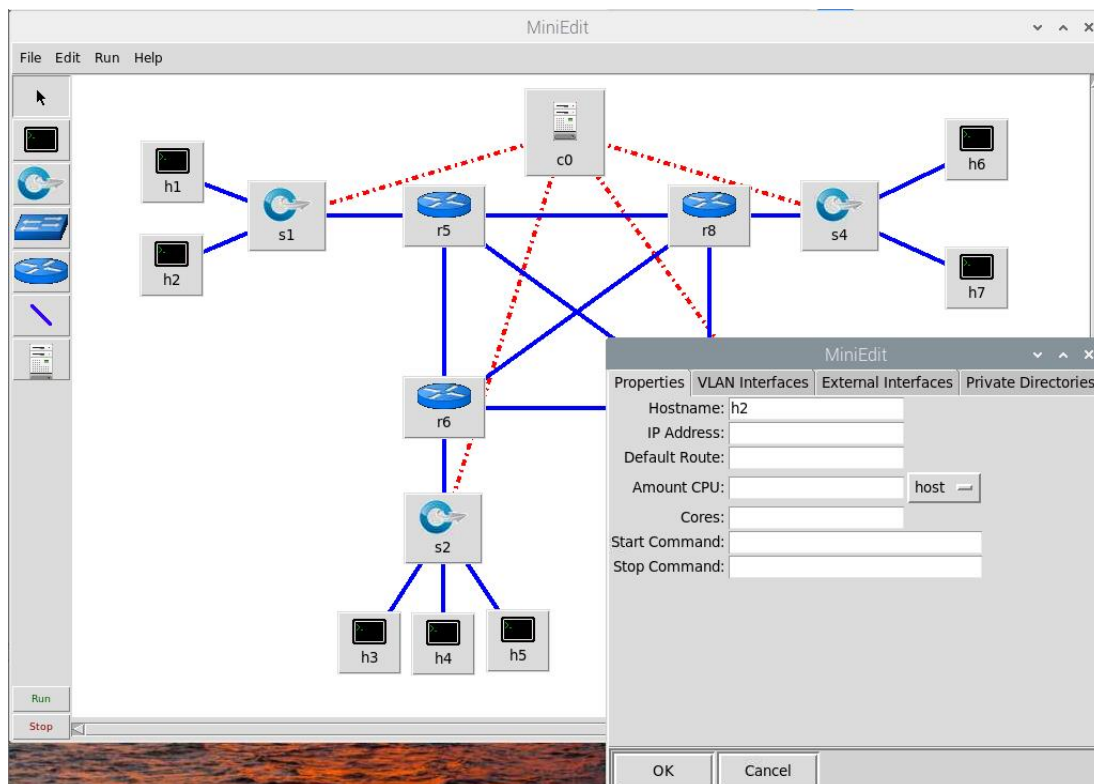


Figura 13: MiniEdit

Fuente: Propia

Los iconos que se muestran en la izquierda de la ventana son:

- Flecha: Permite mover los componentes de la red y mediante clic derecho se accede a su configuración.
- Host: Crea un dispositivo en la red.
- Switch: Por defecto crea un conmutador configurable. Se puede cambiar a otros tipos de conmutador, pero el que viene por defecto Open vSwitch es el que se necesita para este proyecto.
- Legacy Switch: Crea un conmutador independiente del controlador SDN.
- Legacy Router: Crea un enrutador independiente del controlador.
- Enlace: Conecta los componentes de la red.
- Controlador: Crea un controlador configurable que utiliza el estándar de OpenFlow.

En la configuración de los componentes se pueden modificar las interfaces del dispositivo, cambiar las direcciones IP, cuál es la puerta de enlace y si queremos que ejecute algún comando al arrancar.

Abajo a la izquierda se puede comenzar la emulación de la red que se acaba de crear, así como parar la emulación. En el apartado File se puede tanto guardar la topología, para poder realizar cambios en un futuro, o se puede exportar a un fichero ejecutable, que permite obtener un script escrito en Python que crea la topología que se ha generado usando la herramienta. De esta forma, se puede crear una topología visualmente y luego en el script modificar los valores que no se pueden mediante el editor gráfico, como es el caso de este proyecto que tiene una topología separada en 2 dispositivos diferentes.

4.2.4 Controlador POX



Figura 14: Logo POX

Fuente: Learning POX OpenFlow controller [12]

POX es el controlador *SDN* por defecto que viene integrado en la instalación junto con Mininet, por lo que no sería necesario instalar nada de forma independiente. En caso de que se necesite, con el siguiente comando se descarga el repositorio de GitHub del controlador POX. Este paso solo es necesario realizarlo en una de las Raspberry, ya que es suficiente con un único controlador que gestione toda la red *SDN*.

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

Una vez finalice la descarga del repositorio, se puede modificar la versión activa ejecutando dentro de la carpeta donde se ha descargado el repositorio el comando “git branch -a” para listar todas las versiones y después “git checkout eel” para modificar la versión que se pretende utilizar a la hora de ejecutar los comandos del controlador.

```
$ git branch -a
* eel
  gar-experimental
  remotes/origin/HEAD -> origin/gar-experimental
  remotes/origin/angler
```

```
remotes/origin/betta
remotes/origin/carp
remotes/origin/dart
remotes/origin/debugger
remotes/origin/eel
remotes/origin/fangtooth
remotes/origin/gar-experimental
remotes/origin/halosaur
remotes/origin/libfluid_experiment
$ git checkout eel
Ya en 'eel'
Tu rama está actualizada con 'origin/eel'.
```

Se ha optado por la versión “eel” debido a un problema sin solucionar con la actualización de Python3 y el método que utiliza el controlador para comprobar que los paquetes entre las topologías llegan correctamente. Para que la versión “eel” funcione, primero es necesario instalar Python2 en el dispositivo. Por defecto, se instala la última versión de Python3, por lo que hay que ejecutar este comando para instalar Python2.

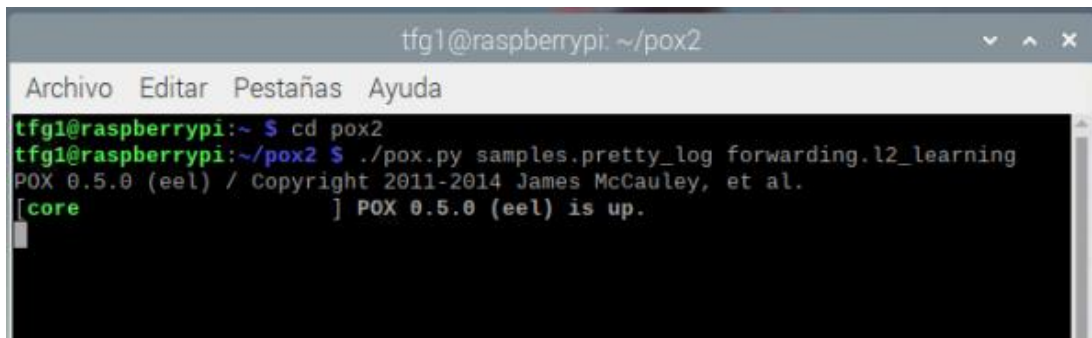
```
Sudo apt-get install python2
```

Para probar si funciona o si se encuentra instalado, se tiene que acceder a la carpeta en la que se hayan descargado los archivos del paso anterior y llamar a este comando:

```
./pox.py samples.pretty_log forwarding.l2_learning
```

Este comando sirve para realizar un arranque básico del controlador POX, sirve para realizar pruebas sencillas como conectar las Raspberry entre sí o en este caso, comprobar que no ha habido ningún problema con la

instalación. Si aparece esto en la consola, significa que el controlador está funcionando correctamente.



```
tfg1@raspberrypi: ~/pox2
Archivo Editar Pestañas Ayuda
tfg1@raspberrypi:~$ cd pox2
tfg1@raspberrypi:~/pox2$ ./pox.py samples.pretty_log forwarding.l2_learning
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
[core] POX 0.5.0 (eel) is up.
```

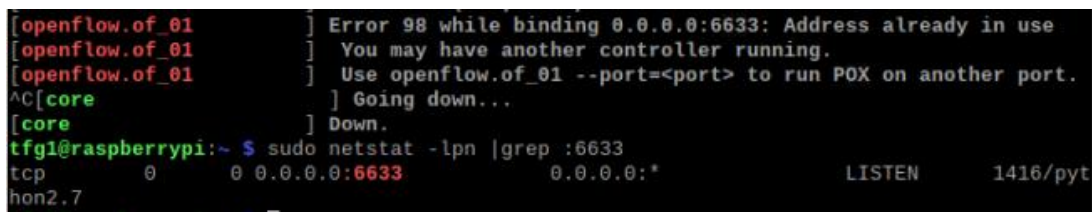
Figura 15: Ejecución de controlador POX eel

Fuente: Propia

En caso de que aparezca algún error indicando que ya se encuentra en uso otro controlador, puede ser debido a que no se haya cerrado adecuadamente el controlador en una sesión anterior. Para evitar ese problema, se utiliza el siguiente comando:

```
sudo netstat -ltn |grep :6633
```

Con este comando, se obtiene el id del proceso que sigue ejecutando. Una vez se obtenga su id, basta con eliminar el proceso con ese identificador y volver a ejecutar el comando anterior para activar el controlador.



```
[openflow.of_01] Error 98 while binding 0.0.0.0:6633: Address already in use
[openflow.of_01] You may have another controller running.
[openflow.of_01] Use openflow.of_01 --port=<port> to run POX on another port.
^C[core] Going down...
[core] Down.
tfg1@raspberrypi:~$ sudo netstat -ltn |grep :6633
tcp        0      0 0.0.0.0:6633          0.0.0.0:*            LISTEN      1416/python
hon2.7
```

Figura 16: Error al levantar el controlador

Fuente: Propia

4.3 Tiempo invertido en el proyecto

En este apartado, se procede a realizar un desglose de las actividades realizadas y un tiempo aproximado en horas de su duración. Estos intervalos no son muy definidos ya que algunas de las tareas se han ido realizando simultáneamente en conjunto con otras, como por ejemplo las secciones de realizar las pruebas junto con sección de solucionar errores o la de documentación de la memoria.

- Configuración de Raspberry: 20h
- Búsqueda de información: 45h
- Creación de pruebas: 30h
- Conexión entre Raspberry: 35h
- Topología Cortafuegos: 15h
- Topología STP: 25h
- Resolución de errores: 70h
- Interpretación de resultados: 10h
- Redacción de memoria: 50h

El tiempo total aproximado es de 300 horas. En la siguiente imagen, se encuentra un gráfico para representar visualmente cuándo se han ido realizando las diferentes tareas:

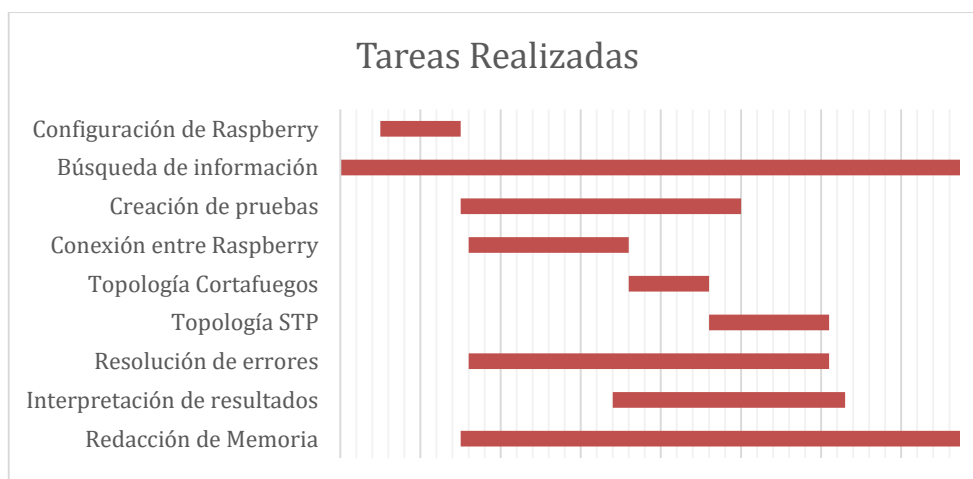


Figura 17: Gráfico de horas

Fuente: Propia

Como se puede observar, la prueba que más tiempo ha requerido se trata de la de conexión entre las Raspberry. Esta era la más importante ya que, en caso de no funcionar correctamente, no se habrían podido llevar a cabo el resto de pruebas porque todas disponen como base lo que se ha utilizado en esta primera prueba para conectar los dispositivos entre sí. En apartados posteriores se explica cómo se ha realizado esa conexión.

Sin embargo, la tarea que más tiempo ha necesitado ha sido la resolución de errores. Las herramientas empleadas se encuentran preparadas para lanzar topologías teniendo en cuenta un único dispositivo en el que se está ejecutando y no la topología distribuida que se pretendía conseguir realizando este proyecto. Por tanto, ha habido que realizar ajustes en las pruebas y en las Raspberry que se encuentran explicados en apartados posteriores de una forma sencilla con el objetivo de evitar esa pérdida de tiempo en futuros proyectos que utilicen este documento como base.

5. Pruebas Realizadas y Resultados

Para realizar las pruebas de la red SDN emulada mediante Raspberry Pi 4 y el controlador POX, es necesario establecer una topología de red adecuada que permita la interconexión de los dispositivos utilizados. La topología puede variar en cada una de las pruebas, pero la estructura que tiene la conexión entre las Raspberry Pi siempre va a tener la misma forma que se muestra en el diagrama a continuación.

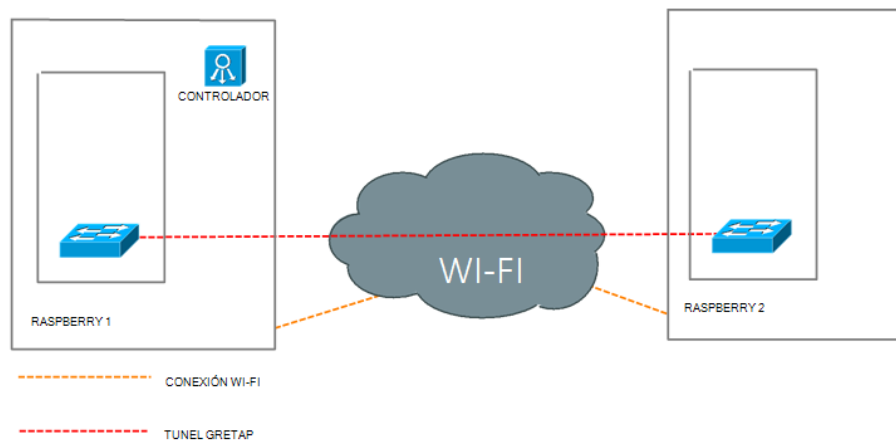


Figura 18: Representación de la conexión de las Raspberry Pi

Fuente: Propia

En este diagrama se muestran dos recuadros grandes representando a los dispositivos físicos. Estos dispositivos se encuentran conectados mediante una red Wi-Fi doméstica. Dentro de la Raspberry Pi 1, se ejecuta la instancia del controlador SDN POX que se encargará de la gestión de las diferentes topologías que se utilizarán en las distintas pruebas que se explicarán en esta sección. Por último, los recuadros interiores representan las diferentes topologías que se van a ejecutar utilizando Mininet y en su interior los conmutadores SDN mediante los que se encuentran conectadas las topologías que están en cada máquina con ese túnel GRE-TAP. Todos estos apartados de la estructura se explicarán con más detalle sobre cómo se inician o cómo se deben configurar en las próximas secciones del documento. En algunos

diagramas, por comodidad visual, se omitirá dibujar enlaces entre los conmutadores y el controlador. Todos los conmutadores en cada una de las topologías y pruebas realizadas se encuentran conectados al controlador, ya que esa es la base del funcionamiento de las redes *SDN*.

En este proyecto, se han llevado a cabo diversas pruebas en escenarios varios para comprobar el funcionamiento de las redes *SDN* en esas situaciones. Para ello, se han realizado pruebas para conectar las topologías, configurar reglas de cortafuegos en redes *SDN* o el tratamiento de bucles en las topologías y capacidad de reacción a los cambios en la red.



Figura 19: Raspberry Pi utilizadas para el proyecto

Fuente: Propia

5.1 Conexión entre las Raspberry Pi

Para comenzar la conexión entre las diferentes topologías, se debe arrancar el controlador. No es necesario arrancar una topología antes que otra, o iniciar el controlador antes que las topologías. Según la prueba que se vaya a realizar, el comando requerido para el arranque del controlador puede variar dependiendo de la funcionalidad que se desee probar. **En cada prueba realizada se indica cuál es el comando que se debe utilizar para iniciar el controlador en cada una de las pruebas.** Para probar este caso, se va a realizar la conexión más básica entre las diferentes topologías. Para ello, hay que abrir una terminal, se introduce la ruta en la que se encuentra el controlador POX descargado y se ejecuta el siguiente comando:

```
./pox.py samples.pretty_log forwarding.l2_learning
```

Con este comando, comienza la ejecución del controlador como se ha realizado en el apartado [4.4 Controlador POX](#). Dicho controlador SDN se encuentra alojado en la Raspberry Pi 1 como se muestra en el diagrama:

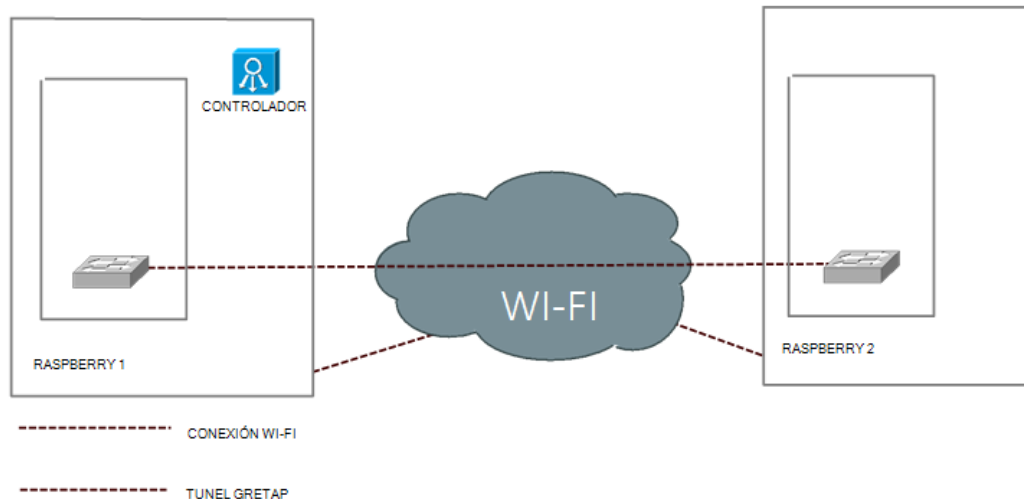


Figura 20: Ubicación del controlador SDN en la topología

Fuente: Propia

Este apartado se debe realizar para comenzar a realizar todas las pruebas que se detallan en los siguientes apartados. Se puede modificar la topología de cada dispositivo añadiendo más terminales, más conmutadores u otros

componentes a la red, pero aquí se describen los mínimos que deben ejecutarse en cada topología de cada ejemplo para poder realizar la conexión entre ellas.

El principal problema a la hora de trabajar con Mininet, es que está preparada para emular redes en un único dispositivo y eso conlleva varios problemas de los que se hablan detalladamente en la sección [5.4 Problemas durante el desarrollo del proyecto](#). Para poder conectar los dispositivos entre sí, se ha optado por utilizar una conexión *GRETAP* (Generic Routing Encapsulation Terminal Access Point) entre las Raspberry. Esta conexión se ha establecido ejecutando el siguiente comando dentro del script de Python:

```
s11.cmd("ip link add s11-tfg2 type gretap local 192.168.1.151 remote 192.168.1.152 ttl 64")
```

Este fragmento de código indica lo que el script debe ejecutar en la consola de comandos del conmutador s11 de la topología. El contenido del paréntesis consiste en crear una conexión llamada **“s11-tfg2”** del tipo **GRETAP**, que es un tipo de encapsulamiento de los datos, y se le indica la **IP origen**, la **IP destino** y un **Tiempo de Vida** (Time To Live, *TTL*) de 64 segundos.

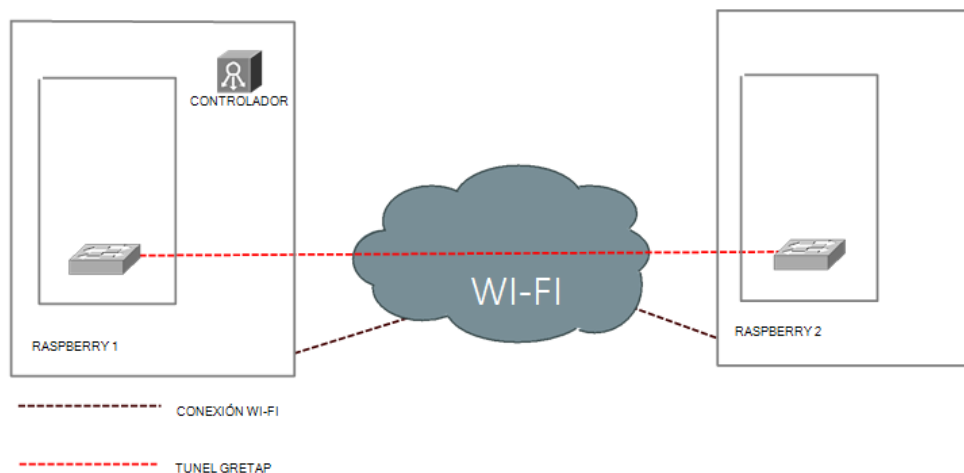


Figura 21: Ubicación del túnel GRETAP en la topología

Fuente: Propia

El código utilizado en cada topología en cada una de las Raspberry se encuentra en la sección [9. Anexos](#) del documento. Para este ejemplo, se han creado **dos topologías**, cada una con **3 terminales** conectadas a un **conmutador SDN** y estos a su vez, se encuentra conectado al **controlador POX**.

La topología resultante de ejecutar los scripts en los respectivos dispositivos quedaría de la siguiente forma:

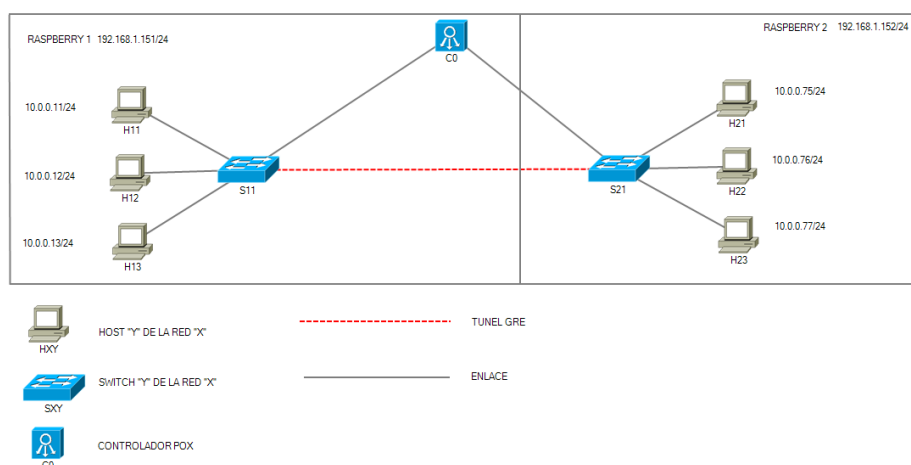


Figura 22: Topología para probar la conexión entre Raspberry

Fuente: Propia

Para comprobar que las diferentes topologías se encuentran conectadas, basta con realizar un ping de una terminal a otra que se encuentre en una topología diferente. Se debe ejecutar uno de estos dos códigos, el primero solo se puede ejecutar si se ha introducido el alias del host en el archivo de hosts como se ha indicado en la sección [5.4 Problemas durante el desarrollo del proyecto](#). En caso de no haberlo hecho, saldrá un mensaje de error en la consola indicando que no existe un dispositivo con ese alias:

```
H11 ping H21  
H11 ping 10.0.0.77
```

5.2 Cortafuegos

Un cortafuegos es un método de seguridad que se utiliza para proteger una red de computadoras contra amenazas externas, como hackers, malware y ataques cibernéticos. Consiste en un dispositivo o software que filtra y controla el tráfico de red entrante y saliente.

El cortafuegos establece reglas de seguridad que permiten o bloquean el tráfico de red en función de ciertos criterios, como la dirección IP de origen o destino, el puerto utilizado y el protocolo utilizado. Estas reglas se basan en políticas de seguridad que se definen según los requisitos específicos de la red y de la organización.

Existen varios tipos de cortafuegos como son los **cortafuegos de red**, que se ubican entre la red interna y la red externa (Internet) y filtra el tráfico de red que entra y sale de la red interna; **cortafuegos del dispositivo**, que se ejecuta en un servidor o dispositivo final y filtra el tráfico de red entrante y saliente en ese dispositivo; y el **cortafuegos de aplicación**, que filtra el tráfico de red para aplicaciones específicas en lugar de todo el tráfico de red.

Un cortafuegos también puede incluir características adicionales, como detección de intrusiones, prevención de intrusiones, filtrado de *URL* y protección contra ataques de denegación de servicio o DoS (*Denial of Service*).

Un cortafuegos en el ámbito de las redes *SDN* es similar a un cortafuegos tradicional, solo que es capaz de gestionar y controlar el tráfico de red mediante el uso de la lógica de programación en lugar de dispositivos de hardware especializados. Puede ser implementado como una aplicación de red que se ejecuta en el controlador *SDN* y utiliza la *API SDN* para monitorizar y controlar el tráfico de red que fluye a través de la red *SDN*. Por ejemplo, el cortafuegos puede permitir el acceso a ciertos recursos de la red solo a usuarios autorizados y aplicar políticas de seguridad personalizadas para proteger la red contra amenazas externas.

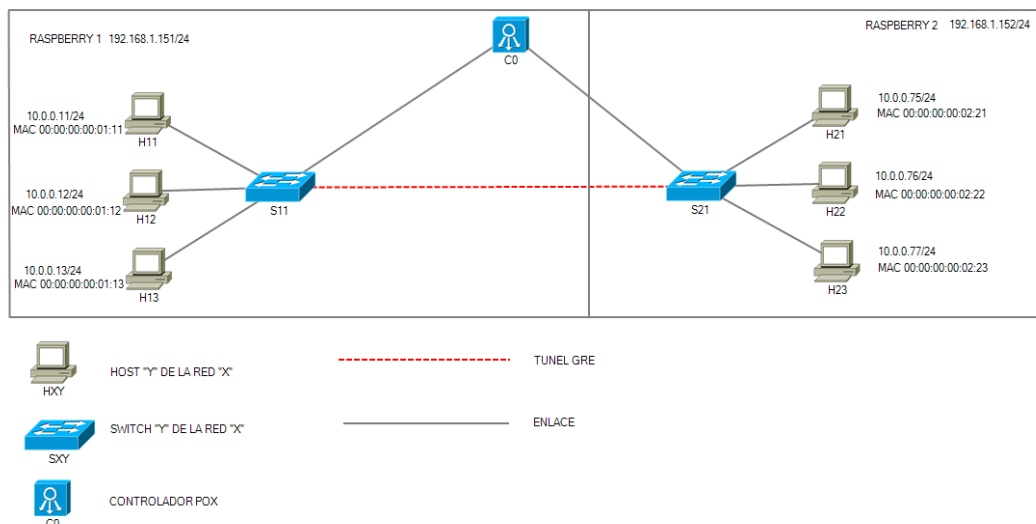


Figura 23: Topología cortafuegos

Fuente: Propia

Para este ejemplo se ha optado por utilizar los mismos componentes de la prueba anterior, con la diferencia de que estos tienen añadida una dirección MAC (Media Access Control), ya que resulta necesaria para realizar las pruebas con el cortafuegos. Para arrancar el controlador con las cualidades requeridas en esta prueba, se debe crear un documento en la carpeta en la que se encuentra el controlador POX descargado y utilizar el código para el cortafuegos que se encuentra en el apartado [9. Anexos](#) denominado **“Firewall.py”**. Una vez se haya creado un documento, hay que abrir una terminal, se introduce la ruta en la que se encuentra el controlador POX descargado y se ejecuta el siguiente comando:

```
./pox.py log.level --DEBUG openflow.of_01
forwarding.l2_learning misc.firewall
```

Este comando se encarga de ejecutar el controlador POX con las funciones implementadas en el documento. La consola del controlador se muestra de la siguiente forma:

```
tfg1@raspberrypi:~$ ./pox2/pox.py log.level --DEBUG openflow.of_01 forwarding
2_learning misc.firewall
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
INFO:misc.firewall:Enabling Firewall Module
DEBUG:core:POX 0.5.0 (eel) going up...
DEBUG:core:Running on CPython (2.7.18/Jul 14 2021 08:11:37)
DEBUG:core:Platform is Linux-5.15.84-v7l+-armv7l-with-debian-11.6
INFO:core:POX 0.5.0 (eel) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-0b 2] connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-0b 2]
INFO:misc.firewall:Adding firewall rule drop: src 10.0.0.11 - dst 10.0.0.75
INFO:misc.firewall:Adding firewall rule drop: src 10.0.0.75 - dst 10.0.0.11
INFO:misc.firewall:Adding firewall rule drop: src 10.0.0.76 - dst 10.0.0.13
INFO:misc.firewall:Adding firewall rule drop: src 10.0.0.13 - dst 10.0.0.76
INFO:misc.firewall:Firewall rules installed on 00-00-00-00-00-0b
```

Figura 24: Ejecución del controlador POX con reglas de Firewall

Fuente: Propia

En esta imagen, el controlador SDN está instalando una serie de reglas a los controladores que se encuentran en la topología. A continuación, se muestra la topología resultante que se está ejecutando entre todas las Raspberry:

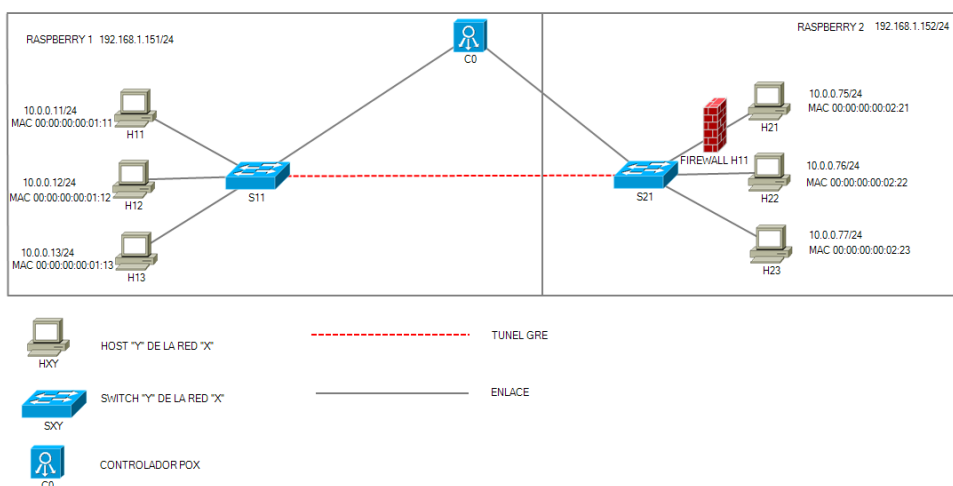


Figura 25: Topología con un cortafuegos entre H11 y H21

Fuente: Propia

Hay varias formas de modificar las reglas que se aplican al flujo de datos mediante el cortafuegos del controlador: Crear un documento CSV con reglas predeterminadas para cuando arranque el cortafuegos o utilizando comandos del Shell de Mininet. Se hablará de ambas opciones a continuación.

El firewall que se encuentra en el anexo dispone de una sección al comienzo del script que dice lo siguiente:

```
policyFile = "%s/pox2/pox/misc/firewall-policies.csv" %  
os.environ[ 'HOME' ]  
policies = csv.DictReader(open(policyFile, "rb"))
```

Este comando abre un documento llamado *firewall-policies.csv*. Contiene unas reglas preestablecidas que se aplican al firewall en cuanto arrancamos el controlador con el comando mencionado anteriormente. En este documento, se encuentran las reglas indicadas siguiendo esta forma:

```
id,ip_0,ip_1  
ID,IP_SRC,IP_DST
```

Siendo ID el identificador numérico único de la regla, *IP_SRC* la dirección *IP* del dispositivo origen y *IP_DST* la dirección *IP* del dispositivo de destino. Esto permite crear una serie de reglas para limitar el tráfico entre componentes basándose en su dirección *IP*. El documento *firewall-policies.csv* se encuentra configurado de la siguiente forma para esta primera prueba:

```
id,ip_0,ip_1  
1,10.0.0.11,10.0.0.75  
3,10.0.0.76,10.0.0.13
```

Se ha cortado la conexión de esta forma desde H11 dirección H21 y entre H22 dirección H13. Para comprobar que el cortafuegos funciona, se puede comprobar utilizando la herramienta Wireshark para monitorizar el tráfico

entre las terminales y haciendo un ping desde la terminal de Mininet utilizando el comando “**h11 ping -c 3 h21**”, que se encarga de realizar 3 pings entre H11 y H21.

1	0.000000000	00:00:00_00:01:11	00:00:00_00:02:21	ARP	42 Who has 10.0.0.75? Tell 10.0.0.11
2	0.052599566	00:00:00_00:02:21	00:00:00_00:01:11	ARP	42 10.0.0.75 is at 00:00:00:00:02:21
3	9.498150605	0.0.0.0	255.255.255.255	DHCP	384 DHCP Discover - Transaction ID 0x79c9f494
4	9.948386673	0.0.0.0	255.255.255.255	DHCP	384 DHCP Discover - Transaction ID 0xe9c4618e
5	18.990211836	0.0.0.0	255.255.255.255	DHCP	384 DHCP Discover - Transaction ID 0xd00f5cbc
6	19.604846632	0.0.0.0	255.255.255.255	DHCP	384 DHCP Discover - Transaction ID 0xdce4e757
7	41.966332378	10.0.0.75	10.0.0.11	ICMP	98 Echo (ping) request id=0x8ff5, seq=1/256, ttl=64 (no respons...
8	42.940427837	10.0.0.75	10.0.0.11	ICMP	98 Echo (ping) request id=0x8ff5, seq=2/512, ttl=64 (no respons...
9	43.982582832	10.0.0.75	10.0.0.11	ICMP	98 Echo (ping) request id=0x8ff5, seq=3/768, ttl=64 (no respons...
10	47.028970192	00:00:00_00:01:11	00:00:00_00:02:21	ARP	42 Who has 10.0.0.75? Tell 10.0.0.11
11	47.052381210	00:00:00_00:02:21	00:00:00_00:01:11	ARP	42 Who has 10.0.0.11? Tell 10.0.0.75
12	47.076013039	00:00:00_00:01:11	00:00:00_00:02:21	ARP	42 10.0.0.11 is at 00:00:00:00:01:11
13	47.106006406	00:00:00_00:02:21	00:00:00_00:01:11	ARP	42 10.0.0.75 is at 00:00:00:00:02:21

Figura 26: Captura Wireshark Raspberry1 h11

Fuente: Propia

Como se puede observar, el cortafuegos no ha permitido que se haya ni intentado el envío del ping de H11 dirección H21. Los mensajes ICMP son desde H21 hacia H11, ya que esta conexión sí funciona porque no se ha especificado en el documento con el que ha arrancado que tampoco se permitiese el envío desde H21 hacia H11, para ello habría que declarar una regla nueva estableciendo como *IP_SRC* H21 e *IP_DST* H11. En la *Figura 26* es la misma situación que se ha descrito, pero desde el punto de vista de la terminal h21.

1	0.000000000	00:00:00_00:01:11	00:00:00_00:02:21	ARP	42 Who has 10.0.0.75? Tell 10.0.0.11
2	0.027985393	00:00:00_00:02:21	00:00:00_00:01:11	ARP	42 10.0.0.75 is at 00:00:00:00:02:21
3	9.652985596	0.0.0.0	255.255.255.255	DHCP	384 DHCP Discover - Transaction ID 0x79c9f494
4	9.931797243	0.0.0.0	255.255.255.255	DHCP	384 DHCP Discover - Transaction ID 0xe9c4618e
5	18.964309343	0.0.0.0	255.255.255.255	DHCP	384 DHCP Discover - Transaction ID 0xd00f5cbc
6	19.560604885	0.0.0.0	255.255.255.255	DHCP	384 DHCP Discover - Transaction ID 0xdce4e757
7	41.942786654	10.0.0.75	10.0.0.11	ICMP	98 Echo (ping) request id=0x8ff5, seq=1/256, ttl=64 (no respons...
8	42.917181018	10.0.0.75	10.0.0.11	ICMP	98 Echo (ping) request id=0x8ff5, seq=2/512, ttl=64 (no respons...
9	43.956667903	10.0.0.75	10.0.0.11	ICMP	98 Echo (ping) request id=0x8ff5, seq=3/768, ttl=64 (no respons...
10	47.017640393	00:00:00_00:01:11	00:00:00_00:02:21	ARP	42 Who has 10.0.0.75? Tell 10.0.0.11
11	47.028219759	00:00:00_00:02:21	00:00:00_00:01:11	ARP	42 Who has 10.0.0.11? Tell 10.0.0.75
12	47.059097200	00:00:00_00:01:11	00:00:00_00:02:21	ARP	42 10.0.0.11 is at 00:00:00:00:01:11
13	47.082249363	00:00:00_00:02:21	00:00:00_00:01:11	ARP	42 10.0.0.75 is at 00:00:00:00:02:21

Figura 27: Captura Wireshark Raspberry2 h21

Fuente: Propia

Para aplicar los cambios en el documento de arranque, se requiere reiniciar el controlador para que instale las reglas que se hayan añadido tras la ejecución del controlador.

El segundo método consiste en crear una nueva regla en el controlador mediante el **Shell de mininet**. Para ello, se debe utilizar esta línea de código, que en este ejemplo se ha empleado en la topología de la Raspberry 4:

```
mininet> sh ovs-ofctl add-flow s21
dl_src=00:00:00:00:01:12,dl_dst=00:00:00:00:02:23,actions=drop

mininet> sh ovs-ofctl add-flow s21
dl_src=00:00:00:00:02:23,dl_dst=00:00:00:00:01:12,actions=drop
```

En este fragmento, se está indicando que esta regla se añada al conmutador s21 y que todos aquellos paquetes que se envíen desde el dispositivo con la MAC: 00:00:00:00:01:12, que es H12, hacia el dispositivo con la MAC destino 00:00:00:00:02:23, que es H23, serán descartados. Para comprobar que no se han recibido los paquetes en el otro dispositivo, se puede ver mediante Wireshark realizando un ping entre los dispositivos tal y como se han realizado en los apartados anteriores.

1	0.0000000000	0.0.0.0	255.255.255.255	DHCP	384 DHCP Discover - Transaction ID 0x14aaa40f
2	1.563177576	0.0.0.0	255.255.255.255	DHCP	384 DHCP Discover - Transaction ID 0xf9f4a5ae
3	7.943200361	0.0.0.0	255.255.255.255	DHCP	384 DHCP Discover - Transaction ID 0xe9c4618e
4	8.058097284	0.0.0.0	255.255.255.255	DHCP	384 DHCP Discover - Transaction ID 0x79c9f494
5	33.787645886	10.0.0.12	10.0.0.77	ICMP	98 Echo (ping) request id=0xe6b5, seq=1/256, ttl=64 (no respons...
6	34.834360610	10.0.0.12	10.0.0.77	ICMP	98 Echo (ping) request id=0xe6b5, seq=2/512, ttl=64 (no respons...
7	35.867655296	10.0.0.12	10.0.0.77	ICMP	98 Echo (ping) request id=0xe6b5, seq=3/768, ttl=64 (no respons...
8	38.832977122	00:00:00_00:01:12	00:00:00_00:02:23	ARP	42 Who has 10.0.0.77? Tell 10.0.0.12
9	39.869957582	00:00:00_00:01:12	00:00:00_00:02:23	ARP	42 Who has 10.0.0.77? Tell 10.0.0.12
10	40.908855792	00:00:00_00:01:12	00:00:00_00:02:23	ARP	42 Who has 10.0.0.77? Tell 10.0.0.12

Figura 28: Captura Wireshark Raspberry2 h23

Fuente: Propia

Al igual que en la prueba anterior, se ha cortado la conexión de H23 hacia H12, por lo que el controlador no le permite enviar ningún paquete a H12, pero sí puede recibirlos. Para poder cortar toda la comunicación entre las dos terminales, habría que ejecutar el comando anterior en aquellos conmutadores que se encuentren conectados a las terminales H23 y H12.

1	0.00000000	0.0.0.0	255.255.255.255	DHCP	384 DHCP Discover - Transaction ID 0x14aaa40f
2	1.564988017	0.0.0.0	255.255.255.255	DHCP	384 DHCP Discover - Transaction ID 0xf9f4a5ae
3	7.936926212	0.0.0.0	255.255.255.255	DHCP	384 DHCP Discover - Transaction ID 0xe9c4618e
4	8.051328930	0.0.0.0	255.255.255.255	DHCP	384 DHCP Discover - Transaction ID 0x79c9f494
5	33.780936128	10.0.0.12	10.0.0.77	ICMP	98 Echo (ping) request id=0xe6b5, seq=1/256, ttl=64 (no respons...
6	34.822617421	10.0.0.12	10.0.0.77	ICMP	98 Echo (ping) request id=0xe6b5, seq=2/512, ttl=64 (no respons...
7	35.862273661	10.0.0.12	10.0.0.77	ICMP	98 Echo (ping) request id=0xe6b5, seq=3/768, ttl=64 (no respons...
8	38.824984122	00:00:00_00:01:12	00:00:00_00:02:23	ARP	42 Who has 10.0.0.77? Tell 10.0.0.12
9	39.862513035	00:00:00_00:01:12	00:00:00_00:02:23	ARP	42 Who has 10.0.0.77? Tell 10.0.0.12
10	40.902048283	00:00:00_00:01:12	00:00:00_00:02:23	ARP	42 Who has 10.0.0.77? Tell 10.0.0.12

Figura 29: Captura Wireshark Raspberry2 h23

Fuente: Propia

Hay que tener en cuenta que ovs-ofctl no trabaja con direcciones IP, por lo que para esta forma de añadir reglas al firewall se debe trabajar con las direcciones MAC. Para poder trabajar con direcciones IP se tiene que utilizar “ovs-appctl”, pero este no permite añadir reglas de flujo OpenFlow en un OVS.

En caso de querer permitir una conexión que se encuentra cortada mediante el cortafuegos, habría que cambiar el comando anterior poniendo “actions=normal” al final en lugar de “actions=drop”.

5.3 Spanning Tree Protocol

En una red con múltiples enlaces, existe la posibilidad de que se produzcan bucles que puedan afectar negativamente al rendimiento y a la eficiencia de la red provocando pérdidas de paquetes. Para solucionar este problema, se ha desarrollado el protocolo de expansión en árbol (Spanning Tree Protocol, STP), que se encarga de eliminar los bucles al cortar enlaces entre los elementos de la red mediante software creando una topología en forma de árbol.

En esta prueba, se evaluará la efectividad del protocolo STP en la topología presentada a continuación, donde se han creado varios enlaces con bucles:

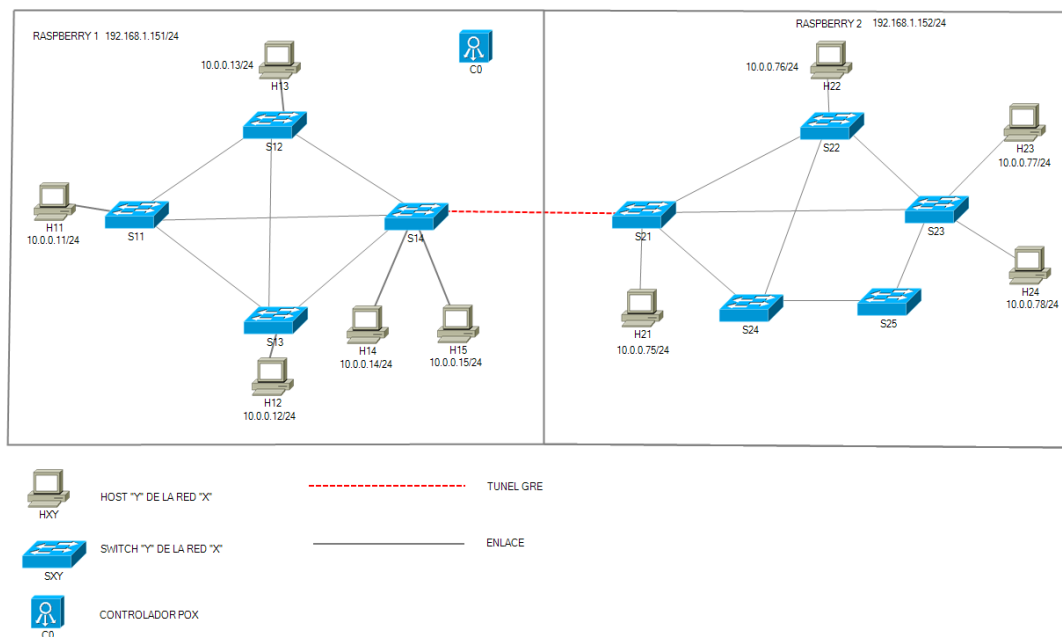


Figura 30: Topología Spanning Tree Protocol

Fuente: Propia

En esta topología, se dispone de varios conmutadores conectados de forma que generan bucles y varias terminales conectadas a esos conmutadores. La idea consiste en que una vez se ejecute el controlador, este analizará la topología y cortará los enlaces que considere innecesarios para acabar creando una estructura que imposibilita la existencia de bucles.

El comando para activar el controlador y que utilice el protocolo STP es el siguiente:

```
./pox2/pox.py forwarding.l2_learning openflow.discovery  
--eat-early-packets openflow.spanning_tree --no-flood -  
-hold-down log --format="[%(asctime)s] %(message)s" --  
datefmt="%H:%M:%S"
```

Mediante el comando “eat early packets”, se indica al resumen que omita todos los paquetes que se realicen al iniciar la topología, ya que son irrelevantes para lo que se pretende comprobar con esta prueba y mediante el comando “format” se pretende mostrar la información de una manera que sea clara y sencilla para el usuario y en caso de no utilizar ese comando, se emitirá una gran cantidad de información bastante compleja y que puede resultar complicada de seguir. Tras la ejecución de este controlador, la consola en la que se ejecuta el controlador se mostrará de la siguiente forma:

```
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
[00:27:21] POX 0.5.0 (eel) is up.
[00:27:29] [00-00-00-00-00-0b 2] connected
[00:27:29] [00-00-00-00-00-0c 3] connected
[00:27:29] [00-00-00-00-00-0d 4] connected
[00:27:30] [00-00-00-00-00-0e 5] connected
[00:27:30] link detected: 00-00-00-00-00-0b.4 -> 00-00-00-00-00-0e.5
[00:27:31] link detected: 00-00-00-00-00-0b.2 -> 00-00-00-00-00-0c.2
[00:27:31] link detected: 00-00-00-00-00-0b.3 -> 00-00-00-00-00-0d.2
[00:27:31] link detected: 00-00-00-00-00-0c.4 -> 00-00-00-00-00-0d.3
[00:27:32] link detected: 00-00-00-00-00-0c.2 -> 00-00-00-00-00-0b.2
[00:27:32] link detected: 00-00-00-00-00-0c.3 -> 00-00-00-00-00-0e.3
[00:27:32] link detected: 00-00-00-00-00-0d.4 -> 00-00-00-00-00-0e.4
[00:27:33] link detected: 00-00-00-00-00-0d.2 -> 00-00-00-00-00-0b.3
[00:27:33] link detected: 00-00-00-00-00-0d.3 -> 00-00-00-00-00-0c.4
[00:27:34] link detected: 00-00-00-00-00-0e.4 -> 00-00-00-00-00-0d.4
[00:27:34] link detected: 00-00-00-00-00-0e.5 -> 00-00-00-00-00-0b.4
[00:27:35] link detected: 00-00-00-00-00-0e.3 -> 00-00-00-00-00-0c.3
[00:27:35] 4 ports changed
[00:27:35] 2 ports changed
[00:27:35] 2 ports changed
[00:27:36] 4 ports changed
[00:28:00] [00-00-00-00-00-15 7] connected
```

Figura 31: Captura del controlador en la prueba STP

Fuente: Propia

Para comprobar los enlaces que se han mantenido tras la comprobación del controlador, debemos ejecutar el siguiente comando de Mininet en ambas topologías:

```
mininet> net
mininet> dpctl dump-ports-desc
```

```
mininet> net
h11 h11-eth0:s11-eth1
h12 h12-eth0:s13-eth1
h13 h13-eth0:s12-eth1
h14 h14-eth0:s14-eth1
h15 h15-eth0:s14-eth2
s11 lo: s11-eth1:h11-eth0 s11-eth2:s12-eth2 s11-eth3:s13-eth2 s11-eth4:s14-eth5
s12 lo: s12-eth1:h13-eth0 s12-eth2:s11-eth2 s12-eth3:s14-eth3 s12-eth4:s13-eth3
s13 lo: s13-eth1:h12-eth0 s13-eth2:s11-eth3 s13-eth3:s12-eth4 s13-eth4:s14-eth4
s14 lo: s14-eth1:h14-eth0 s14-eth2:h15-eth0 s14-eth3:s12-eth3 s14-eth4:s13-eth4 s14-eth5:s11-eth4 s14-tfg2:
c0
```

** s11	** s12	** s13	** s14
<pre> OFFST_PORT_DESC reply (xid=0x2): 1(s11-eth1): addr:6a:7a:bd:18:92:f2 config: 0 state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 2(s11-eth2): addr:0e:c1:38:c8:6a:1b config: 0 state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 3(s11-eth3): addr:5a:6a:3b:3e:c8:0b config: 0 state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 4(s11-eth4): addr:ea:4c:d2:4c:15:71 config: 0 state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max LOCAL(s11): addr:2a:b0:0f:0f:7b:4a config: 0 state: 0 speed: 0 Mbps now, 0 Mbps max </pre>	<pre> OFFST_PORT_DESC reply (xid=0x2): 1(s12-eth1): addr:fe:e5:51:82:11:e9 config: 0 state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 2(s12-eth2): addr:82:54:3b:9a:a6:9b config: 0 state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 3(s12-eth3): addr:c2:94:e5:c0:d4:11 config: NO_FLOOD state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 4(s12-eth4): addr:7e:e4:1c:64:da:e5 config: NO_FLOOD state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max LOCAL(s12): addr:7a:0f:4c:20:5d:43 config: 0 state: 0 speed: 0 Mbps now, 0 Mbps max </pre>	<pre> OFFST_PORT_DESC reply (xid=0x2): 1(s13-eth1): addr:a6:32:fb:88:70:7c config: 0 state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 2(s13-eth2): addr:fe:4a:be:16:08:9e config: 0 state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 3(s13-eth3): addr:92:d9:52:2d:79:08 config: NO_FLOOD state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 4(s13-eth4): addr:02:c1:e0:37:e6:d1 config: NO_FLOOD state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max LOCAL(s13): addr:92:49:74:1e:e1:4b config: 0 state: 0 speed: 0 Mbps now, 0 Mbps max </pre>	<pre> OFFST_PORT_DESC reply (xid=0x2): 1(s14-eth1): addr:82:a3:b9:93:6f:51 config: 0 state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 2(s14-eth2): addr:d6:82:8d:7c:c4:54 config: 0 state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 3(s14-eth3): addr:52:ce:58:d9:a9:fc config: NO_FLOOD state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 4(s14-eth4): addr:86:ab:19:9d:51:f8 config: NO_FLOOD state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 5(s14-eth5): addr:d2:57:1d:e7:d7:2d config: 0 state: 0 current: 10GB-FD COPPER </pre>

Figura 32: Capturas de los enlaces cortados en la topología 1

Fuente: Propia

Aquí se encuentran las salidas de la consola del fragmento de la topología 2 tras llamar a los mismos comandos:

```
h21 h21-eth0:s21-eth1
h22 h22-eth0:s22-eth1
h23 h23-eth0:s23-eth1
h24 h24-eth0:s23-eth2
s21 lo: s21-eth1:h21-eth0 s21-eth2:s22-eth2 s21-eth3:s23-eth3 s21-eth4:s24-eth1 s21-tfg1:
s22 lo: s22-eth1:h22-eth0 s22-eth2:s21-eth2 s22-eth3:s23-eth4 s22-eth4:s24-eth4
s23 lo: s23-eth1:h23-eth0 s23-eth2:h24-eth0 s23-eth3:s21-eth3 s23-eth4:s22-eth3 s23-eth5:s24-eth2 s23-eth6:s25-eth1
s24 lo: s24-eth1:s21-eth4 s24-eth2:s23-eth5 s24-eth3:s25-eth2 s24-eth4:s22-eth4
s25 lo: s25-eth1:s23-eth6 s25-eth2:s24-eth3
c0
```

** s21	** s22	** s23	** s24	** s25
<pre> OFFST_PORT_DESC reply (xid=0x2): 1(s21-eth1): addr:02:5a:f4:d1:c9:e8 config: 0 state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 2(s21-eth2): addr:9a:9a:9c:45:75:24 config: 0 state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 3(s21-eth3): addr:22:b4:fc:b1:3f:14 config: 0 state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 4(s21-eth4): addr:12:7e:0e:63:e9:7f config: 0 state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 5(s21-tfg1): addr:7a:87:15:ce:c1:50 config: 0 state: 0 speed: 0 Mbps now, 0 Mbps max LOCAL(s21): addr:c2:db:9e:59:d8:45 config: 0 state: 0 speed: 0 Mbps now, 0 Mbps max </pre>	<pre> OFFST_PORT_DESC reply (xid=0x2): 1(s22-eth1): addr:9e:1c:c3:d0:6c:ad config: 0 state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 2(s22-eth2): addr:1a:03:d0:f7:16:23 config: 0 state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 3(s22-eth3): addr:52:6f:7b:37:03:ab config: NO_FLOOD state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 4(s22-eth4): addr:1e:df:f2:e2:8e:b1 config: NO_FLOOD state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max LOCAL(s22): addr:46:c3:26:47:7c:44 config: 0 state: 0 speed: 0 Mbps now, 0 Mbps max </pre>	<pre> OFFST_PORT_DESC reply (xid=0x2): 1(s23-eth1): addr:82:5e:2e:fb:32:1e config: 0 state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 2(s23-eth2): addr:0e:b9:d4:0f:af:7a config: 0 state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 3(s23-eth3): addr:66:5e:33:07:a6:0a config: 0 state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 4(s23-eth4): addr:96:ab:70:37:6d:7c config: NO_FLOOD state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 5(s23-eth5): addr:b2:03:14:31:ac:cc config: NO_FLOOD state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 6(s23-eth6): addr:8a:7e:0e:3e:2e:f9 config: 0 state: 0 </pre>	<pre> OFFST_PORT_DESC reply (xid=0x2): 1(s24-eth1): addr:26:c7:45:03:c8:24 config: 0 state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 2(s24-eth2): addr:96:ab:02:f0:0f:1e config: NO_FLOOD state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 3(s24-eth3): addr:82:34:05:07:f4:73 config: NO_FLOOD state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 4(s24-eth4): addr:06:65:f7:37:2c:96 config: NO_FLOOD state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max LOCAL(s24): addr:fe:f2:23:58:cf:47 config: 0 state: 0 speed: 0 Mbps now, 0 Mbps max </pre>	<pre> OFFST_PORT_DESC reply (xid=0x2): 1(s25-eth1): addr:96:d8:e1:bb:39:98 config: 0 state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max 2(s25-eth2): addr:86:a0:eb:0d:1a:ca config: NO_FLOOD state: 0 current: 10GB-FD COPPER speed: 10000 Mbps now, 0 Mbps max LOCAL(s25): addr:82:85:6c:20:23:46 config: 0 state: 0 speed: 0 Mbps now, 0 Mbps max </pre>

Figura 33: Capturas de los enlaces cortados en la topología 2

Fuente: Propia

Mediante el primer comando, se obtiene el nombre de las interfaces de cada componente de la topología, el cual se utiliza para poder seguir cuáles son los enlaces que se han cortado.

Utilizando el segundo comando, se muestran separados cada conmutador indicando cuáles son las conexiones que se han cortado ya que se muestra en el apartado “config:” de la salida de consola el estado “NO_FLOOD” que, en caso de no estar desconectado, aparecería con un “0” indicando que ese enlace está siendo utilizado. La red resultante con las modificaciones realizadas por el controlador sería la siguiente:

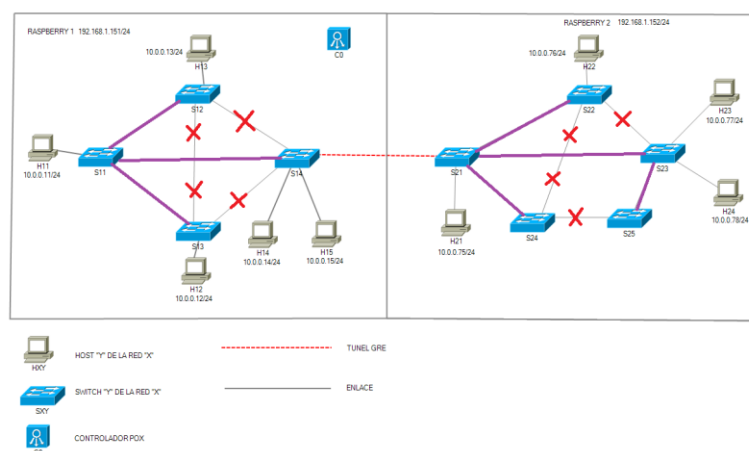


Figura 34: Árbol resultante del STP

Fuente: Propia

Las líneas marcadas en púrpura indican la estructura en árbol resultante del controlador aplicando el protocolo STP, cortando los enlaces marcados con una X roja.

Ahora, se va a comprobar cómo responde a los cambios el controlador cortando uno de los enlaces que se encuentran activos y por tanto obligándole a reestructurar el árbol de nuevo.

Para forzar el cambio, se cortará el enlace entre S11 y S14 mediante el siguiente comando:

```
link s11 s14 down
```

Tras modificar el estado de uno de los enlaces que se encontraban activos según el controlador, este realizará otro cambio en los enlaces que se encontraban activos para reorganizar todo y así actualizar el árbol teniendo en cuenta el enlace que acabamos de cortar. Esto se puede observar en la consola, donde se indica que ha habido una modificación de una serie de enlaces:

```
[01:46:59] link detected: 00-00-00-00-00-0e.5 -> 00-00-00-00-00-0b.4
[01:46:59] 4 ports changed
[01:47:00] link detected: 00-00-00-00-00-0b.4 -> 00-00-00-00-00-0e.5
[01:47:17] link timeout: 00-00-00-00-00-0e.5 -> 00-00-00-00-00-0b.4
[01:47:17] link timeout: 00-00-00-00-00-0b.4 -> 00-00-00-00-00-0e.5
```

Figura 35: Mensaje indicando cambio en los enlaces

Fuente: Propia

Para comprobar la estructura del nuevo árbol, se realiza otra vez el comando para volcar la información de los enlaces disponibles como se ha realizado anteriormente:

*** s11 ***	*** s12 ***	*** s13 ***	*** s14 ***
OFPT_PORT_DESC reply (xid=0x2):	OFPT_PORT_DESC reply (xid=0x2):	OFPT_PORT_DESC reply (xid=0x2):	OFPT_PORT_DESC reply (xid=0x2):
1(s11-eth1): addr:6a:7a:bd:18:92:f2	1(s12-eth1): addr:fe:e5:51:82:11:e9	1(s13-eth1): addr:a6:32:fb:88:70:7c	1(s14-eth1): addr:82:a3:b9:93:6f:51
config: 0	config: 0	config: 0	config: 0
state: 0	state: 0	state: 0	state: 0
current: 10GB-FD COPPER	current: 10GB-FD COPPER	current: 10GB-FD COPPER	current: 10GB-FD COPPER
speed: 10000 Mbps now, 0 Mbps max	speed: 10000 Mbps now, 0 Mbps max	speed: 10000 Mbps now, 0 Mbps max	speed: 10000 Mbps now, 0 Mbps max
2(s11-eth2): addr:0e:c1:38:c8:6a:1b	2(s12-eth2): addr:82:54:3b:9a:a6:9b	2(s13-eth2): addr:fe:4a:be:16:88:9e	2(s14-eth2): addr:d6:82:8d:7c:c4:54
config: 0	config: 0	config: 0	config: 0
state: 0	state: 0	state: 0	state: 0
current: 10GB-FD COPPER	current: 10GB-FD COPPER	current: 10GB-FD COPPER	current: 10GB-FD COPPER
speed: 10000 Mbps now, 0 Mbps max	speed: 10000 Mbps now, 0 Mbps max	speed: 10000 Mbps now, 0 Mbps max	speed: 10000 Mbps now, 0 Mbps max
3(s11-eth3): addr:5a:6a:3b:3e:c8:0b	3(s12-eth3): addr:c2:94:e5:c0:d4:11	3(s13-eth3): addr:92:d9:52:2d:79:08	3(s14-eth3): addr:52:ce:58:d9:a9:fc
config: 0	config: 0	config: NO_FLOOD	config: 0
state: 0	state: 0	state: 0	state: 0
current: 10GB-FD COPPER	current: 10GB-FD COPPER	current: 10GB-FD COPPER	current: 10GB-FD COPPER
speed: 10000 Mbps now, 0 Mbps max	speed: 10000 Mbps now, 0 Mbps max	speed: 10000 Mbps now, 0 Mbps max	speed: 10000 Mbps now, 0 Mbps max
4(s11-eth4): addr:ea:4c:d2:4c:15:71	4(s12-eth4): addr:7e:e4:1c:64:da:e5	4(s13-eth4): addr:02:c1:e0:37:e6:d1	4(s14-eth4): addr:86:ab:19:9d:51:f8
config: PORT_DOWN NO_FLOOD	config: NO_FLOOD	config: NO_FLOOD	config: NO_FLOOD
state: LINK_DOWN	state: 0	state: 0	state: 0
current: 10GB-FD COPPER	current: 10GB-FD COPPER	current: 10GB-FD COPPER	current: 10GB-FD COPPER
speed: 10000 Mbps now, 0 Mbps max	speed: 10000 Mbps now, 0 Mbps max	speed: 10000 Mbps now, 0 Mbps max	speed: 10000 Mbps now, 0 Mbps max
LOCAL(s11): addr:2a:b0:0f:f0:7b:4a	LOCAL(s12): addr:7a:0f:4c:20:5d:43	LOCAL(s13): addr:92:49:74:1e:e1:4b	5(s14-eth5): addr:d2:57:1d:e7:d7:2d
config: 0	config: 0	config: 0	config: PORT_DOWN NO_FLOOD
state: 0	state: 0	state: 0	state: LINK_DOWN
speed: 0 Mbps now, 0 Mbps max	speed: 0 Mbps now, 0 Mbps max	speed: 0 Mbps now, 0 Mbps max	current: 10GB-FD COPPER

Figura 36: Captura de los enlaces cortados tras desactivar un enlace

Fuente: Propia

Estas salidas por consola se traducen en la siguiente modificación del árbol:

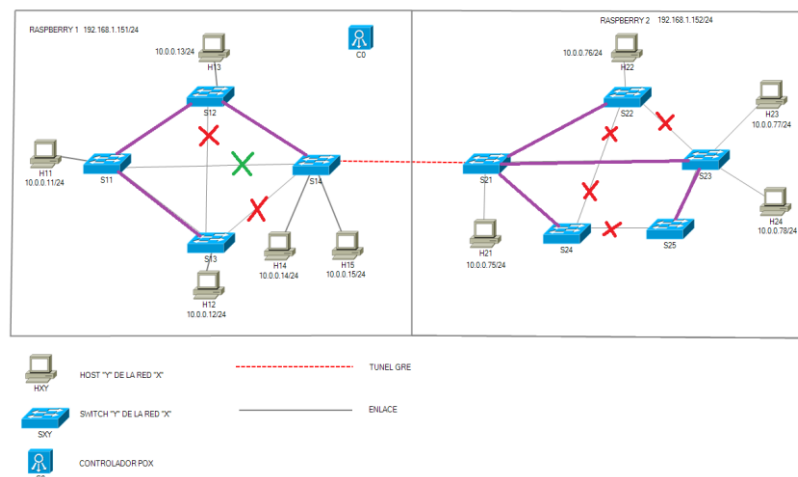


Figura 37: Árbol resultante del STP tras la modificación

Fuente: Propia

Como se puede apreciar, la estructura del árbol ha cambiado respondiendo al estado del nuevo enlace caído, que está marcado mediante una X verde, permitiendo que no se haya perdido conectividad y que se sigan evitando los bucles en la topología.

5.4 Problemas durante el desarrollo del proyecto

En este proyecto, se ha tenido que instalar una versión anterior del controlador ya que había un problema con el más actual al utilizar Python3 cuando he realizado el proyecto que causaba un error a la hora de calcular el checksum de los paquetes que viajaban por la red. Por lo que se decidió utilizar la versión más actual que utilizase Python2, que es “eel”. Para poder elegir una versión anterior, se tiene que emplear el comando `--branch` “nombre de la versión”. Si no se indica nada, elegirá por defecto la versión que esté marcada como la principal, que a fecha de esta documentación es la versión “gar-experimental”.

```
git clone --branch eel https://github.com/noxrepo/pox
```

Al estar mininet especializado en trabajar con una topología en un único dispositivo, algunos de los mayores problemas han venido a la hora de utilizar algunos comandos que no estaban previstos para funcionar con una topología fragmentada en diferentes máquinas como se ha tratado de simular en este proyecto. Hay varios que solo funcionan con la red de la topología en la que se ejecutan y no tienen en cuenta las otras topologías que se encuentran en las otras máquinas y otros tienen problemas con los nombres de los dispositivos como pueden ser los comandos para el QoS que permiten modificar el ancho de banda entre varias terminales, la velocidad de transferencia, y otros que no ha sido posible realizar debido a esta limitación. El comando “ping” permitía utilizar en lugar del nombre del dispositivo su dirección IP, pero otros no permitían el uso de IP a la hora de ejecutar los comandos. Es por eso que tras varias pruebas como intentar crear un servidor *DNS* con programas como *DNSMasq* o *BIND* (Berkeley Internet Name Domain), intentar crear un archivo JSON que se modificase con la información de red que tiene el controlador acerca de todas las topologías, se ha optado por modificar el fichero “**etc/hosts**” de cada Raspberry. En cada una de las Raspberry se ha modificado el fichero para que tengan guardadas las IP de los dispositivos que se emulan en las otras topologías siguiendo esta forma:

IPEquipoEnOtraTopologia	aliasDelDispositivo
-------------------------	---------------------

6. Conclusiones

Para la realización de este proyecto, se ha obtenido información teórica acerca del funcionamiento de las Redes *SDN*, desde la división de la red en varios planos como son el plano de control, en el cual se realiza toda la gestión de la red y se encuentra alojado el controlador encargado de implementar los cambios solicitados por el usuario y obtener información acerca de la red para tener un correcto funcionamiento, y el plano de datos, donde se encuentran los diferentes elementos que componen la red como los conmutadores *SDN*, los enlaces, enrutadores; Esta división, es la que permite diferenciarlas en diversos apartados respecto a las redes convencionales como son la escalabilidad al no incrementar mucho la complejidad de la gestión de la red cuanto mayor sea su tamaño, poder visualizar la red en su totalidad ya que el controlador dispone de toda la información del plano de datos, y también que las herramientas y protocolos utilizados para gestionar estas redes disponen de código abierto para poder ayudar a futuros desarrolladores a mejorar este tipo de controladores, los protocolos *OpenFlow* que emplean o a poder facilitar la creación de herramientas que utilicen la información del controlador para asistir con las tareas administrativas de la red.

Es importante destacar que se ha comprobado la sencillez acerca de utilizar este tipo de redes para gestionar cualquier tipo de topologías, ya que el controlador permite poder manejar muchas topologías diferentes sin aumentar mucho la complejidad al no necesitar configurar todos y cada uno de los componentes que componen la red con elementos como son una dirección *IP*, una dirección *MAC*, las tablas de direccionamiento de los paquetes y otra serie de factores que, en caso de querer configurarlos, también se permite hacerlo de una forma cómoda y sin mucha complejidad.

El objetivo principal de este proyecto ha consistido en utilizar máquinas Raspberry de manera que actuaran como conmutadores *SDN*. Se ha conseguido cumplir ese objetivo mediante el empleo de túneles *GRETAP* entre las diferentes máquinas. Se ha documentado el proceso desde la instalación de las diferentes herramientas empleadas para la gestión y el control de las Raspberry desde la instalación del S.O. hasta el empleo de escritorios remotos para poder manejar el entorno de pruebas de una forma cómoda e incluso que podría llegar a escalarse a más de dos máquinas para poder simular una red de mayores dimensiones. También sería posible emplear otro tipo de máquinas para poder simular la red como por ejemplo espacios virtuales, pero estos podrían requerir de una mayor complejidad a la hora de configurarlos desde cero.

Se ha procedido a realizar diferentes pruebas sobre esa red para demostrar que es posible conectar varias topologías con un resultado satisfactorio. Las pruebas se encuentran debidamente documentadas y han consistido en la comprobación de que se podrían conectar diferentes topologías utilizando Mininet, cómo poder adaptar los comandos que realiza Mininet a ese entorno debido a sus limitaciones de estar desarrollado para emular topologías en una sola máquina. Otras de las pruebas han consistido en un uso del controlador *POX* a modo de cortafuegos para limitar las conexiones entre dispositivos de la propia red simulada de forma unidireccional y bidireccional ya sea en tiempo de ejecución de las topologías como predefiniendo una serie de reglas para cancelar esas conexiones una vez haya comenzado a funcionar la red. Para la última batería de pruebas se ha utilizado el protocolo *STP* para evitar la creación de bucles en la topología comprobando una vez se haya ejecutado que todos los dispositivos se encuentran debidamente conectados y una vez la topología ya haya convergido con los enlaces que ha indicado el controlador, modificar enlaces de forma que actuase como si hubiese algún fallo en una de las conexiones y comprobar cómo reestructura los enlaces de modo que siga sin haber bucles y todo

continúe conectado dentro de la red demostrando así que gestiona de una forma correcta este tipo de errores.

Como futuros añadidos a este proyecto, se podrían intentar conectar un mayor número de dispositivos Raspberry Pi para así poder aumentar el tamaño de la red en la que realizar las pruebas, añadir otro controlador para comprobar cómo se sincronizan ambos controladores con la gestión de la red o incluso probar otros controladores diferentes al controlador *POX*, ya que hay un amplio repertorio de controladores como son *Ryu*, *Floodlight*, *ODL* o incluso *NOX* cada uno con cualidades que los diferencian de los otros como se ha explicado en apartados anteriores de la documentación. También se podrían intentar realizar algunas pruebas tratando de adaptar aquellos comandos de Mininet que no estaban preparados para trabajar con múltiples topologías conectadas entre sí, la creación con éxito de un servidor DNS para poder utilizar los comandos sin tener que configurar manualmente los alias y las *IP* asignadas a las terminales creadas en el despliegue de las diferentes topologías, o incluso buscar otras alternativas a Mininet para comprobar el funcionamiento de las pruebas realizadas en este proyecto como podría ser Shadow.

7. Bibliografía

- [1] R. J. Millán Tejedor, «SDN: el futuro de las redes inteligentes,» Septiembre 2014. [En línea]. Available: <https://www.ramonmillan.com/tutoriales/sdnredesinteligentes.php>. [Último acceso: 7 Noviembre 2022].
- [2] Z. Latif, K. Sharif, F. Li, M. M. Karim y Y. Wang, «A Comprehensive Survey of Interface Protocols for Software Defined Networks,» 2019. [En línea]. Available: https://www.researchgate.net/publication/331273642_A_Comprehensive_Survey_of_Interface_Protocols_for_Software_Defined_Networks. [Último acceso: 10 Marzo 2023].
- [3] D. Macedo, D. Guedes, L. Vleira, M. Vieira y M. Nogueira, «Programmable Networks – From Software Defined Radio to Software Defined Networking,» Abril 2015. [En línea]. Available: https://www.researchgate.net/publication/276369342_Programmable_Networks_-_From_Software_Defined_Radio_to_Software_Defined_Networking. [Último acceso: 12 Marzo 2023].
- [4] A. Tootoonchian, «GitHub - noxrepo/nox: The NOX Controller,» 12 Mayo 2012. [En línea]. Available: <https://github.com/noxrepo/nox>. [Último acceso: 26 Abril 2023].
- [5] M. McCauley et al., «GitHub - noxrepo/pox: The POX network software platform,» 10 Junio 2011. [En línea]. Available: <https://github.com/noxrepo/pox>. [Último acceso: 26 Abril 2023].
- [6] Ryu SDN Framework Community, «Ryu Resources,» 2014. [En línea]. Available: <https://ryu-sdn.org/resources.html>. [Último acceso: 29 Marzo 2023].

- [7] R. Izard et al., «floodlight@groups.io | Home,» 4 Marzo 2015. [En línea]. Available: <https://groups.io/g/floodlight>. [Último acceso: 26 Abril 2023].
- [8] D. Erickson, «Beacon - Confluence,» 12 Septiembre 2011. [En línea]. Available: <https://floodlight.atlassian.net/wiki/spaces/Beacon/overview>. [Último acceso: 29 Marzo 2023].
- [9] K. Kim y Y. Zhi, «GitHub - superkkt/cherry: OpenFlow Controller written in Go,» 27 Marzo 2015. [En línea]. Available: <https://github.com/superkkt/cherry>. [Último acceso: 26 Abril 2023].
- [10] OpenDaylight Project The Linux Foundation, «OpenDaylight,» 2013. [En línea]. Available: <https://www.opendaylight.org/>. [Último acceso: 11 Septiembre 2022].
- [11] Raspberry Pi Foundation, «Raspberry Pi 4 Model B specifications – Raspberry Pi,» [En línea]. Available: <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/>. [Último acceso: 17 Agosto 2022].
- [12] D. Haryachyy, «Learning POX OpenFlow controller : hub implementation | Denys Haryachyy,» 29 Mayo 2014. [En línea]. Available: <https://haryachyy.wordpress.com/2014/05/29/learning-pox-sdn-controller-hub-py-module/>. [Último acceso: 10 Diciembre 2022].
- [13] J. F. Kurose y K. W. Ross, Redes de computadoras. Un enfoque descendente, 7 Ed, Madrid: Pearson, 2017.
- [14] H. Noman y M. Jasim, «POX Controller and Open Flow Performance Evaluation in Software Defined Networks (SDN) Using Mininet Emulator,» 11 Agosto 2020. [En línea]. Available: https://www.researchgate.net/publication/343586922_POX_Controller_and_Open_Flow_Performance_Evaluation_in_Software_Defined_Networks_SDN_Using_Mininet_Emulator. [Último acceso: 05 Marzo 2023].
- [15] N. Feamster, «Examples of SDN Controllers,» 12 Junio 2014. [En línea]. Available: <https://www.youtube.com/watch?v=dpcw2XqLp-E>.

[Último acceso: 6 Septiembre 2022].

- [16] J. Machado, A. F. Ramos y J. C. Cuéllar, «Vista de Implementación de OpenFlow sobre NetFPGA | Ingeniería y Región,» 30 Abril 2014. [En línea]. Available: <https://journalusco.edu.co/index.php/iregion/article/view/743/1425>. [Último acceso: 12 Febrero 2023].
- [17] Gcinst, «Software-defined networking - Wikipedia,» 29 Marzo 2023. [En línea]. Available: https://en.wikipedia.org/w/index.php?title=Software-defined_networking&oldid=1147123298. [Último acceso: 29 Marzo 2023].
- [18] B. Lantz y B. Heller, «Mininet: An Instant Virtual Network on Your Laptop (or Other PC) - Mininet,» 9 Diciembre 2009. [En línea]. Available: <http://mininet.org/>. [Último acceso: 26 Abril 2023].
- [19] M. McCauley et al., «Installing POX POX Manual Current documentation,» 2015. [En línea]. Available: <https://noxrepo.github.io/pox-doc/html/>. [Último acceso: 26 Abril 2023].
- [20] R. Jansen et al., «The Shadow Network Simulator,» [En línea]. Available: <https://shadow.github.io/>. [Último acceso: 9 Mayo 2023].

8. Glosario

API: Interfaz de Programación de Aplicaciones (Application Programming Interfaces)

CSV: Valores separados por comas (Comma-separated Values)

DNS: Sistema de Nombres de Dominio (Domain Name Services)

GRE: Generic Routing Encapsulation Terminal Access Point

HTTP: Protocolo de Transferencia de Hipertexto (Hypertext Transfer Protocol)

MAC: Media Access Control

NETCONF: Network Configuration Protocol

OSGI: Iniciativa de Puerta de Enlace de Servicios Abiertos (Open Services Gateway Initiative)

OVSD: Open vSwitch Database Management Protocol

QoS: Calidad de Servicio (Quality of Service)

Router: Enrutador

REST: Representational State Transfer

Switch: Conmutador

TCP: Protocolo de Control de Transmisión (Transmission Control Protocol)

TTL: Tiempo de vida (Time To Live)

VNC: Computación Virtual en Red (Virtual Network Computing)

9. Anexos

Entorno1Ping.py

Este script se debe ejecutar en una terminal en el primer dispositivo utilizando el comando “sudo python Entorno1Ping.py” para realizar la prueba indicada en el apartado [5.1 Conexión entre las Raspberry Pi](#)

```
#!/usr/bin/python
from mininet.net import Mininet
from mininet.node import Controller, RemoteController,
OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():

    info('***Create a network...\n')
    #Se indica el tipo de conexion que sera Link y el
    tipo de conmutador que sera switchOvs
    net = Mininet( controller=Controller, link=TCLink,
switch=OVSKernelSwitch )

    #Se crea una instancia del controlador con la IP
    del dispositivo en el que se va a ejecutar
```

```
#el controlador
c0 = net.addController('c0',
controller=RemoteController, ip='192.168.1.151',
port=6633 )

info('*** Add switches\n')
#Se crea una instancia de Switch
s11 = net.addSwitch('s11')

info('*** Add hosts\n')
#Se crean los terminales de la topologia
h11 = net.addHost('h11',cls=Host,
ip='10.0.0.11/24')
h12 = net.addHost('h12',cls=Host,
ip='10.0.0.12/24')
h13 = net.addHost('h13',cls=Host,
ip='10.0.0.13/24')

info('*** Add links\n')
#Se vinculan las terminales al switch de la
topologia
net.addLink(s11,h11)
net.addLink(s11,h12)
net.addLink(s11,h13)

info('*** Starting controllers\n')
#Conecta la topologia al controlador
c0.start()
```

```
    info('*** Creating interfaces\n')
    #Se indica al conmutador cual es el controlador al
que debe conectarse
    s11.start([c0])
    #Creando el tunel GRE entre la raspberry actual y
raspberry2
    s11.cmd("ip link add s11-tfg2 type gretap local
192.168.1.151 remote 192.168.1.152 ttl 64")
    s11.cmd("ip link set s11-tfg2 up")
    #Anyadiendo las interfaces al switch de la
topologia
    Intf("s11-tfg4",node=s11)
    net.start()
    CLI(net)

    info('***Stopping network')
    #Elimina las interfaces que se han creado para esta
topologia
    s11.cmd("ip link del dev s11-tfg2")
    net.stop()

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

Entorno2Ping.py

Este script se debe ejecutar en una terminal en el segundo dispositivo utilizando el comando “sudo python Entorno2Ping.py” para realizar la prueba indicada en el apartado [5.1 Conexión entre las Raspberry Pi](#)

```
#!/usr/bin/python
from mininet.net import Mininet
from mininet.node import Controller, RemoteController,
OVSSwitch
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():

    info('***Create a network...\n')
    #Se indica el tipo de conexion que sera Link y el
    tipo de conmutador que sera switchOvs
    net = Mininet( controller=Controller, link=TCLink,
switch=OVSKernelSwitch )

    #Se crea una instancia del controlador con la IP
del dispositivo en el que se va a ejecutar
    #el controlador
    c0 = net.addController('c0',
```

```
controller=RemoteController, ip='192.168.1.151',
port=6633 )

    info('*** Add switches\n')
    #Se crea una instancia de Switch
    s21 = net.addSwitch('s21')

    info('*** Add hosts\n')
    #Se crean los terminales de la topologia
    h21 = net.addHost('h21',cls=Host,
ip='10.0.0.75/24')
    h22 = net.addHost('h22',cls=Host,
ip='10.0.0.76/24')
    h23 = net.addHost('h23',cls=Host,
ip='10.0.0.77/24')

    info('*** Add links\n')
    #Se vinculan las terminales al switch de la
topologia
    net.addLink(s21,h21)
    net.addLink(s21,h22)
    net.addLink(s21,h23)

    info('*** Starting controllers\n')
    #Conecta la topologia al controlador
    c0.start()

    info('*** Creating interfaces\n')
    #Se indica al conmutador cual es el controlador al
```

```
que debe conectarse
    s21.start([c0])
    #Creando el tunel GRE entre la raspberry actual y
raspberry1
    s21.cmd("ip link add s21-tfg1 type gretap local
192.168.1.152 remote 192.168.1.151 ttl 64")
    s21.cmd("ip link set s21-tfg1 up")
    #Anyadiendo las interfaces al switch de la
topologia
    Intf("s21-tfg1",node=s21)
    net.start()
    CLI(net)

    info('***Stopping network')
    #Elimina las interfaces que se han creado para esta
topologia
    s21.cmd("ip link del dev s21-tfg1")
    net.stop()

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

Entorno1Firewall.py

Este script se debe ejecutar en una terminal en el primer dispositivo utilizando el comando “sudo python Entorno1Firewall.py” para realizar la prueba indicada en el apartado [5.2 Cortafuegos](#)

```
#!/usr/bin/python
from mininet.net import Mininet
from mininet.node import Controller, RemoteController,
OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():

    info('***Create a network...\n')
    #Se indica el tipo de conexion que sera Link y el
    tipo de conmutador que sera switchOvs
    net = Mininet( controller=Controller, link=TCLink,
switch=OVSKernelSwitch )

    #Se crea una instancia del controlador con la IP
    del dispositivo en el que se va a ejecutar
    #el controlador
    c0 = net.addController('c0',
```



```
controller=RemoteController,          ip='192.168.1.151',
port=6633 )

    info('*** Add switches\n')
    #Se crea una instancia de Switch
    s11 = net.addSwitch('s11')

    info('*** Add hosts\n')
    #Se crean los terminales de la topologia
    h11          =          net.addHost('h11',cls=Host,
ip='10.0.0.11/24', mac="00:00:00:00:01:11")
    h12          =          net.addHost('h12',cls=Host,
ip='10.0.0.12/24', mac="00:00:00:00:01:12")
    h13          =          net.addHost('h13',cls=Host,
ip='10.0.0.13/24', mac="00:00:00:00:01:13")

    info('*** Add links\n')
    #Se vinculan las terminales al switch de la
topologia
    net.addLink(s11,h11)
    net.addLink(s11,h12)
    net.addLink(s11,h13)

    info('*** Starting controllers\n')
    #Conecta la topologia al controlador
    c0.start()

    info('*** Creating interfaces\n')
    #Se indica al conmutador cual es el controlador al
```

```
que debe conectarse
    s11.start([c0])
    #Creando el tunel GRE entre la raspberry actual y
raspberrry2
    s11.cmd("ip link add s11-tfg2 type gretap local
192.168.1.151 remote 192.168.1.152 ttl 64")
    s11.cmd("ip link set s11-tfg2 up")
    #Anyadiendo las interfaces al switch de la
topologia
    Intf("s11-tfg4",node=s11)
    net.start()
    CLI(net)

    info('***Stopping network')
    #Elimina las interfaces que se han creado para esta
topologia
    s11.cmd("ip link del dev s11-tfg2")
    net.stop()

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

Entorno2Firewall.py

Este script se debe ejecutar en una terminal en el segundo dispositivo utilizando el comando “sudo python Entorno2Firewall.py” para realizar la prueba indicada en el apartado [5.2 Cortafuegos](#)

```
#!/usr/bin/python
from mininet.net import Mininet
from mininet.node import Controller, RemoteController,
OVSSwitch
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():

    info('***Create a network...\n')
    #Se indica el tipo de conexion que sera Link y el
    tipo de conmutador que sera switchOvs
    net = Mininet( controller=Controller, link=TCLink,
switch=OVSKernelSwitch )

    #Se crea una instancia del controlador con la IP
del dispositivo en el que se va a ejecutar
    #el controlador
    c0 = net.addController('c0',
```

```
controller=RemoteController, ip='192.168.1.151',
port=6633 )

    info('*** Add switches\n')
    #Se crea una instancia de Switch
    s21 = net.addSwitch('s21')

    info('*** Add hosts\n')
    #Se crean los terminales de la topologia
    h21 = net.addHost('h21',cls=Host,
ip='10.0.0.75/24', mac="00:00:00:00:02:21")
    h22 = net.addHost('h22',cls=Host,
ip='10.0.0.76/24', mac="00:00:00:00:02:22")
    h23 = net.addHost('h23',cls=Host,
ip='10.0.0.77/24', mac="00:00:00:00:02:23")

    info('*** Add links\n')
    #Se vinculan las terminales al switch de la
topologia
    net.addLink(s21,h21)
    net.addLink(s21,h22)
    net.addLink(s21,h23)

    info('*** Starting controllers\n')
    #Conecta la topologia al controlador
    c0.start()

    info('*** Creating interfaces\n')
    #Se indica al conmutador cual es el controlador al
```

```
que debe conectarse
    s21.start([c0])
    #Creando el tunel GRE entre la raspberry actual y
    raspberry1
    s21.cmd("ip link add s21-tfg1 type gretap local
192.168.1.152 remote 192.168.1.151 ttl 64")
    s21.cmd("ip link set s21-tfg1 up")
    #Anyadiendo las interfaces al switch de la
topologia
    Intf("s21-tfg1",node=s21)
    net.start()
    CLI(net)

    info('***Stopping network')
    #Elimina las interfaces que se han creado para esta
topologia
    s21.cmd("ip link del dev s21-tfg1")
    net.stop()

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

firewall.py de <https://github.com/jashdesai95/SDN-OpenFlow-Pox-Firewall>

Este script se ejecuta en la llamada del controlador en una terminal del primer dispositivo diferente a la terminal en la que se está ejecutando el script de la topología “Entorno1Firewall.py” como se indica en el apartado [5.2 Cortafuegos](#).

```
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.revent import *
from pox.lib.util import dpidToStr
from pox.lib.addresses import EthAddr, IPAddr
import pox.lib.packet as pkt
from collections import namedtuple
import os
import csv

log = core.getLogger()
# For flow rule entries in the OpenFlow table
policyFile = "/pox2/pox/misc/firewallpolicies.csv"

class Firewall (EventMixin):

    def __init__ (self):
        self.listenTo(core.openflow)
        log.info("Enabling Firewall Module")
        # Our firewall table
        self.firewall = {}

    def sendRule (self, src, dst, duration = 0):
```

```
"""
    Drops this packet and optionally installs a
flow to continue
    dropping similar ones for a while
"""
    if not isinstance(duration, tuple):
        duration = (duration,duration)
    msg = of.ofp_flow_mod()
    match = of.ofp_match(dl_type = 0x800,
                        nw_proto = pkt.ipv4.ICMP_PROTOCOL)
    match.nw_src = IPAddr(src)
    match.nw_dst = IPAddr(dst)
    msg.match = match
    msg.idle_timeout = duration[0]
    msg.hard_timeout = duration[1]
    msg.priority = 10
    self.connection.send(msg)

# function that allows adding firewall rules into
the firewall table
def AddRule (self, src=0, dst=0, value=True):
    if (src, dst) in self.firewall:
        log.info("Rule already present drop: src %s
- dst %s", src, dst)
    else:
        log.info("Adding firewall rule drop: src %s
- dst %s", src, dst)
        self.firewall[(src, dst)]=value
        self.sendRule(src, dst, 10000)
```

```
# function that allows deleting firewall rules from
the firewall table
def DeleteRule (self, src=0, dst=0):
    try:
        del self.firewall[(src, dst)]
        sendRule(src, dst, 0)
        log.info("Deleting firewall rule drop: src
%s - dst %s", src, dst)
    except KeyError:
        log.error("Cannot find in rule drop src %s
- dst %s", src, dst)

def _handle_ConnectionUp (self, event):
    ''' Add your logic here ... '''
    self.connection = event.connection

    ifile = open(policyFile, "rb")
    reader = csv.reader(ifile)
    rownum = 0
    for row in reader:
        # Save header row.
        if rownum == 0:
            header = row
        else:
            colnum = 0
            for col in row:
                #print '%-8s: %s' %
(header[colnum], col)
```



```
        colnum += 1
        self.AddRule(row[1], row[2])
        rownum += 1
    ifile.close()

    log.info("Firewall rules installed on %s",
dpidToStr(event.dpid))

def launch ():
    '''
    Starting the Firewall module
    '''
    core.registerNew(Firewall)
```

Entorno1STP.py

Este script se debe ejecutar en una terminal en el primer dispositivo utilizando el comando “sudo python Entorno1STP.py” para realizar la prueba indicada en el apartado [5.3 Spanning Tree Protocol](#)

```
#!/usr/bin/python
from mininet.net import Mininet
from mininet.node import Controller, RemoteController,
OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
```

```
from subprocess import call

def myNetwork():

    info('***Create a network...\n')
    #Se indica el tipo de conexion que sera Link y el
    tipo de conmutador que sera switchOvs
    net = Mininet( controller=Controller, link=TCLink,
switch=OVSKernelSwitch )

    #Se crea una instancia del controlador con la IP
del dispositivo en el que se va a ejecutar
    #el controlador
    c0 = net.addController('c0',
controller=RemoteController, ip='192.168.1.151',
port=6633 )

    info('*** Add switches\n')
    #Se crea una instancia de Switch
    s11 = net.addSwitch('s11', mac="00:00:00:00:01:21")
    s12 = net.addSwitch('s12', mac="00:00:00:00:01:22")
    s13 = net.addSwitch('s13', mac="00:00:00:00:01:23")
    s14 = net.addSwitch('s14', mac="00:00:00:00:01:24")

    info('*** Add hosts\n')
    #Se crean los terminales de la topologia
    h11 = net.addHost('h11',cls=Host,
ip='10.0.0.11/24', mac="00:00:00:00:01:11")
    h12 = net.addHost('h12',cls=Host,
```

```
ip='10.0.0.12/24', mac="00:00:00:00:01:12")
    h13 = net.addHost('h13',cls=Host,
ip='10.0.0.13/24', mac="00:00:00:00:01:13")
    h14 = net.addHost('h14',cls=Host,
ip='10.0.0.14/24', mac="00:00:00:00:01:14")
    h15 = net.addHost('h15',cls=Host,
ip='10.0.0.15/24', mac="00:00:00:00:01:15")

    info('*** Add links\n')
    #Se vinculan las terminales al switch de la
topologia
    net.addLink(s11,h11)
    net.addLink(s12,h13)
    net.addLink(s13,h12)
    net.addLink(s14,h14)
    net.addLink(s14,h15)
    net.addLink(s11,s12)
    net.addLink(s11,s13)
    net.addLink(s12,s14)
    net.addLink(s12,s13)
    net.addLink(s13,s14)
    net.addLink(s11,s14)

    info('*** Starting controllers\n')
    #Conecta la topologia al controlador
    c0.start()

    info('*** Creating interfaces\n')
    #Se indica al conmutador cual es el controlador al
```

```
que debe conectarse
    s11.start([c0])
    s12.start([c0])
    s13.start([c0])
    s14.start([c0])
    #Creando el tunel GRE entre la raspberry actual y
raspberry2
    s14.cmd("ip link add s14-tfg2 type gretap local
192.168.1.151 remote 192.168.1.152 ttl 64")
    s14.cmd("ip link set s14-tfg2 up")
    #Anyadiendo las interfaces al switch de la
topologia
    Intf("s14-tfg2",node=s14)
    net.start()
    CLI(net)
    info('***Stopping network')
    #Elimina las interfaces que se han creado para esta
topologia
    s11.cmd("ip link del dev s14-tfg2")
    net.stop()
if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()
```

Entorno2STP.py

Este script se debe ejecutar en una terminal en el segundo dispositivo utilizando el comando “sudo python Entorno2STP.py” para realizar la prueba indicada en el apartado [5.3 Spanning Tree Protocol](#)

```
#!/usr/bin/python
from mininet.net import Mininet
from mininet.node import Controller, RemoteController,
OVSTController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():

    info('***Create a network...\n')
    #Se indica el tipo de conexion que sera Link y el
    tipo de conmutador que sera switchOvs
    net = Mininet( controller=Controller, link=TCLink,
switch=OVSKernelSwitch )

    #Se crea una instancia del controlador con la IP
    del dispositivo en el que se va a ejecutar
    #el controlador
    c0 = net.addController('c0',
controller=RemoteController, ip='192.168.1.151',
port=6633 )

    info('*** Add switches\n')
    #Se crea una instancia de Switch
```

```
s21 = net.addSwitch('s21')
s22 = net.addSwitch('s22')
s23 = net.addSwitch('s23')
s24 = net.addSwitch('s24')
s25 = net.addSwitch('s25')

info('*** Add hosts\n')
#Se crean los terminales de la topologia
h21 = net.addHost('h21',cls=Host,
ip='10.0.0.75/24')
h22 = net.addHost('h22',cls=Host,
ip='10.0.0.76/24')
h23 = net.addHost('h23',cls=Host,
ip='10.0.0.77/24')
h24 = net.addHost('h24',cls=Host,
ip='10.0.0.78/24')

info('*** Add links\n')
#Se vinculan las terminales al switch de la
topologia
net.addLink(s21,h21)
net.addLink(s22,h22)
net.addLink(s23,h23)
net.addLink(s23,h24)
net.addLink(s21,s22)
net.addLink(s21,s23)
net.addLink(s22,s23)
net.addLink(s21,s24)
net.addLink(s24,s23)
```

```
net.addLink(s25,s23)
net.addLink(s24,s25)
net.addLink(s22,s24)

info('*** Starting controllers\n')
#Conecta la topologia al controlador
c0.start()

info('*** Creating interfaces\n')
#Se indica al conmutador cual es el controlador al
que debe conectarse
s21.start([c0])
s22.start([c0])
s23.start([c0])
s24.start([c0])
s25.start([c0])

#Creando el tunel GRE entre la raspberry actual y
raspberrry1
s21.cmd("ip link add s21-tfg1 type gretap local
192.168.1.152 remote 192.168.1.151 ttl 64")
s21.cmd("ip link set s21-tfg1 up")

#Anyadiendo las interfaces al switch de la
topologia
Intf("s21-tfg1",node=s21)
net.start()
CLI(net)
```

```
    info('***Stopping network')
    #Elimina las interfaces que se han creado para esta
topologia
    s21.cmd("ip link del dev s21-tfg1")
    net.stop()

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