uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Telemática
Año 2017-2018

Trabajo de Fin de Grado

"Configuración de equipos de comunicaciones embarcados en MAVs mediante SDN"

Ana Fernández Santos

Tutor

Iván Vidal Fernández

Leganés, Octubre de 2018

TÍTULO

Configuración de equipos de comunicaciones embarcados en MAVs mediante SDN.

AUTOR

Ana Fernández Santos

TUTOR

Iván Vidal Fernández

EL TRIBUNAL

Presidente:

Vocal:

Secretario:

Realizado el acto de defensa y lectura del Trabajo de Fin de grado el día ……. de ……. de ……. en ……., en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de:

VOCAL

PRESIDENTE **SECRETARIO**

AGRADECIMIENTOS

A mis padres, Charo y Antonio, y a mi hermano Carlos. Por la paciencia y los ánimos siempre que los he necesitado en esta etapa y durante toda la vida.

A mis amigos de siempre. A María, por ser durante veinte años apoyo en todo momento. A Mamen, Elena, Encinas, Adolfo, Juanjo, Alba y Marisa, por no fallar nunca.

A mis amigos de la uni, Joni, Cristian, Madru, Alberto, Pedro, Javi y Andoni. Porque ha sido un camino duro pero siempre hemos sabido ayudarnos y pasarlo muy bien.

A mis chicas del Erasmus, María, Andrea, Mayte, Cristina, Laura y Lidia. Por enseñarme otra forma de vivir. Por conocernos desde hace poco y haber confiado siempre en mi.

A mis compañeros de trabajo, a Mayra, por enseñarme que la risa es la mejor medicina en momentos de mucho estrés.

A mi tutor Iván y a Víctor, por las horas dedicadas y por todos los consejos y la ayuda.

Y por último y más importante, a mis abuelos, porque enseñarme el valor del esfuerzo del trabajo y la perseverancia desde el más profundo cariño.

RESUMEN

Una de las tecnologías más desarrolladas en los últimos años es la relacionada con los UAVs (*Unmanned Aerial Vehicles*), comúnmente conocidos como drones. Estos dispositivos caracterizados principalmente por volar gracias a un control remoto tienen numerosas utilidades en campos tanto militares como civiles. Las ventajas que suponen los drones son cuantiosas. Reducen tiempo en la realización de diferentes operaciones y el riego humano al tratarse dispositivos no tripulados. Pueden acceder a la mayoría de espacios y ser manejados con facilidad.

Al mismo tiempo, SDN (*Software Defined Network*) es otra tecnología que al igual que los UAVs ha experimentado un gran desarrollo recientemente. Las SDN están caracterizadas principalmente porque el control de la red recae en una nueva figura no existente en las redes convencionales: el controlador. Esto quiere decir que los administradores de la red pueden configurarla y manejarla como deseen. La separación de separación de este plano de control del plano de datos produce ventajas y supone un aumento en la velocidad de innovación en las redes.

Este tipo de redes poseen varias características que las diferencian del resto de redes móviles y dichas características deben tenerse en cuenta a la hora realizar una correcta configuración. El uso de la tecnología SDN se presenta como un posible recurso para establecer una comunicación eficiente en estas redes, pero supone un reto de gran complejidad.

En este trabajo se realiza, principalmente, una exploración de esta tecnología. Esta exploración puede ser dividida en diferentes fases. Primeramente, se realiza un estudio de la tecnología SDN y del protocolo OpenFlow. Se procede también a la elección de un controlador de licencia de código abierto. Es importante recordar que el controlador es la figura principal de las redes SDN. Con la decisión ya tomada, se realiza una exploración del mismo gracias a la herramienta de simulación Mininet. Gracias a esta herramienta se pueden realizar pruebas sobre diferentes topologías y con distintas configuraciones del controlador para identificar problemas y trabajar sobre soluciones para dichos problemas.

Una vez se tiene una configuración del controlador que se ajusta a las necesidades de este trabajo se realiza una exploración de plataformas SBC (*Single-board computer*) en el laboratorio, concretamente las plataformas son Raspberry Pi. Estos dispositivos son unos ordenadores de muy bajo coste que consisten tan sólo en una placa con un tamaño aproximado del palmo de una mano. Se realiza entonces una configuración de los equipos usando el software OVS (*Open Virtual Switch*). Se trata también de un software con licencia de código abierto. Se procede a ejecutar diferentes pruebas sobre estos dispositivos. Las primeras se realizan sobre un escenario cableado y finalmente se lleva a cabo una exploración de la interfaz inalámbrica de los dispositivos.

ABSTRACT

One of the most developed technologies in recent years is related to UAVs (Unmanned Aerial Vehicles), commonly known as drones. These devices, characterized mainly by flying thanks to a remote control, have numerous utilities in both military and civil fields. The advantages of drones are numerous. They reduce time in carrying out different operations and human irrigation as they are unmanned devices. They can access most spaces and be handled easily.

At the same time, SDN (Software Defined Networking) is another technology that, like UAVs, has recently undergone a great development. The SDN technology is characterized mainly because the control of the network falls on a new figure not exiting in conventional networks: the controller. This means that network administrators can configure and manage the network as they wish. The separation of this control layer from the data layer produces advantages and increases the speed of innovation in networks.

This type of networks have several characteristics that differentiate them from other mobile networks and these characteristics must be taken into consideration when making a correct configuration. The use of SDN technology is proposed as a possible resource to establish an efficient communication in these networks, but it is a very complex challenge.

In this End-of-Grade Work, an exploration of this technology is mainly carried out. This exploration can be divided into different phases. First, a study of the SDN technology and the OpenFlow protocol is made. It also proceeds to the choice of an open source license controller. It is important to remember that the controller is the main figure of the SDN networks. With the decision already taken of which controller to choose, an exploration of it is carried out thanks to the simulation tool Mininet. Thanks to this tool, test can be done on different topologies and with diverse configurations of the controller to identify problems and work on solutions for these problems.

Once it is achieved a configuration of the controller that adjust to the needs of this End-of-Grade Work a scan of SBC (Single-board computer) platforms in the laboratory is done, specifically the platforms are Raspberry Pi. These devices are very low cost computers that have the approximate size of the palm of a hand. The equipment is the configured using the OVS (Open Virtual Switch) software. Different test are made on these devices. The first are carried out on a wired scenario and finally a exploration of the wireless interface of the devices is taken.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1

INTRODUCCIÓN

1. INTRODUCCIÓN

En este primer capítulo se va a explicar la motivación y los objetivos de este trabajo de fin de grado. También se va a exponer la estructura que va a seguir la memoria y una breve introducción del trabajo desarrollado en cada capítulo.

1.1 MOTIVACIÓN

Los UAVs (Unmanned Aerial Vehicles) o vehículos aéreos no tripulados comenzaron su historia durante el siglo XIX con fines militares. Los primeros que fueron construidos se trataban de aeronaves muy costosas y de gran tamaño. El continuo desarrollo de la electrónica y la miniaturización de los componentes ha permitido la evolución de estos dispositivos hacia lo que comúnmente se conoce como drones. Los SUAVs (Small Unmanned Aerial Vehicles) o MAVs (Micro Air Vehicles) llevan varios años formando parte de nuestro día a día y parece que están aquí para quedarse durante mucho tiempo.

Los MAVs poseen un tamaño muy inferior que otros vehículos aéreos u otros UAVs como pueden ser Predator, Siva o Milano. Gracias a esta característica los MAVs proporcionan accesibilidad en lugares hostiles, tienen un precio menor, reducen el riesgo humano y pueden ser controlados de una manera muy precisa. También reducen el tiempo necesario para llevar a cabo diversas operaciones por no necesitar tareas previas y poder despegar o aterrizar en prácticamente cualquier lugar.

Como se puede observar, las ventajas que nos ofrecen los vehículos aéreos no tripulados son cuantiosas y por ello actualmente se realizan diversas misiones tanto en el ámbito militar como en el civil. Realizan operaciones de rescate y de control de riesgo, transmiten tráfico multimedia y pueden llevar a cabo el transporte de pequeñas mercancías.

Otra de las ventajas principales que aportan estos pequeños dispositivos es la posibilidad de completar operaciones usando varios UAVs al mismo tiempo. Las primeras misiones completadas con estos dispositivos usan sólo uno de ellos, pero cada vez es más amplia la investigación desarrollada en las redes de drones. El uso de un mayor número de dispositivos supone una menor probabilidad de fracaso de una misión y un ahorro de tiempo en la misma. A su vez son menos costosas, pues se pueden emplear varios dispositivos más económicos en vez de uno más caro.

Las FANETs (Flying Ad Hoc Networks) quedan agrupadas dentro de las VANETs (Vehicular Ad Hoc Network) puesto que los drones son considerados vehículos. Las VANETS a su vez pertenecían al gran grupo de las MANETs (Mobile

Ad Hoc Networks). Tanto MANETs como VANETs han sido altamente desarrolladas en los últimos tiempos y por sus similaridades se intentan usar muchos de los protocolos creados para ellas en las FANETs. Aún así son bastantes las diferencias que existen y por lo tanto surge la necesidad de buscar alternativas para el control de estas redes. Por ello, se ha pensado que una de las opciones que pueden emplearse para realizar una dirección efectiva es el uso de SDN.

La tecnología SDN (*Software Defined Networking*), que será explicada detenidamente en capítulos posteriores de este trabajo, posee una característica principal. Dicha característica es que se produce una separación entre el plano de datos y el plano de control. Esta separación conlleva la aparición de una nueva figura en la red, el controlador SDN, que como su propio nombre indica se encargará del plano de control. Desde este dispositivo se podrá dirigir la configuración del resto de dispositivos de la red y gestionarla de una manera centralizada.

En la unión de estos dos conceptos es donde reside principalmente la motivación de este trabajo de fin de grado. La tecnología SDN puede ser una buena solución para manejar redes de datos entre drones con una mayor eficiencia, pues el controlador puede tener una imagen en tiempo real de la red y realizar una rápida reconfiguración de la misma. El uso de esta tecnología en estas redes supone un reto de gran complejidad y por lo tanto resulta necesario realizar una primera exploración.

1.2 OBJETIVOS

El objetivo principal de este trabajo de fin de grado es realizar una primera exploración de la tecnología SDN para configurar el reenvío de información. Esta exploración se llevará a cabo primero en un entorno de simulación y más tarde en el laboratorio con dispositivos físicos.

Este objetivo global puede dividir en los siguientes puntos:

- Estudio de las redes de MAVs. Comprender sus principales características y necesidades.
- Estudio de la tecnología SDN. Comprender sus principales características y ventajas que puede aportar. Entender la arquitectura de las redes SDN y los principales componentes de estas.
- Estudio de diferentes controladores SDN teniendo en cuenta sus principales características y elección de un controlador SDN para la red.
- Estudio del funcionamiento del protocolo OpenFlow, fundamental en las redes SDN.
- Estudio del funcionamiento y características de los switches OpenFlow. Realizar un análisis de los mensajes más importantes intercambiados

entre lo switches y el controlador que serán fundamentales en el desarrollo de este trabajo.
- Realización de pruebas sobre el controlador elegido simulando diferentes topologías de red con la herramienta Mininet.
- Identificación de diferentes problemas en la configuración del controlador y creación de soluciones para dichos problemas.
- Estudio de las características de los dispositivos Raspberry Pi y configuración de los mismos para que funcionen como switches OpenFlow.
- Realización de pruebas en un escenario fijo y cableado en el laboratorio con la configuración previa del controlador y los dispositivos Raspberry Pi.
- Exploración de la interfaz inalámbrica de los dispositivos.

1.3 ESTRUCTURA

Este trabajo se encuentra dividido en diferentes capítulos. A continuación se explica brevemente el contenido de cada uno de ellos:

CAPÍTULO 1: INTRODUCCIÓN

En este capítulo se realiza una breve introducción a la idea fundamental del trabajo y se enmarcan los objetivos que se quieren cumplir durante el desarrollo del mismo. Finalmente también se indica la estructura que va a seguir la memoria.

CAPÍTULO 2: ESTADO DEL ARTE

En este capítulo se realiza un estudio de diferentes conceptos teóricos necesarios para el desarrollo del proyecto. Se cumplen los objetivos de estudio de las redes de MAVs, la tecnología SDN y el protocolo OpenFlow. Pero también se hablan de otras herramientas necesarias como el protocolo LLDP o los dispositivos Raspberry Pi.

CAPÍTULO 3: USO DE LA TECNOLOGÍA SDN EN REDES MULTISALTO

Este es el capítulo más extenso del trabajo. En él se comienza realizando una primera exploración del controlador elegido que nos sirve para identificar los problemas. Una vez se han identificado los problemas para nuestro tipo de redes SDN multisalto se procede a detallar las soluciones que se van proponiendo. A su vez se explica cuales son las características de dichas soluciones y si resultan adecuadas para los problemas presentes. Todas las pruebas realizadas durante este apartado se ejecutan en un escenario fijo simulado.

CAPÍTULO 4: ESCENARIO REAL

En este capítulo, una vez se tiene configurado un controlador SDN, se continúa configurando un escenario real con los equipos embarcados en el laboratorio. Se especifica la distribución de los componentes y cómo se realiza la configuración de los mismos. Las pruebas realizadas en este apartado consisten en dos partes. Unas primeras pruebas sobre el escenario cableado con el controlador configurado y una primera exploración de la interfaz inalámbricas de los dispositivos SBC.

CAPÍTULO 5: CONCLUSIONES Y TRABAJOS FUTUROS.

En este capítulo se explican las conclusiones de las implementaciones y pruebas realizadas y los trabajos necesarios para poder continuar la investigación de la idea fundamental de este proyecto.

ANEXOS

En los Anexos se encuentra información adicional acerca del trabajo desarrollado. Consta de los siguientes apartados:

- **ANEXO I**: Planificación
- **ANEXO II**: Entorno socio económico
- **ANEXO III**: Marco regulador
- **ANEXO IV**: Instalación y ejecución de Mininet
- **ANEXO V**: Instalación de OVS en Rpi
- **ANEXO VI**: Fichero de configuración de red de RPi
- **ANEXO VII**: Comandos OVS
- **ANEXO VIII**: *Extended abstract*
- **ANEXO IX:** Bibliografía

CAPÍTULO 2

ESTADO DEL ARTE

2. ESTADO DEL ARTE

Teniendo en cuenta los objetivos descritos en el apartado anterior en este capítulo se explican los fundamentos teóricos en los que se basan y los protocolos y herramientas que son necesarios entender y utilizar para poder cumplir con dichos objetivos.

2.1 UNMANNED AIR VEHICLES

Como se ha comentado en el capítulo anterior, los SUAVs o MAVs han llegado a nuestras vidas para quedarse. Aunque la idea de los vehículos no tripulados es anterior a este siglo y tenían un uso exclusivamente militar, el desarrollo que han tenido en los últimos años ha sido muy elevado.

Utilizados en muchas misiones tanto civiles como militares, poseen una serie de virtudes que los diferencian del resto de vehículos aéreos y que hacen de ellos una opción ideal para el desarrollo de numerosas operaciones. Los drones pueden llevar a cabo misiones desde la transmisión de contenido multimedia o la asistencia durante emergencias y catástrofes hasta el reparto de mercancías. Estos vehículos aéreos no son comparables al resto de dispositivos que ocupan nuestro cielo. El hecho de que se trate de dispositivos mucho más baratos y pequeños que el resto de vehículos aéreos conlleva la realización de misiones con un menor coste y unas mejores prestaciones.

Gracias a su tamaño pueden acceder a lugares inhóspitos que resultan inaccesibles de otra forma. Reducen de forma considerable tres importantes factores el cualquier misión. Estos factores son el humano, el tiempo y los costes. El factor humano, es decir, el riesgo laboral que supone para una persona, disminuye gracias al hecho de tratarse de vehículos no tripulados. El coste también se ve reducido y no sólo porque el precio del dispositivo sea menor, sino porque también se ahorra en otros aspectos relacionados con los vuelos como el combustible o tasas aeroportuarias. El tiempo de las operaciones se ve disminuido también debido a la posibilidad de realizar despegues y aterrizajes desde casi cualquier lugar y a que poseen una mayor agilidad.

Estas son algunas de las principales ventajas que este tipo de vehículos pueden aportar. Ahora que se han visto sus virtudes se va a estudiar cuáles son la partes fundamentales para que estos dispositivos puedan volar. Para que un UAV tenga un correcto funcionamiento y pueda desarrollar una operación satisfactoriamente son necesarios cuatro subsistemas [2].

En primer lugar se necesita una plataforma de vuelo, es decir, un vehículo. En segundo lugar se necesita un FCS (Flight Control System) o sistema de control de

vuelo. Consiste en el conjunto de sensores y sistemas integrados en el vehículo encargados de controlar la localización de éste y su estabilidad. En tercer lugar debe existir una GCS (Ground Control Station) o estación de control terrestre compuesta por todos los dispositivos necesarios para monitorizar el vehículo y su FCS. En cuarto y último lugar se necesita un sistema de comunicaciones para el tráfico de datos entre el vehículo y la GCS. En la figura 1 se pueden observar un esquema de estos elementos.

Figura 1 - *Arquitectura UAV*

2.1.1 MULTI - UAV NETWORK

Las primeras operaciones que se llevaron a cabo con UAVs estaban caracterizadas por el uso de un único dispositivo. Actualmente se puede hablar de varias ventajas que tiene el uso de un grupo de pequeños UAVs frente al uno sólo. Entre estas virtudes se puede destacar:

- Un menor coste, pues se pueden emplear varios dispositivos más baratos y sencillo de mantener en vez de uno mucho más caro.
- Un aumento de probabilidad de éxito de la misión dada la existencia enlaces redundantes y la posibilidad de la reconfiguración del sistema.
- Un aumento de la velocidad en la que las operaciones son llevadas a cabo y que por consiguiente produce una disminución del tiempo de operación.
- Una mayor escalabilidad de las misiones pues se produce un aumento en el aŕea cubierta por los dispositivos.

Por su naturaleza, las FANETs (Flying Ad Hoc Networks) [3] se han comparado con las VANETs (Vehicular Ad Hoc Networks) y MANETs (Mobile Ad Hoc Network). Cierto es que son muchas las similitudes que pueden encontrarse entre ellas puesto que

las FANETs quedan agrupadas dentro de las VANETs al tratarse los drones como vehículos y a su vez dentro de las MANETs ya que se trata de redes móviles.

A pesar de sus similitudes las redes de drones poseen varias diferencias respecto de las redes móviles ya conocidas. Algunas de estas diferencias son:

- La movilidad de los nodos es mayor que en las MANETs y las VANETs. Los UAVs tienen una velocidad muy variable, entre los 30 y los 460 km/h. Por lo tanto, el movimiento de estos dispositivos resulta más impredecible que en el resto de dispositivos móviles.
- Los UAVs presentan un modelo de movilidad no predeterminado. Incluso aunque existan planes de vuelo prefijados pueden cambiar durante el transcurso de la misión lo que puede afectar a la comunicación entre los nodos.
- La densidad de nodos disminuye respecto de las topologías de MANETs y VANETs, es decir, en un mismo espacio hay un menor número de dispositivos. En las redes de UAVs los dispositivos pueden estar a una distancia de hasta kilómetros entre ellos.
- En las FANETs la topología de la red puede variar más rápidamente que en otras redes móviles. Los nodos pueden desactivarse o sufrir problemas por diferentes motivos como la finalización de la batería. Esto produce la caída de todos los enlaces que mantenía activos dicho nodo y la necesidad de reconfigurar la red.
- En diferencia con otras redes vehiculares las FANETs tienen limitación de batería, de peso y de capacidad de computación y procesamiento. Gracias a la miniaturización de los componentes en los últimos años este problema es poco a poco menor, pero aún así se debe tener en cuenta.

Estas son algunas de las características de las redes formadas por vehículos aéreos no tripulados y es importante tenerlas en cuenta para el diseño de éstas. Existen muchos protocolos de comunicación implementados para redes móviles que se han intentado aplicar para las redes de drones. En este trabajo el método usado para la comunicación entre los dispositivos será SDN. Qué es y por qué se ha pensado en utilizarlo se explica en puntos posteriores.

2.2 SOFTWARE DEFINED NETWORK

Las redes definidas por software o SDN [4] por sus siglas en inglés (Software Defined Networks) surgen hace pocos años de la necesidad de gestionar una red de forma centralizada. La principal característica de estas redes es su programabilidad, es

decir, la capacidad de controlar distintos comportamientos de la red de una forma dinámica gracias a unas interfaces activas.

Las redes antiguas resultan poco flexibles. En ellas en plano de datos y el plano de control se encuentran unidos dentro de un mismo dispositivo. El plano de datos es el conjunto de recursos en los dispositivos de red responsables de reenviar el tráfico. En cambio, el plano de control es el conjunto de recursos encargados de la gestión del funcionamiento de los dispositivos de red del plano de datos. Es decir, en las redes antiguas un mismo dispositivo se encuentran los recursos para realizar en reenvío del tráfico y el software necesario para controlar cómo se produce dicho reenvío. En las SDN el plano de control se separa del plano de datos y es en una nueva figura, el controlador, sobre la que se va a delegar la toma de decisiones y se va a privar de inteligencia a los demás elementos.

Algunas de las ventajas de implementar una SDN son [5]:

- El uso de diferentes dispositivos de red realizando un control centralizado sobre todos ellos. En una misma red se pueden emplear tanto dispositivos dispositivos virtualizados como físicos que a su vez pueden ser producidos por distintos fabricantes.
- Velocidad de innovación en las redes al no es necesario esperar por actualizaciones del fabricante o configurar los dispositivos individualmente.
- Pueden modificarse redes en tiempo real mientras que en las redes antiguas para realizar cambios se necesitaban días o semanas.
- Facilidad de diseño de distintas topologías de red, los administradores pueden configurar una red de una manera sencilla. Esto aporta agilidad a la hora de crearlas, modificarlas y afrontar sus cambios.
- Capacidad de administración del ancho de banda según las necesidades del usuario o de la red.
- El uso de entornos de programación comunes e intuitivos proporciona a diferentes usuarios, como operadores o empresas, la oportunidad de desarrollar e innovar la red.

En una SDN, de forma básica, se encuentran tres capas completamente diferenciadas: la capa de aplicación, la capa de control y la capa de infraestructura. El controlador se encuentra situado en la capa de control y se comunica hacia arriba con la capa de aplicación y hacia abajo con la capa de infraestructura [Figura 2]

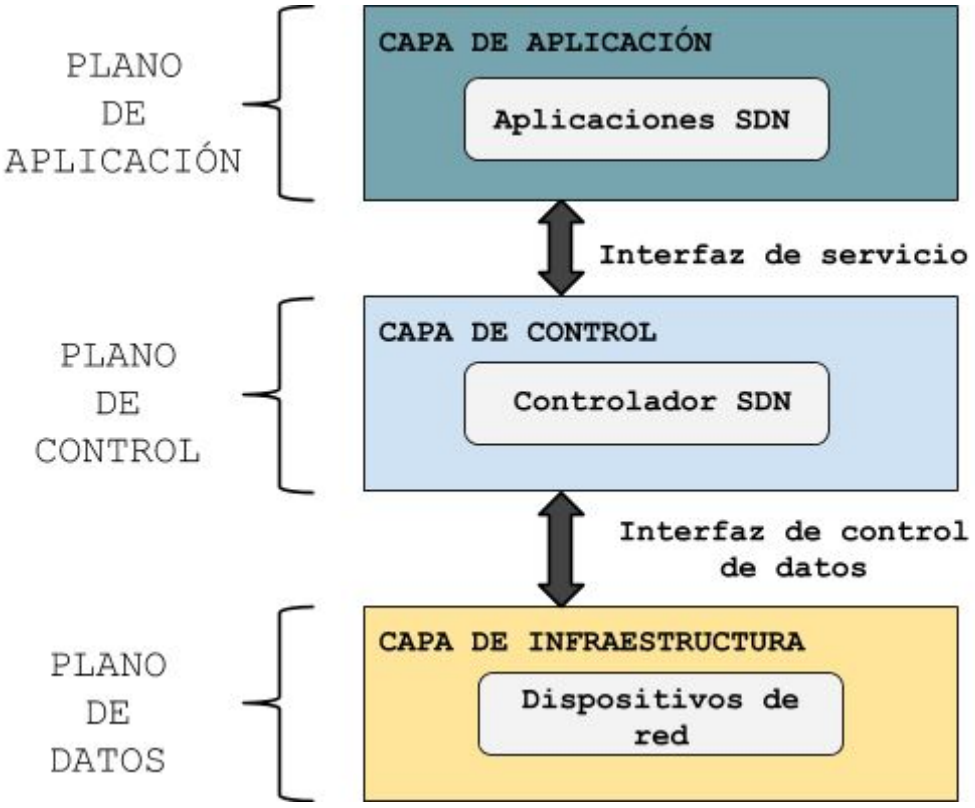


Figura 2 - *Arquitectura SDN*

En la capa de aplicación residen las aplicaciones encargadas de comunicar a la capa de control qué necesidades tiene la red y cuáles son las formas en las que se debe proceder para obtener los comportamientos deseados en ella. Esta información se comunica a la capa de control mediante la interfaz de de servicio.

En la capa de control se encuentra la figura principal de las SDN, el controlador. Su misión más importante es traducir las órdenes y especificaciones impuestas por la capa de aplicación a la capa de infraestructura. Estas órdenes se envían por el canal de comunicación entre el plano de control y el plano de datos, la interfaz de control de datos. Hay varios estándares que se pueden utilizar en este canal de comunicación, como OpenFlow, protocolo del que se hablará en apartados sucesivos.

En la capa de aplicación se encuentran los dispositivos de red. Estos dispositivos, que son configurados según las instrucciones que han recibido por el controlador, se encargan del reenvío de datos de la red. Este reenvío se realiza en base a unas rutas o flujos que son impuestos por el controlador.

2.2.1 CONTROLADORES SDN

Como se podido observar, el controlador SDN, es la figura principal de este tipo de redes. Pudiéndose considerar el corazón de la red, la elección de un controlador que mejor se ajuste a las necesidades del proyecto resulta fundamental. A la hora de escoger un controlador estos son algunos de los puntos que son considerados más relevantes y que se deben tener en cuenta [6].

Una de las características más importantes a tener en cuenta a la hora de escoger un controlador es la versión del protocolo OpenFlow que puede soportar. Este protocolo, que ya ha sido mencionado previamente y que se explicará con detalle más adelante, es el encargado de la comunicación entre el controlador y los dispositivos de la red. Existen más protocolos que pueden ser utilizados en la interfaz de control de datos pero OpenFlow es el más extendido. Por este motivo es muy importante la elección de la versión de este protocolo y la comprensión del funcionamiento del mismo.

Otra característica que es relevante analizar es la funcionalidad de red que nos puede ofrecer el controlador. Un controlador debe poder tomar decisiones de enrutamiento basándose en diferentes campos de la cabecera de un mensaje [Figura 3] , como por ejemplo el puerto de entrada, la dirección de destino o la VLAN a la que pertenece entre otros muchos.

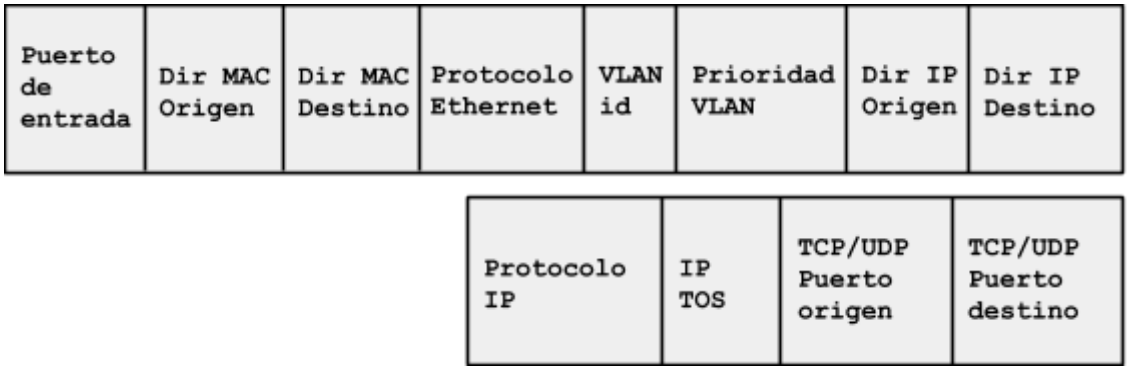


Figura 3 - *Cabecera trama OpenFlow*

Dado este hecho un controlador puede ser capaz de crear diferentes rutas entre un origen y un destino basándose en diferentes parámetros y dividir el tráfico entre los dos nodos por diferentes enlaces. La división del tráfico puede traducirse como un gran aumento del rendimiento de la red puesto que puede evitarse la saturación de enlaces y la congestión de éstos.

Un factor que no se debe olvidar a la hora de escoger un controlador es su escalabilidad. En redes de crecimiento continuo y topología cambiante es necesario un

controlador que sea capaz de reaccionar ante los diferentes sucesos que vayan ocurriendo y que lo haga que una manera eficiente. Los controladores tienen un número máximo de dispositivos de red que pueden controlar al mismo tiempo y a su vez estos dispositivos cuentan con un número limitado de entradas existentes en su tabla de flujo. Por este motivo el controlador es responsable de realizar una correcta administración de los recursos de la red, gestionando los nodos y sus tablas de flujo de una manera eficiente.

Por último, pero no menos importante, se encuentra el análisis del rendimiento que puede ofrecernos el controlador. Como se ha mencionado, una de las funciones de los controlador es la gestión de los flujos que poseen los dispositivos de red. Pero también es necesario tener en cuenta los tiempos empleados en dicha gestión y la forma en la que se realiza. La asignación de los flujos se puede realizar de forma reactiva o proactiva. Si se realiza de manera reactiva quiere decir que el controlador establece los flujos una vez ha comenzado el tráfico en la red, es decir, cuando un dispositivo no tiene una entrada en su tabla de flujos que coincida con el paquete recibido lo enviará al controlador y quedará a la espera de órdenes. Si la red se configura de manera proactiva los flujos se establecen previamente a que comience el tráfico de datos en la red. Si se escoge un controlador que sea capaz de obtener la topología de la red desde que ésta se establece, se podrá configurar de manera reactiva o proactiva según se prefiera.

Existen más características de los controladores que se pueden tener en cuenta para su elección pero en estos cuatro puntos queda resumido los que se consideran más relevantes para realizar una decisión correcta.

En [7] puede observarse una comparativa de los principales controladores SDN más utilizados actualmente. Estos controladores son: Beacon, Floodlight, NOX, POX, Trema, Ryu y Open Day Light. Se comparan las características como la versión del protocolo OpenFlow que soportan, el lenguaje en el que han sido desarrollados y el soporte de plataformas entre otras. Estos puntos también es necesario tenerlos en cuenta a la hora de elegir un controlador.

2.2.2 RYU

Como ha sido mencionado ya en numerosas ocasiones, el controlador SDN es el cerebro de este tipo de redes, son los encargados de comunicar a los diferentes dispositivos de red las instrucciones de funcionamiento creadas por usuarios finales o desarrolladores. Para este trabajo el controlador escogido es Ryu y en este punto se explicarán sus características.

Ryu fue diseñado por el NTT para crear redes SDN más ágiles haciendo más sencilla la manera de soportar y manejar el tráfico cursado por la red. Ryu proporciona

diferentes componentes de software completamente definidos en una API [8]. Estos componentes pueden ser modificados por los desarrolladores según las necesidades de la red que se desea programar.

Algunas de las características de este controlador son:

- Es un controlador de código abierto bajo la licencia Apache 2.0.
- Es compatible con varios protocolos de administración de dispositivos de red como son NetConf, OF-Config y OpenFlow en todas sus versiones (1.0, 1.2, 1.3, 1.4, 1.5 y extensiones Nicira).
- Soporta virtualización con Mininet y OVS (Open Virtual Switch).
- Sus componentes han sido programados en Python.
- Posee una interfaz gráfica web y una REST API
- Soporte sobre Linux.

Gracias al hecho de poseer distintos componentes programados en un lenguaje de alto nivel, se puede decir que Ryu es un controlador altamente flexible que facilita la labor de desarrollo de nuevas aplicaciones de administración, control y gestión de las redes.

2.3 OPENFLOW

OpenFlow es un protocolo nacido del proyecto "OpenFlow: Enabling Innovation in Campus Networks" en la universidad de Stanford y actualmente se encuentra en desarrollo de estándares bajo la ONF (Open Networking Foundation) [9]. Es el primer protocolo diseñado y estandarizado para las SDN.

Como se ha comentado previamente, se trata de un protocolo necesario para la comunicación entre el controlador y los dispositivos de la red. Gracias a este protocolo podemos programar los flujos que queremos que aprendan nuestros switches openflow.

2.3.1 SWITCH OPENFLOW

Un switch, en su concepto tradicional, es un dispositivo de red que sirve de conexión entre otros dispositivos, separando la red en diferentes dominios de colisión. El funcionamiento de un switch es sencillo, se encarga del reenvío de paquetes por sus diferentes puertos. Dispone de una tabla de flujo en la que en para cada entrada en la tabla se relacionan diferentes características de los paquetes y se establece un puerto de salida. Los switches OpenFlow, como su propio nombre indica, son diseñados para

funcionar en las SDN con controladores OpenFlow y se comunicarán mediante este protocolo.

El concepto de switch OpenFlow implica el conjunto de tres conceptos [Figura 4]. Una tabla de flujos dónde se especifica qué acciones tomar con los paquetes entrantes. De manera obvia se necesita un controlador que gestiona las entradas de la tabla. Por último se necesita un canal seguro de comunicación para conectar el switch OpenFlow y el controlador y que puedan intercambiar diferentes paquetes.

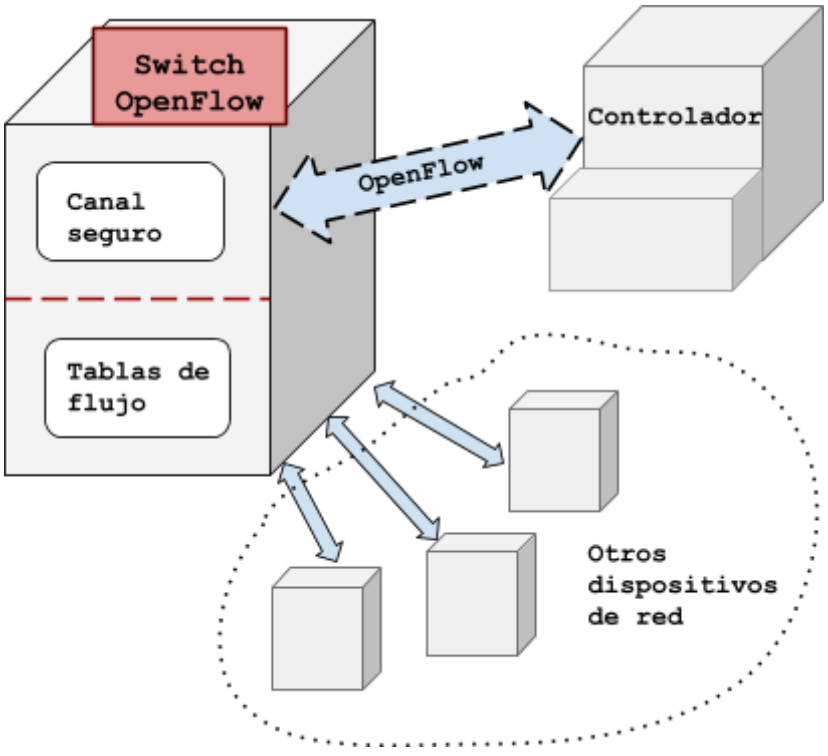


Figura 4 - *Entorno switch OpenFlow*

Estos dispositivos poseen unas funcionalidades básicas como:

- Ser capaces de establecer flujos para diferentes campos de los paquetes.
- Soportar funciones de OpenFlow como la reescritura de cabeceras.
- Usar el protocolo LLDP (Link Layer Discovery Protocol). Este protocolo, que será explicado con detalle más adelante, permite el descubrimiento de otros dispositivos de la red para que el controlador pueda tener una imagen de la topología de ésta antes del comienzo de la transmisión de datos.

El funcionamiento básico de los switches OpenFlow es sencillo y queda descrito en el siguiente diagrama. [Figura 5]

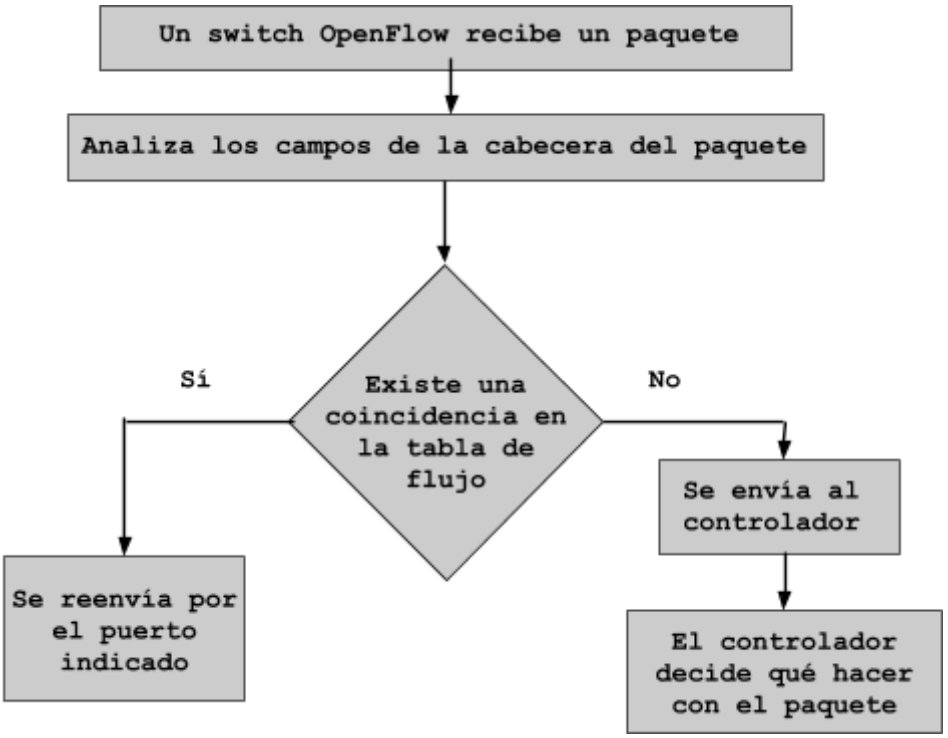


Figura 5 - *Funcionamiento switch OpenFlow*

Como ya ha sido mencionado, son los controladores los que establecen las entradas de flujo en las tablas de los switches. Pero entre el controlador y los switches existen más tipos de paquetes que aquellos destinado a la gestión de los flujos. A continuación se procede a explicar algunos mensajes OpenFlow y flujos de éstos existentes entre el controlador y los dispositivos.

2.3.2 MENSAJES OPENFLOW

En este apartado se explicarán los diferentes tipos de mensajes que el protocolo OpenFlow tiene estandarizados en la versión 1.3 [10] para la comunicación entre el controlador y los dispositivos de red.

Se pueden diferenciar tres claros grupos de mensajes: mensajes del controlador al switch, mensajes asíncronos y mensajes simétricos.

Los mensajes del **controlador al switch**, como el propio nombre indica, son iniciados por el controlador y pueden necesitar o no una respuesta del switch. Estos paquetes tienen los objetivos de obtener información sobre las capacidades y

características de los switches y realizar un control y diversas operaciones sobre los flujos de los switches. Los mensajes de este grupo son los siguientes:

- ***Features*:** El controlador envía al switch un *features request* dónde le pregunta al switch por sus capacidades y el switch debe contestarle con un mensaje *features reply* dónde deja indicada toda la información requerida. Este mensaje típicamente tiene lugar durante el inicio de la conexión.
- ***Configuration:*** El controlador puede configurar o preguntar al switch por sus parámetros de configuración. El switch sólo debe responder cuando se le pregunta.
- ***Modify-State:*** El controlador envía estos mensajes para realizar la gestión de las tablas de flujo de los switches. Se pueden añadir, modificar o borrar entradas de la tabla de flujo y configurar las propiedades de los puertos de los switches.
- ***Read-State:*** El controlador envía estos mensajes para conseguir información de los switches como su actual configuración, estadísticas o capacidades.
- ***Packet-out*:** Este tipo de mensajes son enviados por el controlador para redireccionar un mensaje *packet in* previamente recibido o enviar paquetes por cierto puerto del switch. El mensaje contiene el paquete que se debe redireccionar y las acciones para ello. Si el campo de acciones se encuentra vacío quiere decir que el paquete debe ser descartado.
- ***Barriel:*** Los mensajes *barriel request* y *barriel reply* son enviados por el controlador para controlar que las dependencias de un mensaje han sido cumplidas o para recibir notificaciones sobre operaciones ya completadas.
- ***Role-Request:*** Este mensaje es útil cuando un dispositivo se encuentra conectado a varios controladores y es utilizado para establecer o consultar el rol de un canal.

Los mensajes **asíncronos** son enviado por el switch al controlador sin ningún requerimiento por parte de éste último. Los switches envían estos mensajes para notificar la llegada de un paquete, un cambio en el estado o un error. Los principales mensajes de este grupo son los siguientes:

- ***Packet-In:*** La finalidad de este mensaje es transferir el control de un paquete al controlador. Cuando un switch no posee una entrada en su tabla de flujo para un paquete lo envía en forma de *packet in* al controlador. Éste lo procesa, realiza las acciones que considere necesarias y responde con un *packet out.*
- ***Flow-Removed:*** El switch envía este mensaje para informar al controlador sobre una entrada en su tabla de flujo ha sido eliminada. Solo en flujos que tuvieran activo el flag *OFPFF_SEND_FLOW_REM* activo. La entrada de flujo puede ser borrada por un mensaje *flow delete* enviado por el controlador o porque el temporizador de la entrada haya expirado.
- ***Port-Status:*** El switch envía este mensaje al controlador cuando el estado de uno de sus puertos cambia.

- ***Error:*** El switch puede notificar al controlador diferentes errores con este mensaje.

Los mensajes **simétricos** pueden ser enviados en ambos sentidos de la comunicación entre el controlador y los switches y no necesitan una solicitud previa. Los mensaje de este grupo son los siguientes:

- ***Hello:*** Estos mensajes son intercambiados entre el switch y el controlador al establecimiento de la conexión.
- ***Echo:*** Durante la conexión se intercambian continuamente mensajes de *echo request* y *echo reply*. La principal finalidad de este mensaje es comprobar que la conexión sigue activa pero también pueden ser usados para medir la latencia o el ancho de banda.
- ***Experimenter:*** Con estos mensajes el controlador puede añadir funcionalidad adicionales a los switches. Planeado para futuras versiones del protocolo.

Una vez han quedado aclarados los diferentes mensajes, en los siguientes puntos se va a detallar el intercambio de mensajes en algunos momentos que se consideran importantes para este trabajo durante una conexión entre switch y controlador.

2.3.2.1 INICIO DE LA CONEXIÓN

El inicio de la conexión, como en todas las conexión en una red es un punto importante. En el inicio de la conexión el controlador recibirá información sobre la configuración y las capacidades de los switches que será necesaria para el correcto control de la red.

Si el switch conoce la dirección IP del controlador iniciará una sesión TLS o TCP con el controlador. Una vez establecida dicha sesión tanto el switch como el controlador enviarán un mensaje *HELLO* para proceder a la configuración de la conexión. En este mensaje deben especificar la máxima versión del protocolo OpenFlow que soportan. Cada uno calculará la versión que debe usarse durante la comunicación teniendo en cuenta la información proporcionada por el contrario. Si la versión negociada es soportada por ambas partes la conexión continúa. En caso contrario se debe enviar un mensaje de *ERROR*.

Si la versión negociada es soportada por ambas partes la conexión continúa. Es ahora cuando el controlador requiere información al switch. Mediante el mensaje de *FEATURES REQUEST* el controlador le pregunta al switch por sus capacidades y éste las incluye en el mensaje de respuesta *FEATURES REPLY*. En la [Figura 6] se puede ver un ejemplo de una trama de respuesta de un switch indicando sus capacidades.

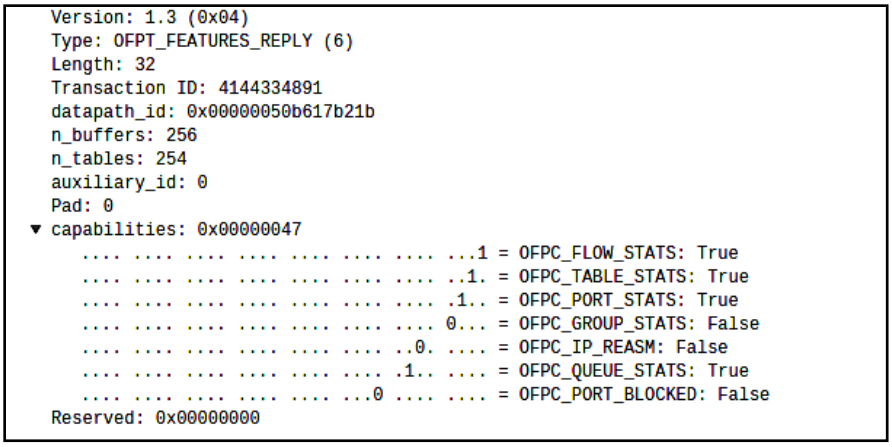


Figura 6 - Mensaje *Features Request*

A continuación, el controlador enviará un mensaje *PORT DESC STATS REQUEST*. Con este mensaje quiere que el switch le envíe una descripción de los puertos que tiene y del estado de configuración de los mismos. También añadirá una entrada en la tabla de flujo mediante el mensaje *ADD FLOW*. Esta entrada es fundamental en la futura comunicación pues es la que indica a un switch por donde debe enviar el tráfico hacia el controlador cuando reciba un paquete y no coincida con ninguna otra entrada en su tabla de flujos. Por último, el switch contesta al requerimiento realizado por el controlador con un mensaje *PORT DESC STATS REPLY*. Un ejemplo de ese dicho mensaje y cómo se encuentra estructurada la información enviada puede verse en la [Figura 7].

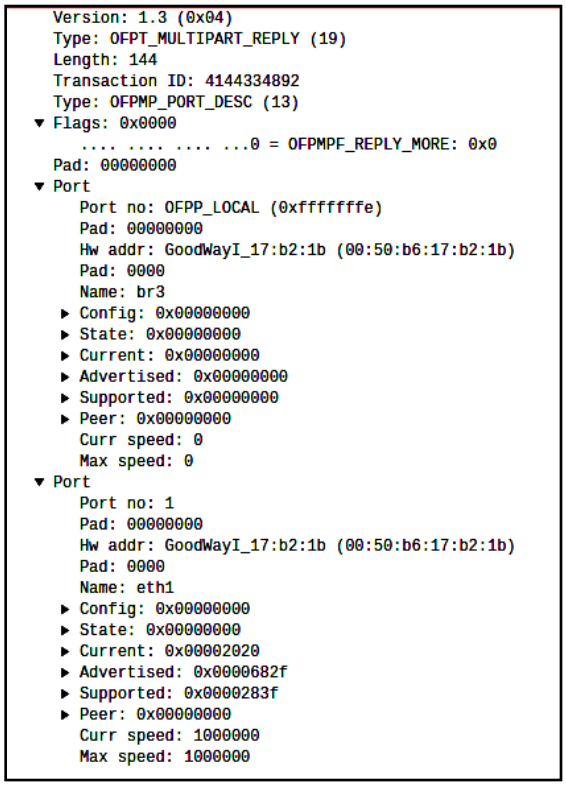


Figura 7 - Mensaje *Port Desc Stats Reply*

La [Figura 8] muestra un diagrama del intercambio de mensajes para el inicio de conexión que ha sido detallado.

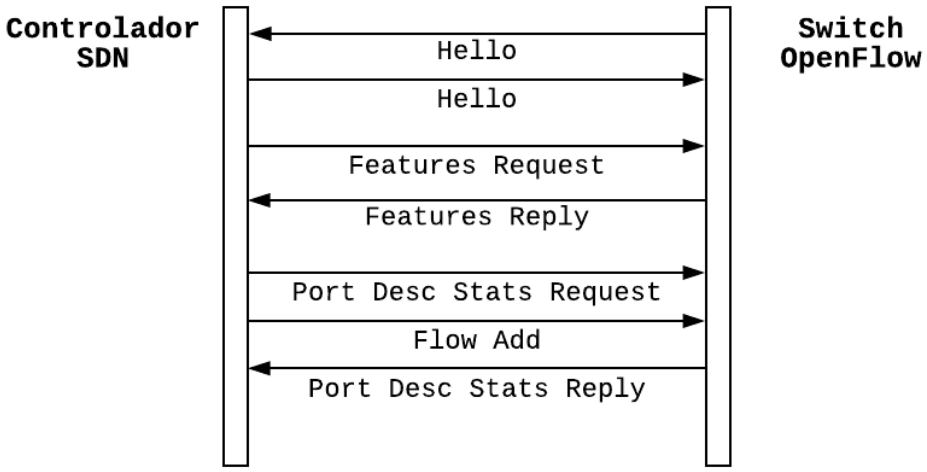


Figura 8 - *Inicio conexión OpenFlow*

2.3.2.2 CAMBIO EN LA RED

El estado de los puertos de los dispositivos de red puede cambiar. Es importante que esto cambios sean notificados al controlador para que este pueda realizar una reconfiguración de la red.

Para este trabajo es interesante estudiar el tráfico de mensajes que tiene lugar cuando un enlace cae o vuelve a estar activo. Las notificaciones de cambios de estado de los puertos se envían al controlador mediante un mensaje *PORT STATUS.*

En la [Figura 9] se puede observar un ejemplo de este tipo de mensaje. Posee numerosos campos en los que el switch puede indicar diferentes estados y características de sus puertos. Un switch para notificar el estado de un enlace y de los puertos debe modificar el valor de los flags *OFPPC_PORT_DOWN* y *OFPPS_LINK_DOWN*. En la figura se puede apreciar que el puerto número uno presenta los dos flags con valor 1, esto indica que el puerto está caído y que ello ha provocado la caída del enlace. Si el primero de los flags no estuviera activo se trataría de un mensaje enviado por un switch que ha detectado la caída del enlace que lleva por ese puerto, pero el puerto sí estaría activo. Al contrario, cuando el enlace o los puertos vuelven a estar activos el mensaje de notificación presentará los flags con valor 0.

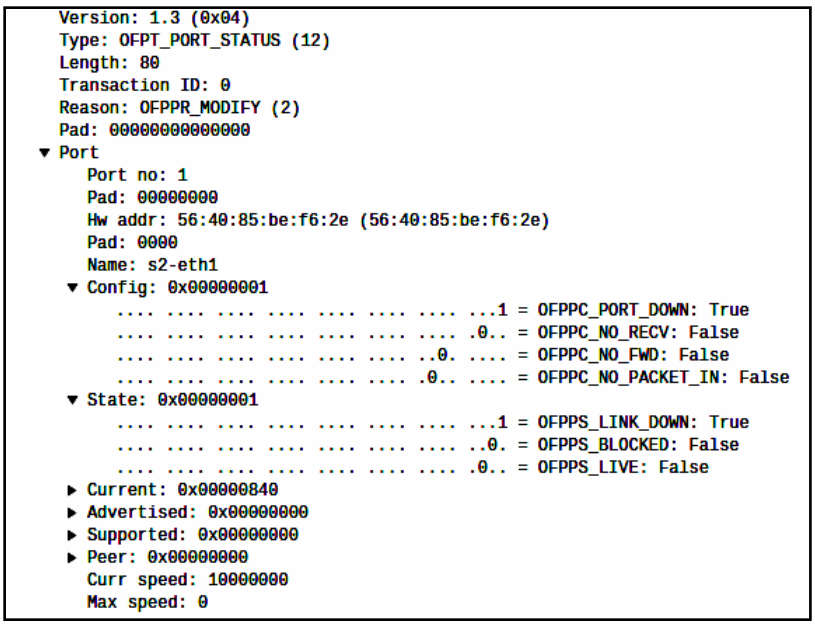


Figura 9 - Mensaje *Port Status*

2.3.2.3 MODIFICAR FLUJOS

La tabla de flujos es fundamental en el funcionamiento de los switches y el controlador es el encargado de gestionarla. Añade, borra y modifica las entradas de la tabla según la configuración de la red.

El protocolo especifica 5 tipos de mensajes de modificación de flujos:

- *Add:* Añade un flujo.
- *Modify:* Se modifican todas las entradas que coincidan con un valor.
- *Modify Strict:* Se modifican una entrada que coincide específicamente con unos valores incluídos en el mensaje.
- *Delete:* Se borrar todas las entradas que coincidan con un valor.
- *Delete Strict:* Se borra una entrada que coincide específicamente con unos valores incluídos en el mensaje.

En la [Figura 10] puede observarse los campos de un mensaje que añade una nueva entrada a la tabla de flujo. En el campo *match* se especifican los valores con los que deben compararse los paquetes entrantes, en este caso se indica un puerto de entrada y una dirección Ethernet de destino. En el campo *actions* se especifican las acciones que debe tomar el switch con los paquetes que coincidan con esa entrada, en este caso se indica un puerto de salida.

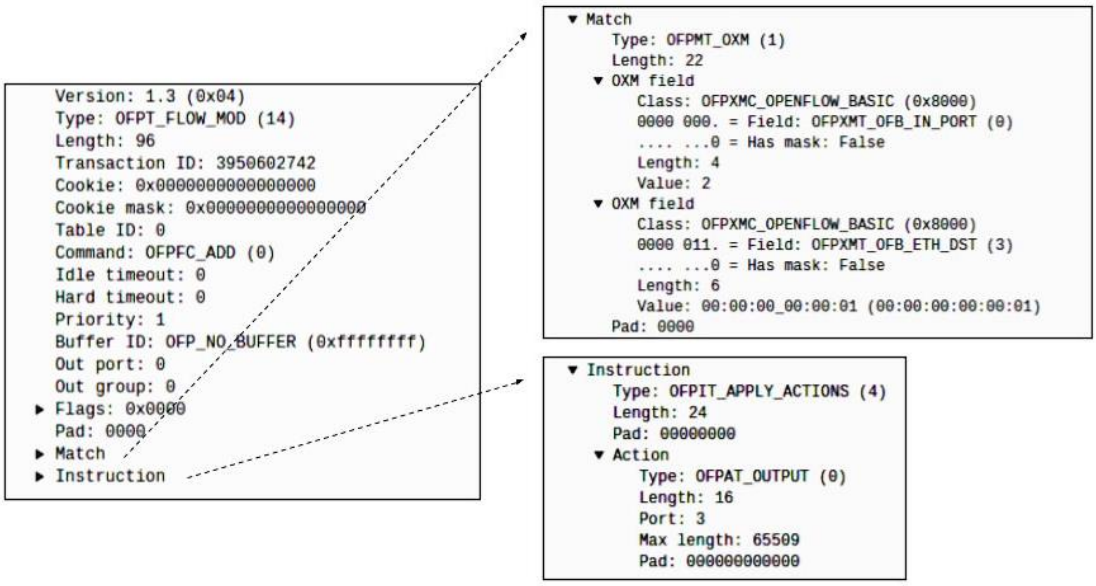


Figura 10 - Mensaje *Flow Mod (add)*

A continuación, en la [Figura 11] se puede observar un mensaje similar. En este caso se trata de un ejemplo de mensaje donde lo que se ordena es borrar un flujo. Se puede ver que en el campo *match* está indicado que deben borrarse los flujos que contengan una determinada dirección Ethernet.

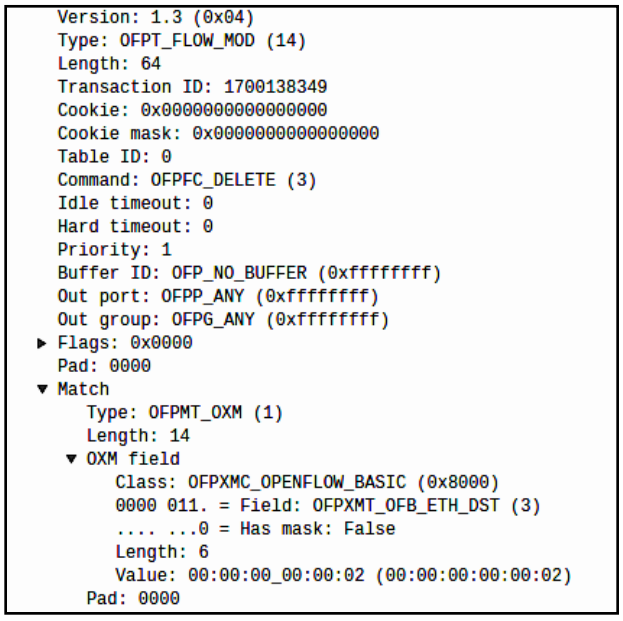


Figura 11 - Mensaje *Flow Mod (Delete)*

2.4 SDN Y UAV NETWORKS

Una vez se han visto las características de las redes SDN y se ha entendido el protocolo OpenFlow se puede hablar de porqué puede resultar útil su uso en las redes de UAVs.

Como se dice en [1] las SDN poseen características que pueden adaptarse a las condiciones y problemas de las las redes de drones que se pretende probar en este trabajo. Las SDN se pueden reconfigurar rápidamente, el controlador posee una imagen real de la red y de la topologías y por tanto puede adaptarse a los continuos cambios que pueden producirse en redes con alta movilidad. Las redes SDN funcionan en diferentes entornos y pueden ayudar a reducir interferencias gracias a que se puede seleccionar el canal en el que emitir. El uso de OpenFlow proporciona la capacidad de usar diferentes dispositivos de una manera efectiva. También puede establecer reglas para el envío y enrutado del tráfico de datos de una manera sencilla.

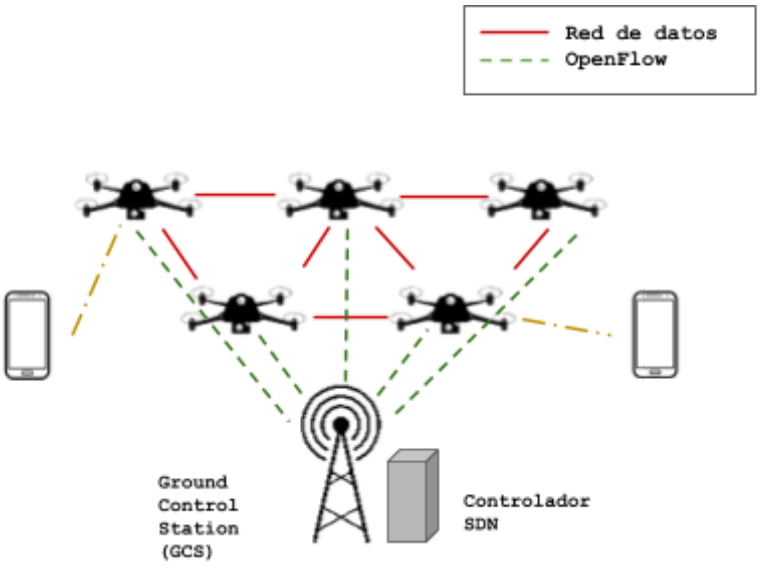


Figura 12 - *Combinación SDN y UAVs*

En la [Figura 12] se puede observar un esquema de cómo sería una red de drones según la motivación de este comlejo proyecto. El controlador de la SDN quedaría ubicado en la GCS. Todos los drones usarían el sistema de comunicación para enviar y recibir el tráfico necesario con el controlador. Los enlaces existentes entre los drones formarían la red de datos. Un funcionamiento correcto de la red se produce si dos dispositivos conectados a dos UAVs diferentes son capaces de establecer una conexión por la ruta más eficiente entre los drones y mantenerla aunque la topología de la red cambie. Se debe recordar que el principal objetivo de este trabajo es la exploración de la tecnología SDN para el reenvío de datos, la [Figura 12] refleja el resultado de un proyecto mucho más complejo.

2.5 LINK LAYER DISCOVERY PROTOCOL

Como se ha explicado previamente, los controladores SDN son capaces de tener una visión de la topología de la red en tiempo real. En relación con esta característica, en este punto se va a hablar del protocolo LLDP (Link Layer Discovery Protocol) [11]. Como se podrá observar el capítulos posteriores, el uso de este protocolo resulta útil en determinados puntos de este trabajo y por lo tanto es necesario explicar su funcionamiento.

LLDP es un protocolo de nivel de red estandarizado por el IEEE que se ejecuta entre los dispositivos de red para identificarse y mostrar sus capacidades ante sus vecinos. Se usa principalmente en redes Ethernet cableadas. Algunas de las características que los dispositivos pueden compartir entre ellos son:

- Identificación y características del dispositivo.
- Identificación y características de los puertos del dispositivo.
- VLAN.
- Direcciones IP.
- Capacidades del dispositivo.
- Información de los niveles físico y de enlace.

Las tramas que contienen la información se envían en paquetes Ethernet periódicamente en un intervalo previamente especificado. En la [Figura 13] se puede observar la estructura de una trama del protocolo LLDP. El destino de las tramas es un grupo multicast.

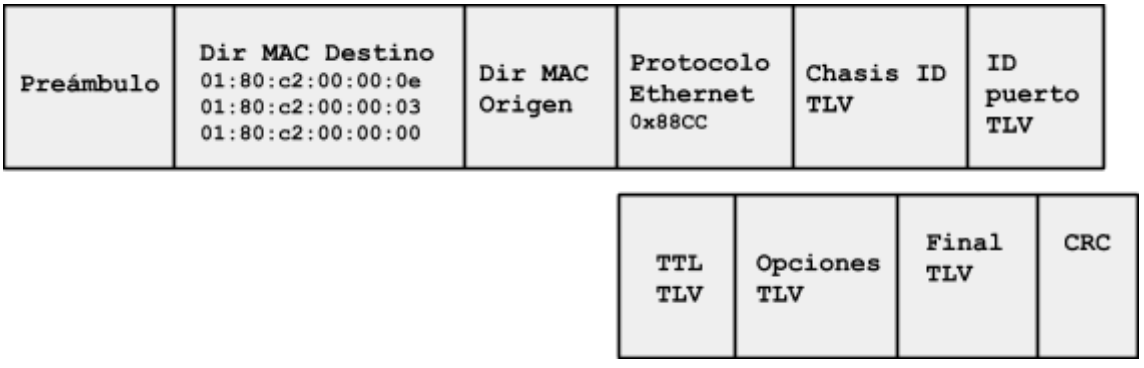


Figura 13 - *Trama Ethernet protocolo LLDP*

Cada trama contiene una LLDPDU (*LLDP Data Unit).* Cada LLDPDU está codificada en formato TLV (*Type-Length-Value).*

Por tratarse de un protocolo de descubrimiento de vecinos, en un principio resultará útil para el controlador a la hora de formar una topología de red inicial.

2.6 RASPBERRY PI

Los equipos que irán embarcados en los MAVs serán Raspberry Pi. Estos dispositivos son pequeños ordenadores desarrollados por la Fundación Raspberry Pi en 2012 [12]. Tienen el tamaño de una tarjeta de crédito y pueden ser conectados a una pantalla y a un teclado o ratón estándares para poder manejarlos. Con ellos se pueden realizar numerosas actividades al igual que en un ordenador más grande, como pueden ser: navegar por internet, editar textos u hojas de cálculo, reproducir contenidos multimedia o jugar a juegos.

En éste trabajo se usará principalmente la Raspberry Pi 3 modelo B y sus características principales son las siguientes:

- Procesador Quad-Core Cortex A8 a 1200MHz
- 1GB de memoria RAM
- 4 puertos USB 2.0
- 40 pines GPIO (General Purpose Input Output)
- 1 puerto HDMI
- 1 puerto Ethernet
- 1 conector de audio o micrófono
- 1 interfaz de cámara CSI (Camera Serial Interface)
- 1 interfaz de pantalla DSI (Display Serial Interface)
- 1 ranura para tarjeta micro SD
- 1 núcleo gráfico 3D
- Conectividad mediante red local 10/100, Bluetooth y WiFi

La diferencia más significativa respecto del modelo anterior, Raspberry Pi 2, es la incorporación del chip BCM43438 que dota la placa con la posibilidad de la realización de comunicaciones inalámbricas mediante Bluetooth o WiFi. En capítulos posteriores de este trabajo se realizará una exploración de esta característica que resulta interesante a la vez que necesaria para el tipo de redes que se desean manejar.

CAPÍTULO 3

USO DE LA TECNOLOGÍA SDN EN REDES MULTISALTO

3.USO DE LA TECNOLOGÍA SDN EN REDES MULTISALTO

Como se ha comentado en capítulos previos el controlador utilizado durante este trabajo será Ryu. Recordar que se trata de un controlador que consta de diferentes componentes programados en el lenguaje Python y que soporta todas las versiones del protocolo OpenFlow.

En este capítulo se van a explicar los requisitos que se exigen en el controlador para una red de UAVs, los componentes ya programados en Ryu y los cambios que se deben realizar sobre ellos para poder cumplir con los requisitos. A su vez se explicará una herramienta necesaria para probar el correcto funcionamiento de la configuración.

3.1 REQUISITOS DE LA RED Y ESCENARIO EXPERIMENTAL

En este trabajo se pretende configurar un controlador que como cerebro de la red de instrucciones de funcionamiento a diferentes switches. La principal característica es que dichos switches van embarcados en vehículos aéreos no tripulados y por lo tanto es importante tener en cuenta algunas particularidades como:

- Es importante la detección de eventos de caída o establecimiento de enlaces. Se debe tener en cuenta que se trata de una red con una alta movilidad y que este tipo de eventos serán muy frecuentes. Es necesario gestionarlos de una manera rápida y lo más transparente posible al usuario final.
- Se debe establecer la ruta más corta entre los dispositivos para una mayor optimización de los recursos. Este punto se encuentra ligado con el anterior, porque en una buena gestión de los eventos de caída y establecimiento se incluye el recálculo de una ruta más corta entre dos dispositivos finales que se encuentren en comunicación.
- Debe tenerse en cuenta la existencia de bucles en la red y se deben gestionar de una manera eficiente.

En este apartado se realizarán pruebas sobre un escenario básico, [Figura 14], en el que podemos encontrar las características anteriores y configurar el controlador de una forma correcta para realizar una buena gestión.

En la [Figura 14] también se puede observar la existencia de un bucle que deberá gestionarse. En este escenario primeramente el tráfico de datos entre el host 1 y el host 2

deberá ser cursado entre los switches 1 y 2. Si el enlace cayera, la red debería ser capaz de reconfigurarse rápidamente y el tráfico de datos sería cursado a través del switch 3.

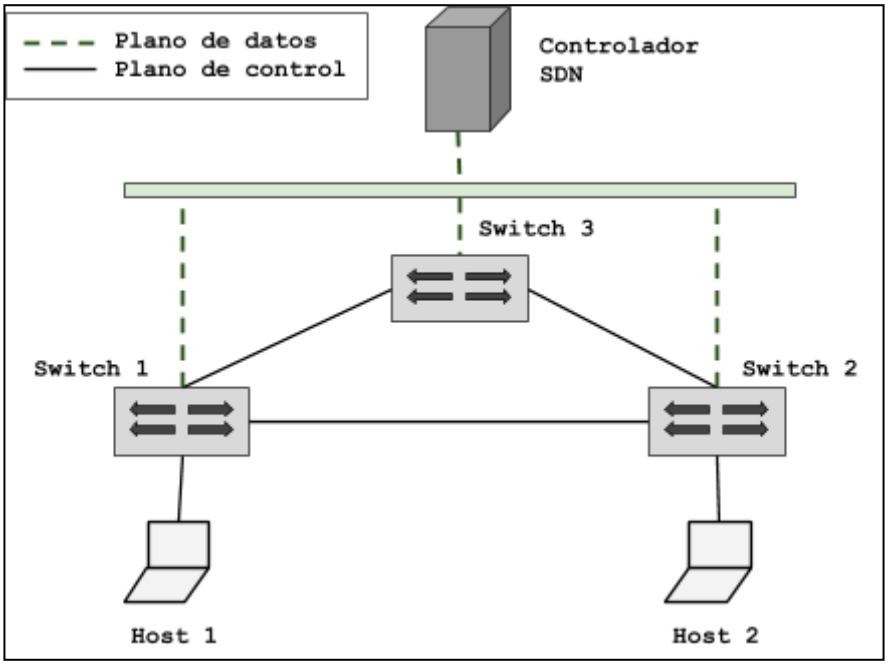


Figura 14 - *Red experimental en triángulo*

En la [Figura 14] también se pueden diferenciar la red de datos y la red de control. En el apartado 2.3.1 se mencionó la existencia de un canal seguro para la comunicación en el plano de control entre los switches y el controlador. Cada switch tiene una interfaz reservada para comunicarse con el controlador. El protocolo de comunicación en la red de control será OpenFlow en la versión 1.3.

Por supuesto, a la hora de programar el controlador se debe tener en cuenta que los switches deben funcionar como switches OpenFlow. En la [Figura 5], en el apartado 2.3.1. , se explicó que cuando el switch recibe un paquete que no coincide con ninguna de las entradas en la tabla de flujo de éste, debe enviar el paquete al controlador para que lo procese y decida qué debe hacerse con él.

3.2 MININET

Mininet [13] es software de código abierto para la emulación de redes que es rápido y fácil de configurar. En las redes SDN es muy importante la parte de emulación ya que puede resultar bastante complicado la implementación real para la investigación. Mininet crea un controlador OpenFlow, switches, hosts y enlaces virtuales y algunas de las ventajas que supone son:

- Es adaptativa a las necesidades del desarrollador pues se pueden experimentar con topologías de red ya implementadas o personalizarlas.
- Muy útil para la investigación de las SDN pues las redes personalizadas pueden ser de gran tamaño y se implementan de una manera rápida y sencilla.
- Sencilla e intuitiva pues para crear las redes personalizadas basta escribir un pequeño script en python o utilizar la herramienta Miniedit, ya incorporada, y exportar el script.

Para este trabajo, esta herramienta ha resultado de gran ayuda para entender de una manera práctica el flujo de mensajes OpenFlow, la configuración de los switches y la comunicación entre el controlador y los switches.

Mininet en su versión 2.2.2, se ejecuta como máquina virtual gracias a la herramienta Oracle VM VirtualVox instalada en Ubuntu 16.04 LTS utilizando los comandos que se encuentran indicados en [13]. Una vez instalada se puede iniciar la máquina virtual con usuario y contraseña *Mininet*. Desde un terminal del ordenador se puede lanzar una sesión ssh contra la máquina virtual que nos permita utilizarla de una manera más fácil.

Todas las pruebas realizadas en este capítulo se realizarán con esta herramienta, la implementación en un escenario real se llevará a cabo en el capítulo siguiente.

3.3 FUNCIONAMIENTO BÁSICO

Como ya se ha mencionado en varias ocasiones, el controlador Ryu tiene implementados diferentes componente en python. Uno de ellos es el script *simple_switch_13.py*. En dicho script se encuentra la implementación de un switch estándar con la versión 1.3 del protocolo OpenFlow.

En este apartado se va a explicar el funcionamiento básico del switch que viene implementado por defecto en Ryu. El funcionamiento de *simple_switch_13.py* queda descrito a continuación en el diagrama de la [Figura 15]. Es importante realizar un entendimiento de este script para poder analizar el funcionamiento del controlador y comprobar las mejoras que se deben realizar sobre él.

Como se puede ver en el diagrama [Figura 15], al principio se intercambiarán los mensajes de inicio de OpenFlow que se vieron el capítulos anteriores. Es decir, los mensaje de HELLO y FEATURES REQUEST/REPLY. Cuando el controlador ya tiene identificado al switch comienza con el siguiente paso.

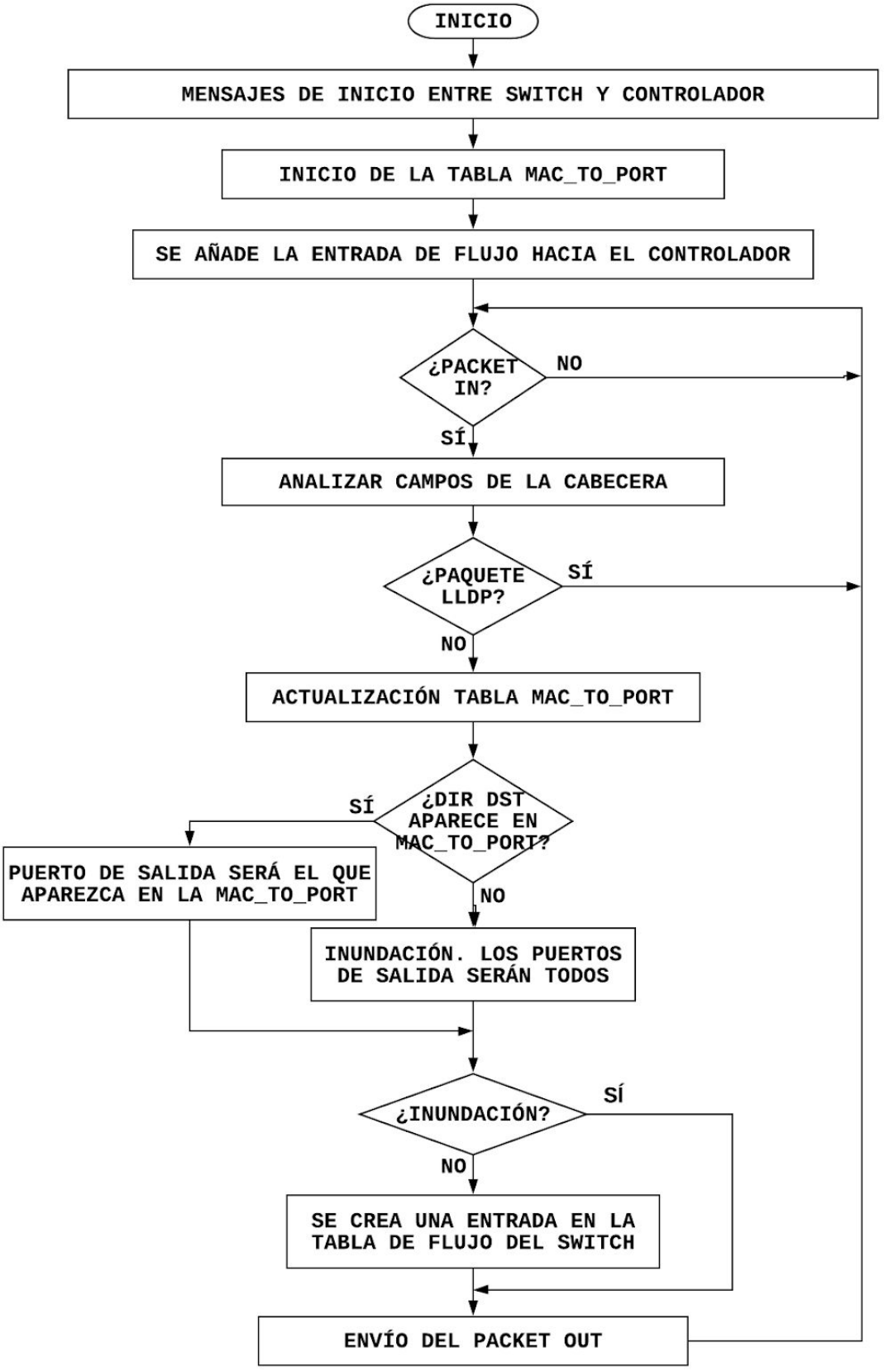


Figura 15 - *Diagrama de flujo de simple_switch_13.py*

En el diagrama se menciona numerosas veces la tabla MAC TO PORT. Se trata de una estructura que relaciona distintas direcciones Ethernet con un puerto y un identificador de switch. Para entender mejor el funcionamiento de esta tabla, que resulta fundamental, así como el del diagrama entero se va a proceder a explicar el procedimiento en un sencillo escenario con solo dos switches y dos host. [Figura 16]

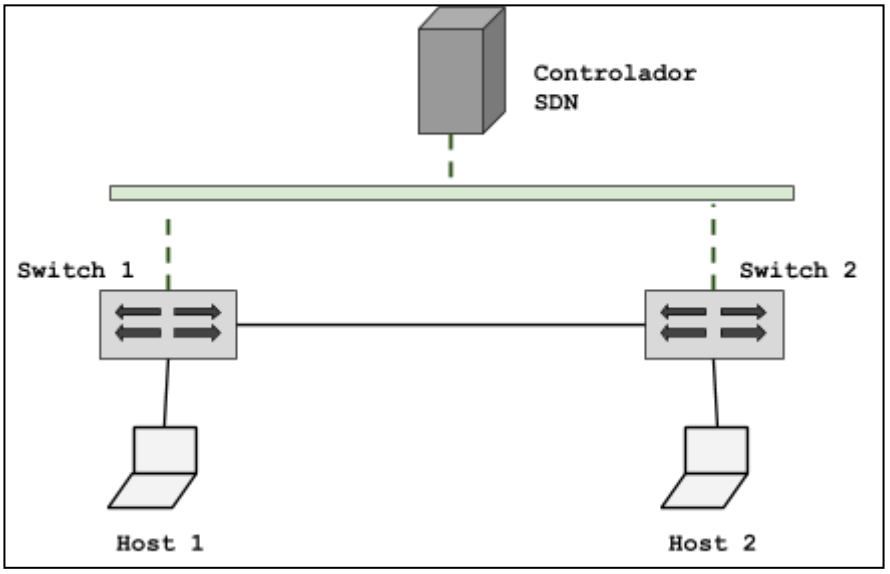


Figura 16 - *Topología 2 switches 2 hosts*

El objetivo es que el host 1 y el host 2 puedan establecer un tráfico de datos entre ambos mediante el comando *ping*, para ello es necesario un tráfico de mensajes de configuración que quedan detallados en la [Figura 17].

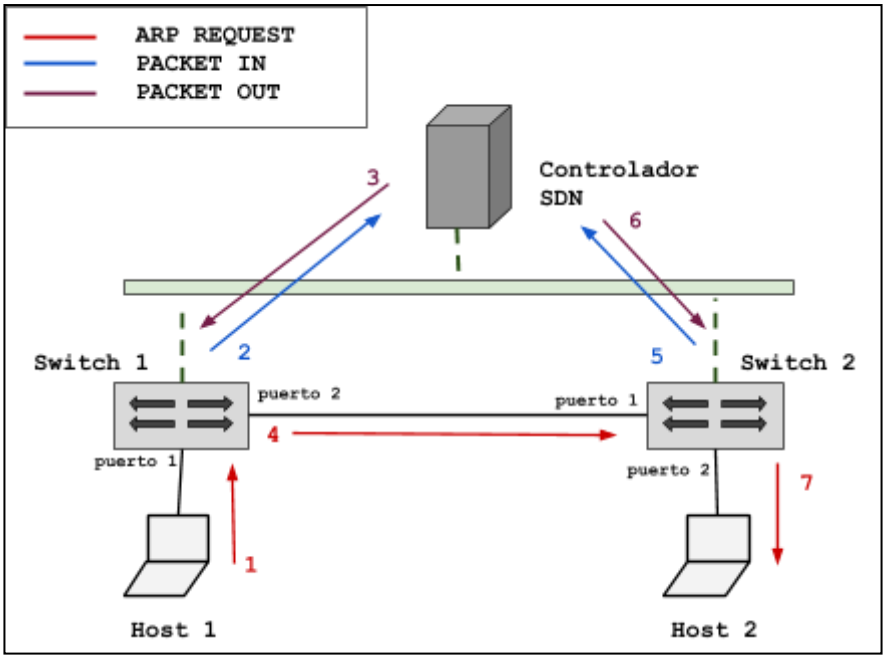


Figura 17 (a) - Conexión 2 host 2 switches

1. El host 1 envía un *ARP REQUEST*. Como la red acaba de iniciarse el primer mensaje será este en el que el host 1 pregunta la dirección Ethernet del host 2.
2. El switch 1 recibe el *ARP REQUEST*. Al no existir entrada con la que coincida en la tabla de flujo envía el paquete al controlador en un *PACKET IN*.

   El controlador analiza lo campos del mensaje y actualiza la tabla *MAC TO PORT* quedando esta de la siguiente manera:

ID SWITCH	MAC	PORT
switch 1	mac host 1	puerto 1

   Así el controlador aprende que el host 1 está ubicado en el puerto 1 del switch 1.

   El destino al tratarse de un *ARP REQUEST* es la dirección de broadcast. El controlador comprueba que no tiene entrada en *MAC TO PORT* para esa dirección de destino y da la orden de inundar.

3. El controlador envía un *PACKET OUT* de respuesta con la orden de inundar.
4. El switch 1 envía el *ARP REQUEST* por todos sus puertos menos por el que lo recibió y por el que tiene conectado el controlador.
5. El switch 2 recibe el *ARP REQUEST*. Al no existir entrada con la que coincida en la tabla de flujo envía el paquete al controlador en un *PACKET IN*.

   El controlador analiza lo campos del mensaje y actualiza la tabla *MAC TO PORT* quedando esta de la siguiente manera:

ID SWITCH	MAC	PORT
switch 1	mac host 1	puerto 1
switch 2	mac host 1	puerto 1

   Así el controlador aprende que el host 1 está ubicado en el puerto 1 del switch 2.

   El destino al tratarse de un *ARP REQUEST* es la dirección de broadcast. El controlador comprueba que no tiene entrada en *MAC TO PORT* para esa dirección de destino y da la orden de inundar.

6. El controlador envía un *PACKET OUT* de respuesta con la orden de inundar.
7. El host 2 recibe el *ARP REQUEST*.

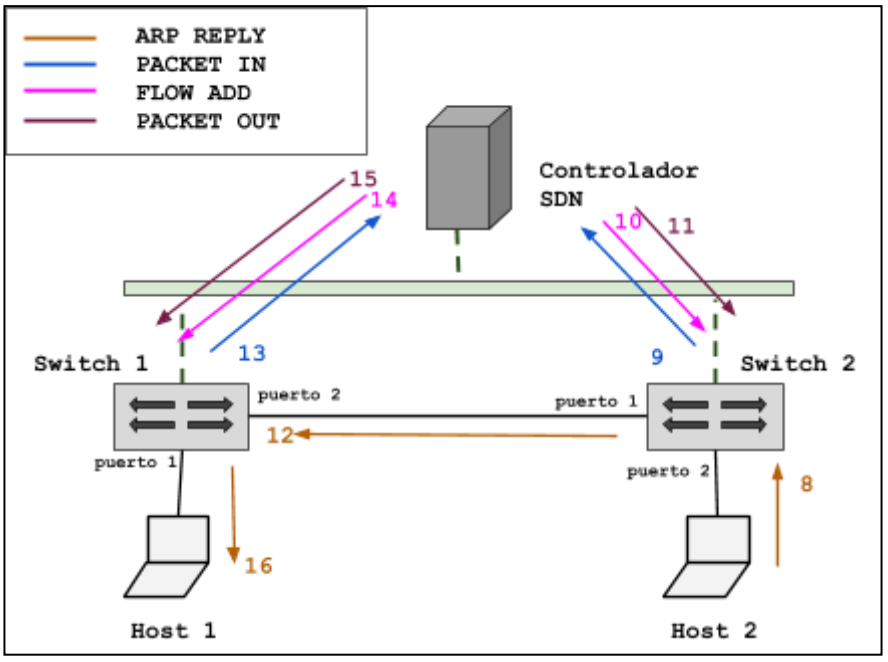


Figura 17 (b) - Conexión 2 host 2 switches

8. El host 2 envía el *ARP REPLY* en respuesta al host 1.
9. El switch 2 recibe el *ARP REPLY*. Al no existir entrada con la que coincida en la tabla de flujo envía el paquete al controlador en un *PACKET IN*.

   El controlador analiza lo campos del mensaje y actualiza la tabla *MAC TO PORT* quedando esta de la siguiente manera:

ID SWITCH	MAC	PORT
switch 1	mac host 1	puerto 1
switch 2	mac host 1	puerto 1
switch 2	mac host 2	puerto 2

   Así el controlador aprende que el host 2 está ubicado en el puerto 2 del switch 2.

   En este caso el destino es la mac del host 1 y en la tabla *MAC TO PORT* sí existe una entrada para el switch 2 con dicho destino.

10. El controlador añade una entrada en la tabla de flujo del switch 2 para que la siguiente vez que reciba un mensaje con destino el host 1 por el puerto 2 no tenga que preguntar al controlador. La tabla del switch 2 queda de la siguiente manera.

FLOW TABLE SWITCH 2			
IN PORT	**MAC SCR**	**OUT PORT**	**MAC DST**
2	mac host 2	1	mac host 1

11. El controlador envía un *PACKET OUT* de respuesta con la orden de que debe enviarlo por el puerto 1.
12. El switch 2 envía el *ARP REPLY* por su puerto 1.
13. El switch 1 recibe el *ARP REPLY.* Al no existir entrada con la que coincida en la tabla de flujo envía el paquete al controlador en un *PACKET IN.*

    El controlador analiza lo campos del mensaje y actualiza la tabla *MAC TO PORT* quedando esta de la siguiente manera:

ID SWITCH	MAC	PORT
switch 1	mac host 1	puerto 1
switch 2	mac host 1	puerto 1
switch 2	mac host 2	puerto 2
switch 1	mac host 2	puerto 2

    Así el controlador aprende que el host 2 está ubicado en el puerto 2 del switch 1.

    En este caso el destino es la mac del host 1 y en la tabla *MAC TO PORT* sí existe una entrada para el switch 1 con dicho destino.

14. El controlador añade una entrada en la tabla de flujo del switch 1 para que la siguiente vez que reciba un mensaje con destino el host 1 por el puerto 2 no tenga que preguntar al controlador. La tabla del switch 2 queda de la siguiente manera.

FLOW TABLE SWITCH 1			
IN PORT	**MAC SCR**	**OUT PORT**	**MAC DST**
2	mac host 2	1	mac host 1

15. El controlador envía un *PACKET OUT* de respuesta con la orden de que debe enviarlo por el puerto 1.

16. El host 1 recibe el *ARP REPLY*.

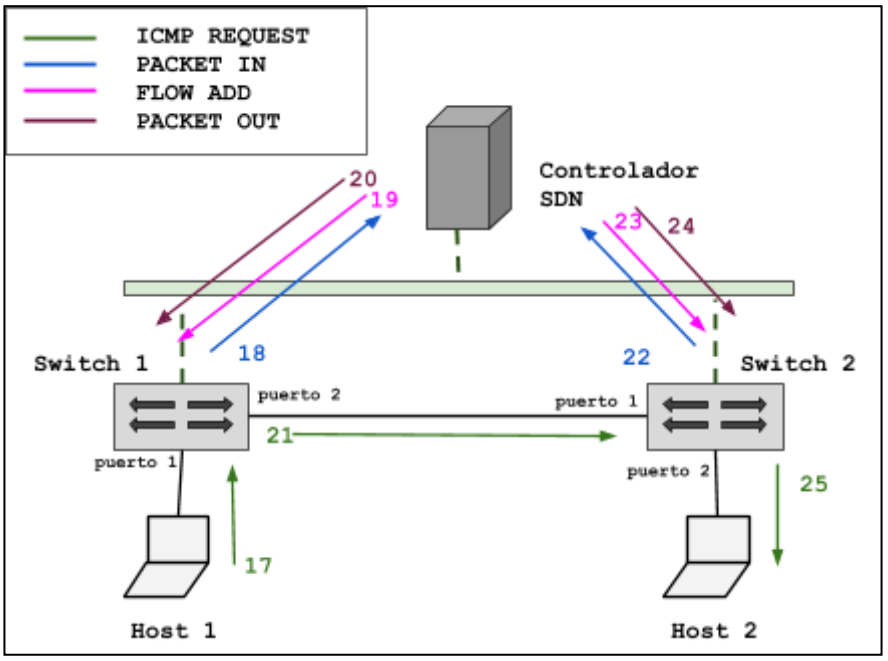


Figura 17 (c) - Conexión 2 host 2 switches

17. El host 1 envía el *ICMP REQUEST*.
18. El switch 1 recibe el *ICMP REQUEST*. Al no existir entrada con la que coincida en la tabla de flujo envía el paquete al controlador en un *PACKET IN*.

    El controlador analiza lo campos del mensaje. Ya existe una entrada en *MAC TO PORT* para los mismos campos.

    En este caso el destino es la mac del host 2 y en la tabla *MAC TO PORT* sí existe una entrada para el switch 1 con dicho destino.

19. El controlador añade una entrada en la tabla de flujo del switch 1 para que la siguiente vez que reciba un mensaje con destino el host 2 por el puerto 1 no tenga que preguntar al controlador. La tabla del switch 1 queda de la siguiente manera.

FLOW TABLE SWITCH 1			
IN PORT	**MAC SCR**	**OUT PORT**	**MAC DST**
2	mac host 2	1	mac host 1
1	mac host 1	2	mac host 2

20. El controlador envía un *PACKET OUT* de respuesta con la orden de que debe enviarlo por el puerto 2.

21. El switch 1 envía el *ICMP REQUEST* por su puerto 2.
22. El switch 2 recibe el *ICMP REQUEST.* Al no existir entrada con la que coincida en la tabla de flujo envía el paquete al controlador en un *PACKET IN.*

    El controlador analiza lo campos del mensaje. Ya existe una entrada en *MAC TO PORT* para los mismos campos.

    En este caso el destino es la mac del host 2 y en la tabla *MAC TO PORT* sí existe una entrada para el switch 2 con dicho destino.

23. El controlador añade una entrada en la tabla de flujo del switch 2 para que la siguiente vez que reciba un mensaje con destino el host 2 por el puerto 1 no tenga que preguntar al controlador. La tabla del switch 2 queda de la siguiente manera.

FLOW TABLE SWITCH 2			
IN PORT	**MAC SCR**	**OUT PORT**	**MAC DST**
2	mac host 2	1	mac host 1
1	mac host 1	2	mac host 2

24. El controlador envía un *PACKET OUT* de respuesta con la orden de que debe enviarlo por el puerto 2.
25. El host 2 recibe el *ICMP REQUEST.*

Cuando el host 2 conteste con el ICMP REPLY el tráfico ya irá directo hasta el host 1 sin que que el tráfico tenga que pasar por el controlador puesto que las tablas de los switches ya están completamente configuradas para cursar el tráfico entre ambos equipos finales.

Con este simple programa se puede aprender cómo puede aprender en controlador dónde están ubicados los equipos finales y crear los flujos correctamente. También se aprende cómo y en qué orden se establecen los flujos en los switches OpenFlow.

3.3.1 PROBLEMAS

Una vez se ha entendido el funcionamiento básico se puede hablar de qué mejoras se necesitan implementar.

La red de dos equipos finales y dos switches resulta una red demasiado sencilla. El primer problema que surge al cambiar a una topología en triángulo es la aparición de un bucle. Las redundancias o bucles son necesarias en las redes para tener enlaces

secundarios que permitan que una conexión no se interrumpa por un enlace caído. Con el script *simple_switch_13.py* no hay manera de controlar el bucle cuando un switch inunda. La tabla *MAC TO PORT* se actualiza constantemente con información errónea, En la [Figura 18] se puede entender esto.

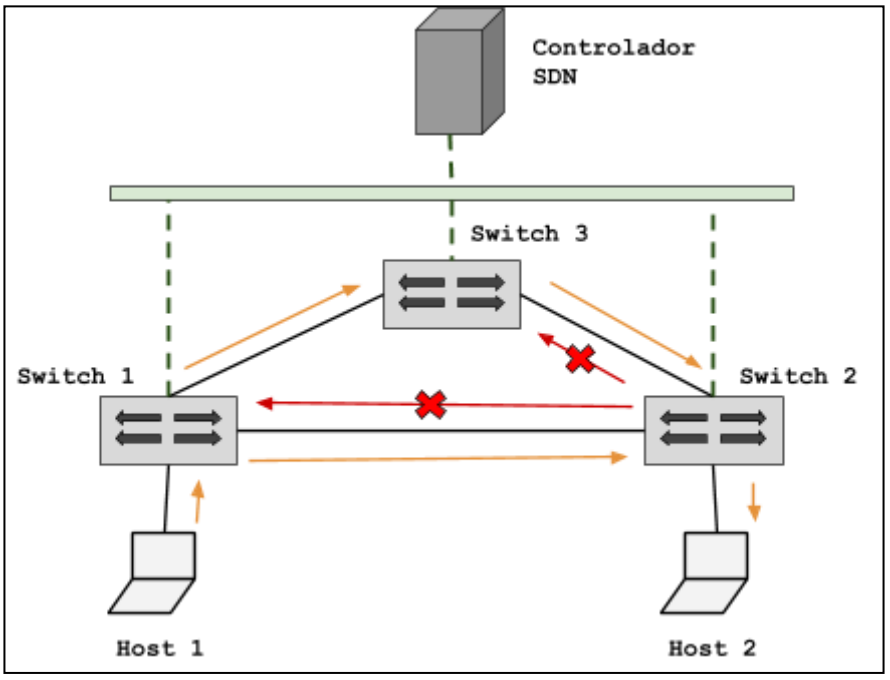


Figura 18 - *Problema bucle triángulo topología 2 host 2 switches*

Cuando el switch 2 reciba el mensaje procedente del switch 3 lo enviará hacia el switch 1 y el controlador aprenderá que el host 1 está por la dirección del switch 2, lo cual es erróneo. Lo mismo pasa hacia el switch 3. A parte de la información errónea, los switches no pararían de reenviar el mismo mensaje por la red inundándola constantemente y provocando que esta se quede colgada.

Se debe tener en cuenta que a la vez que se gestionan las redundancias debe elegirse la ruta más corta o rápida entre los dispositivos finales que desean comunicarse. El tráfico de datos debe ir entre los switches 1 y 2 sin pasar por el switch 3.

Por último se debe implementar un gestor de eventos de caída y establecimiento de enlace. Es decir, si se cae el enlace entre los switches 1 y 2, se debe gestionar el evento y realizar una reconfiguración de la red para que el tráfico ahora pase a través del switch 3. Si el enlace volviera a estar activo después de un tiempo se debe retomar la configuración inicial.

3.4 ENFOQUE 1: SPANNING TREE PROTOCOL

Para evitar los bucles se pensó en usar otro de los componentes de Ryu, *simple_switch_stp_13.py*. Este script se diferencia del anterior en que implementa el protocolo STP (Spanning Tree Protocol) [14]. Este es un protocolo red de la capa de enlace y su principal objetivo es la gestión de redundancias en las redes. Los switches se envían entre ellos unos mensajes específicos para este protocolo. Estos mensajes se conocen como BPDUs y gracias a ellos la red se configura de forma que, como el propio nombre del protocolo indica, se construye una topología de árbol. Esto significa que existe un nodo raíz y que todo el tráfico debe pasar por él.

El nodo raíz de la red es aquel que tenga un identificador menor. Los demás switches designan uno sus puertos como puerto raíz, que es por el puerto a través de cuál se comunican con el nodo raíz. El resto de puertos serán designados o bloqueados según la configuración de la red. De esta manera se consigue la topología de árbol.

Teóricamente en la topología triangular con la que se experimenta en este capítulo, el resultado de la ejecución del protocolo es la indicada en la [Figura 19].

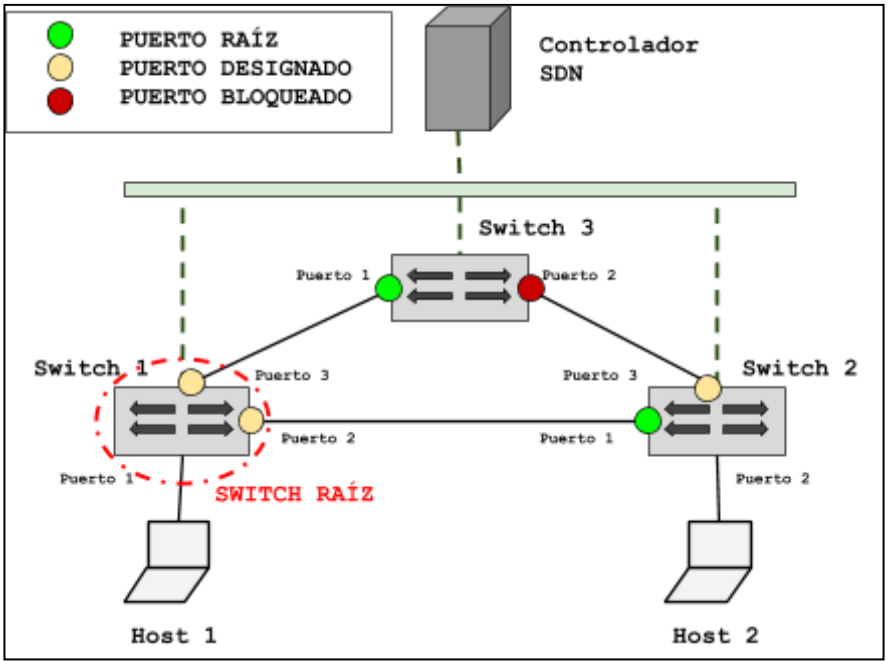


Figura 19 - *STP en 3 switches 2 host*

La red se ha configurado de la siguiente manera:

- El nodo raíz será el switch 1 porque es el que tiene un menor identificador y configura todos sus puertos como puertos designados.

- Los switches 2 y 3 configuran sus puertos raíz que serán aquellos que tienen un menor coste hasta el nodo raíz.
- Una vez se ha configurado el nodo raíz y los puertos raíz se deben configurar los puertos designados del resto de segmentos de la red. En este caso se tiene el enlace entre el switch 2 y el switch 3.
- La norma indica que será el puerto designado de un segmento aquel que tenga un menor coste hasta el nodo raíz. En este caso ambos puertos del segmento tienen el mismo coste y por lo tanto será designado el puerto que pertenezca al switch con un menor identificador, es decir, el switch 2.
- Por último, aquellos puertos que no hayan sido configurados como raíz o designados deben bloquearse.

Tras realizar algunas pruebas sobre la topología se encontraron diferentes problemas. Uno de ellos es que si hubiera un dispositivo final, un host 3, conectado al switch 3 que quisiera comunicarse con el host 2 el tráfico no sería cursado por la ruta más corta entre ambos. Este problema es mayor, cuanto más grande es el tamaño de la red, pues podría darse un gran retardo en la comunicación entre dos usuarios finales porque el tráfico no sea cursado por la ruta más corta.

Tampoco debe olvidarse que se trata de una red de alta movilidad. La red se debe reconfigurar cada vez que exista un cambio en ella, como la caída o el movimiento de un nodo y de todos sus enlaces. Esto puede suponer un tráfico de control excesivo a la vez que innecesario que acarrea un retardo aún mayor para la comunicación entre usuarios. Por último, también se debe tener en cuenta que los equipos embarcados, como se comentó en el capítulo anterior, serán Raspberry Pi. Estos dispositivos están limitados en memoria y en la capacidad de su procesador y por lo tanto no se considera conveniente que todo el tráfico de la red tenga que pasar por un dispositivo, pues podrían producirse cuellos de botella en la parte central de la red.

Por estas razones el protocolo STP fue descartado para la realización de este trabajo. Se puede afirmar que se necesita una solución que no cree topologías de árbol y que nos asegure que se va a poder emplear la ruta más corta o rápida entre dos dispositivos.

3.5 ENFOQUE 2: MODIFICACIÓN IMPLEMENTACIÓN BÁSICA

En este apartado se va a implementar una solución modificando el script básico de *simple_switch_13.py.* El funcionamiento completo queda detallado en el diagrama de la [Figura 20]. Esta solución implementa un control de las inundaciones y una gestión de los eventos de modificación de enlaces y/o puertos.

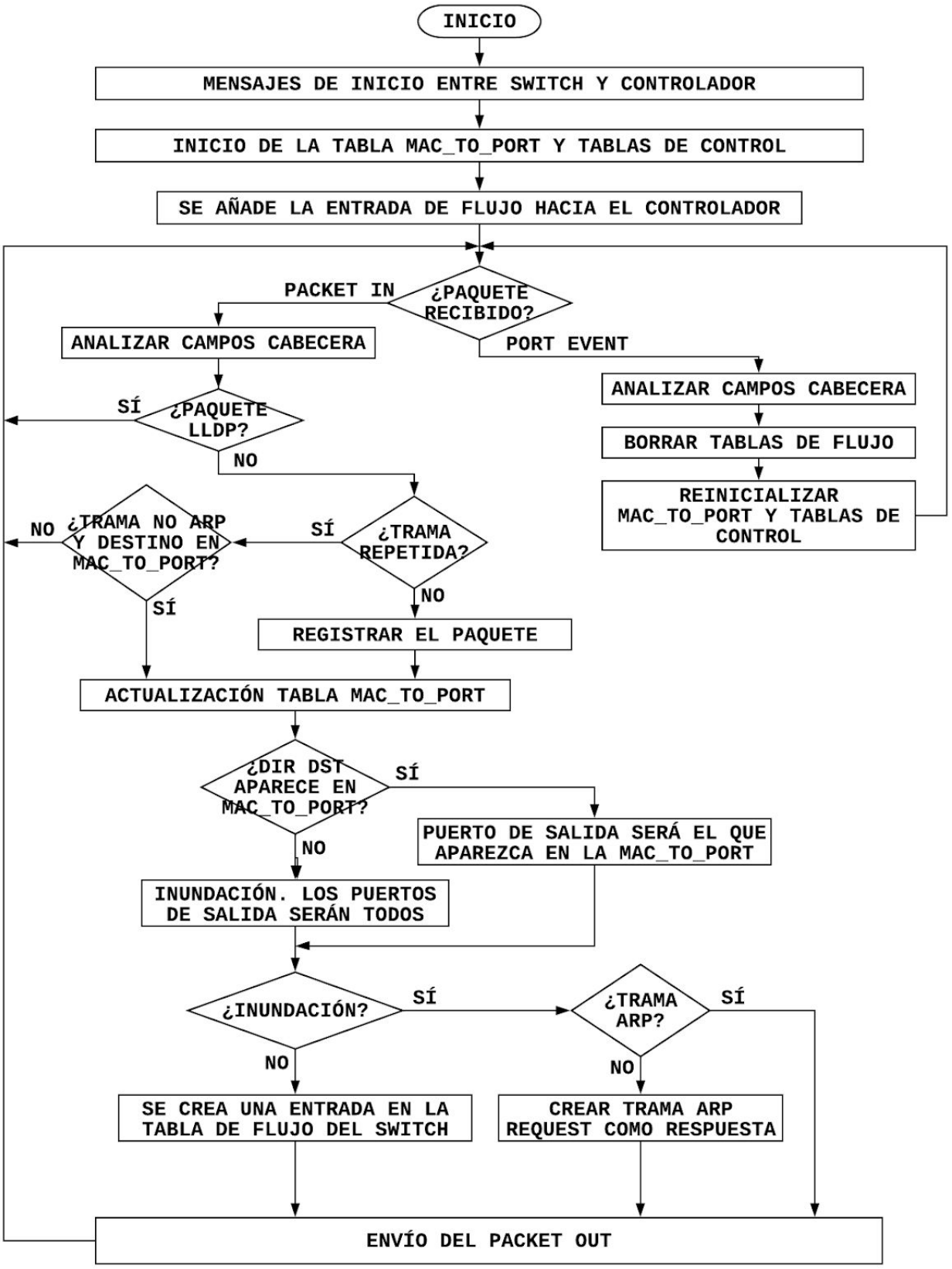


Figura 20 - *Diagrama de flujo solución 2*

Esta solución se basa en la idea de que el controlador lleve un registros de los mensajes que ha recibido para saber si un dispositivo ya se lo había mandando en forma de packet in previamente. El registro se borra cuando se produce un cambio en la red. A continuación en la [Figura 21], se detalla el intercambio de mensajes para mostrar el control de la redundancia.

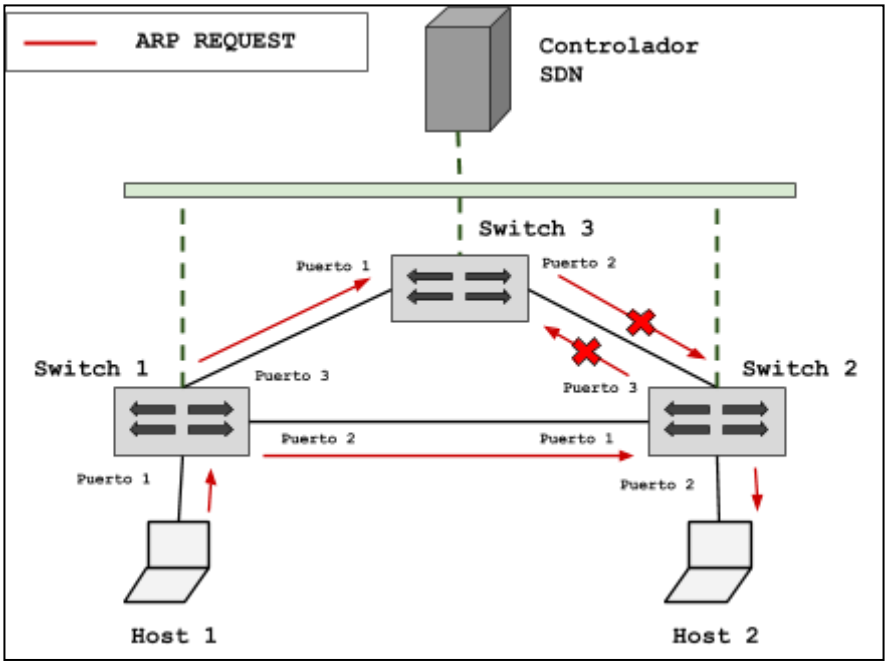


Figura 21 - *Conexión 3 switches 2 host*

1. El host 1 envía un *ARP REQUEST*. Como la red acaba de iniciarse el primer mensaje será este en el que el host 1 pregunta la dirección Ethernet del host 2.
2. El switch 1 recibe el *ARP REQUEST*. Al no existir entrada con la que coincida en la tabla de flujo envía el paquete al controlador en un *PACKET IN*. El controlador procesa el paquete, actualiza sus tablas y enviar un *PACKET OUT* con la orden de inundar. Por lo tanto el switch 1 reenvía el mensaje por los puertos 1 y 3.
3. Los switches 2 y 3 reciben el mensaje, ambos envían el paquete al controlador en un *PACKET IN*. El controlador realiza las mismas operaciones que en apartado anterior y les contesta con la orden de inundar. Los switches 2 y 3 envian el paquete por todos sus puertos menos por el de entrada. En este punto la tabla *MAC TO PORT* y una tabla de control de mensajes implementada contienen la siguiente información:

ID SWITCH	MAC	PORT
switch 1	mac host 1	puerto 1
switch 2	mac host 1	puerto 1
switch 3	mac host 2	puerto 1

ID SWITCH	IP ORIGEN	IP DESTINO	PROTOCOLO
switch 1	IP host 1	IP host 2	ARP
switch 2	IP host 1	IP host 2	ARP
switch 2	IP host 2	IP host 2	ARP

4. En este punto es cuando los switches reciben los mensajes repetidos. El switch 2 recibe el *ARP REQUEST* procedente del switch 3. Lo envía al controlador y este analiza que el switch 2 ya recibió un paquete con mismas direcciones IP de origen y destino y mismo protocolo. Por lo tanto el paquete es descartado. Lo mismo ocurre en el switch 3 cuando recibe el paquete repetido de parte del switch 2.

De esta manera queda controlada la redundancia de la red y la ruta que será establecida entre los equipos finales será la más corta o rápida puesto que queda registrado aquel mensaje que se recibe primero. El posterior intercambio de mensajes y establecimiento de los flujos en los dispositivos es similar al del apartado anterior.

Una vez se ha conseguido establecer la ruta entre ambos se procede a gestionar los eventos de modificaciones de enlace. Ambos extremos del enlace notifican al controlador la modificación del estado de éste. El controlador entonces borra todas las entradas de flujo que tuvieran los switches y la red debe reconfigurarse.

El proceso de reconfiguración es similar al de inicio pero se debe tener en cuenta un importante detalle. Como se ha mencionado, al inicio de la conexión los equipos finales mandan una trama *ARP REQUEST* puesto que nos conocen las direcciones Ethernet de los destinos. En una reconfiguración, los equipos finales siguen enviando tráfico que ya no es *ARP*. Es entonces cuando el controlador controla si debe dar orden de inundar como resultado de una trama que no es un *ARP REQUEST*. En el caso de que se dé esta condición es el controlador el que fuerza esta trama para que la red pueda configurarse correctamente.

Una vez se probó que la conexión mediante *Ping* funcionaba, se comenzaron a realizar pruebas de tráfico de datos con *Iperf* tanto sobre TCP como sobre UDP y los resultados obtenidos son los siguientes. En la [Figura 22] se observa el resultado del

tráfico TCP entre los dos dispositivos finales. El enlace entre los switch 1 y 2 se activa y desactiva cada 20 segundos aproximadamente. En cada modificación del enlace se aprecia una caída momentánea del ancho de banda, pero se recupera rápidamente.

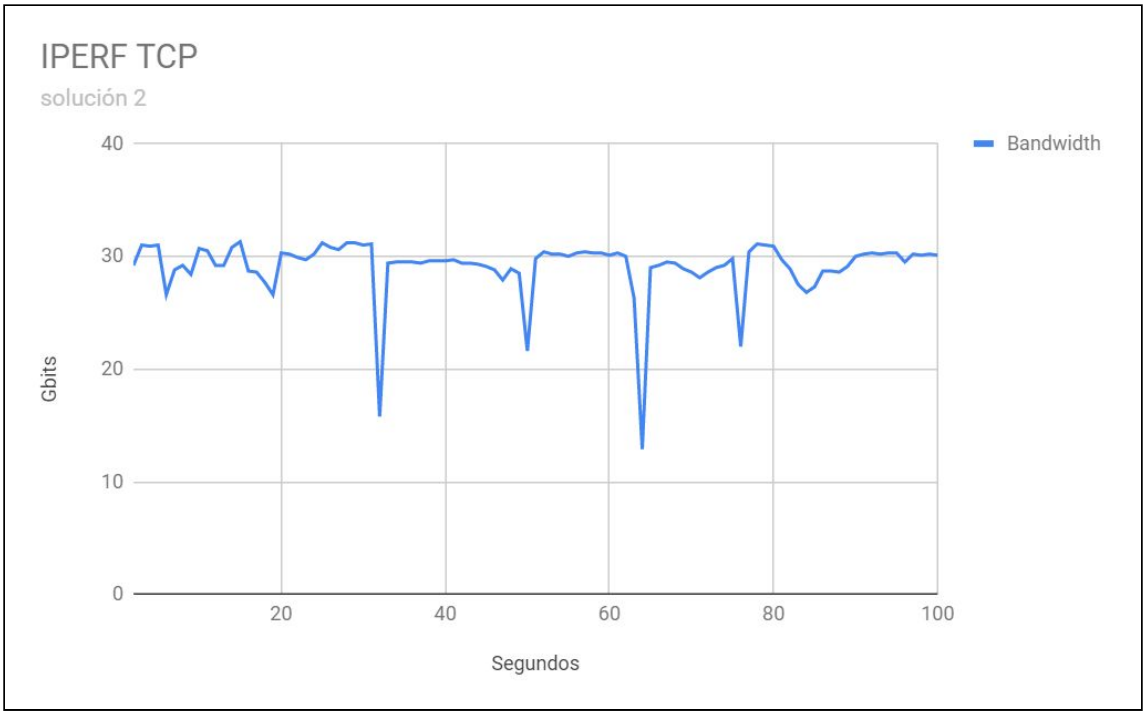
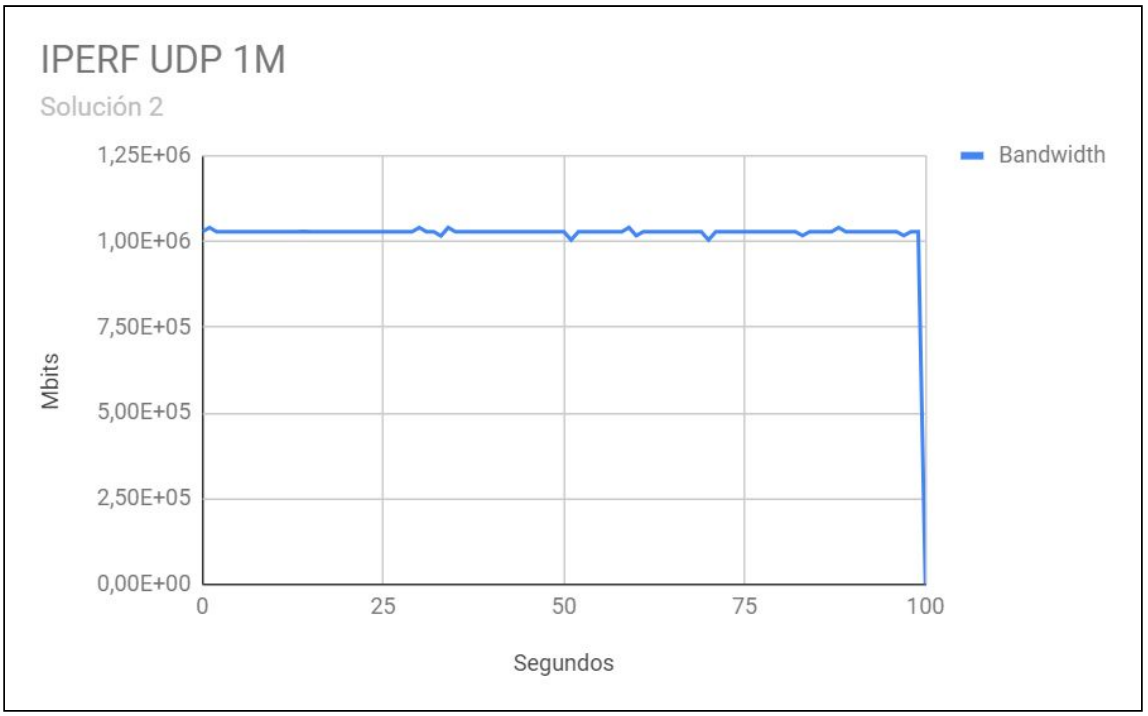
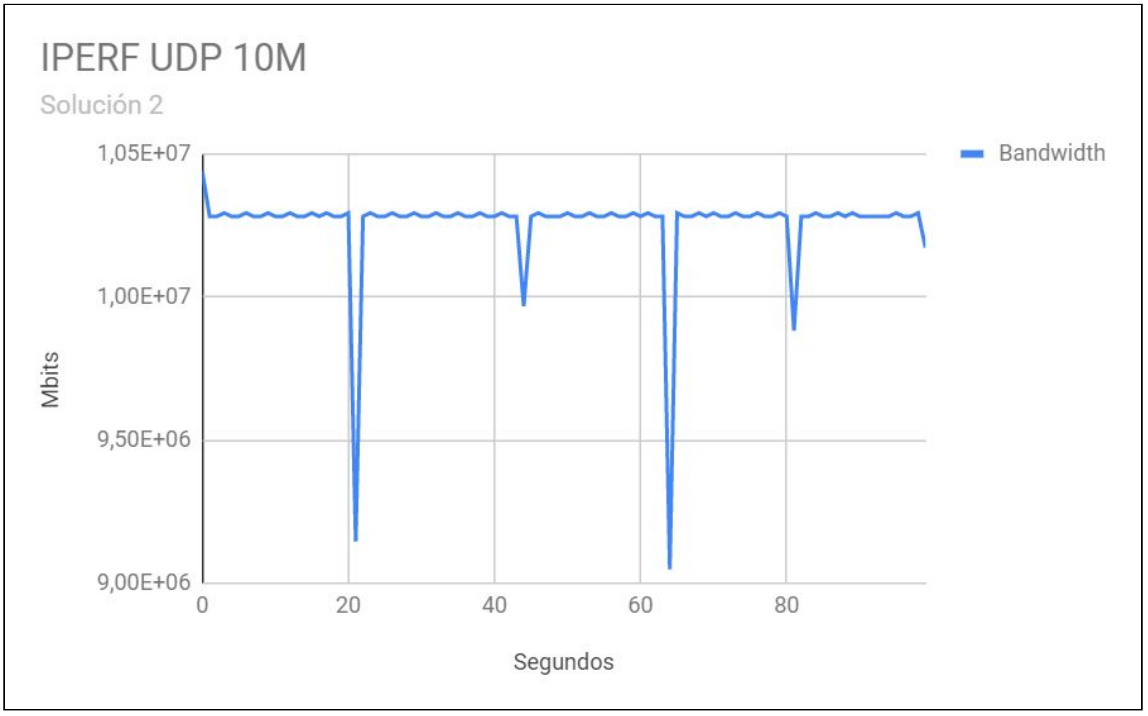


Figura 22 - *Iperf TCP solución 2*

Seguidamente se realizó la misma prueba sobre UDP. En este caso se escogen diferentes anchos de banda: 1, 10, 50 y 100 Mbits/s. En las [Figura 23] a continuación se puede apreciar el comportamiento de la red para los diferentes anchos de banda.


IPERF UDP 1M
Solución 2
1,25E+06
1,00E+06
7,50E+05
5,00E+05
2,50E+05
0,00E+00
Mbits
0
25
50
75
100
Segundos
Bandwidth


IPERF UDP 10M
Solución 2
1,05E+07
1,00E+07
9,50E+06
9,00E+06
Mbits
0
20
40
60
80
Segundos
Bandwidth

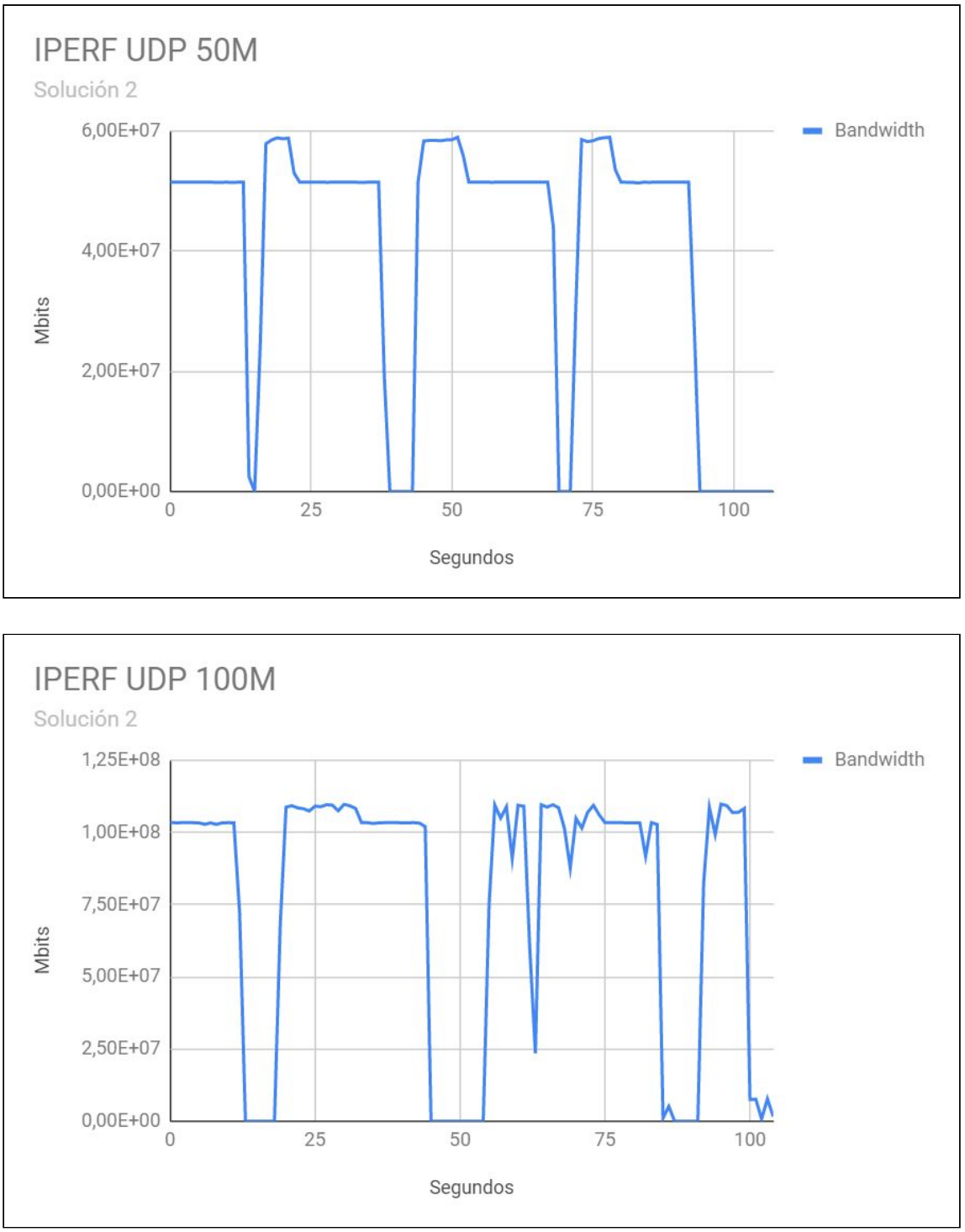


Figura 23 - *Iperf UDP solución 2*

En los gráficos se puede apreciar cómo según se aumenta el ancho de banda, aumenta también el tiempo de recuperación de la red. Para 1M no existe apenas variación. Para 10M se puede apreciar caídas del ancho de banda cuando se modifica el estado de los enlaces, pero no se llega a perder la comunicación nunca. En cambio para

50M y 100M sí que se puede observar como la comunicación queda interrumpida y que el tiempo de recuperación es cada vez mayor.

Este problema no se demostró durante la conexión TCP puesto que este protocolo implementa un control de las tramas, que evita la saturación de la red. Por lo tanto nuestra red SDN no se encuentra saturada de tráfico mientras se borran las entradas de las tablas de flujos y se configura de nuevo la red con los primeros mensajes. Sin embargo, en una conexión UDP la información se envía continuamente y por lo tanto a mayor ancho de banda mayor saturación y mayor tiempo de convergencia.

Después de evaluar estos resultados se decidió que ésta tampoco es una solución válida para este trabajo. Se debe encontrar una solución más eficiente en la que el tiempo de convergencia sea menor dado el gran número de reconfiguraciones que pueden producirse en una red de tan alta movilidad.

3.6 ENFOQUE 3: DESCUBRIMIENTO INICIAL DE LA RED

Una de las ventajas de las redes SDN, como se comentó en apartados anteriores, es que es posible el descubrimiento de la topología de la red antes del comienzo del tráfico de datos. Esta va a ser la clave de la solución propuesta en este capítulo. Mediante el protocolo LLDP se podrá llevar a cabo el descubrimiento de vecinos y el controlador podrá crear un grafo de la red mediante una librería de Python. Esta librería tiene métodos implementados para actualizar el grafo según las diferentes modificaciones que pueden darse en la red y otros para el cálculo de la ruta más corta entre dos nodos.

3.6.1 NETWORKX

NetworkX [15] es un paquete Python para la creación y manipulación de estructuras diseñado para que el usuario trabaje con grafos dónde se representan las diferentes topologías de red. Se trata de un software libre bajo la licencia BSD. Sus principales características son:

- Posee clases y estructuras de datos para los grafos.
- Existe la posibilidad de construir los grafos de forma aleatoria o de forma incremental.
- Los bordes pueden contener datos.
- Posibilidad de crear subgrafos.
- Gran flexibilidad hacia el usuario.

Para este trabajo resultará de gran utilidad aquellas funciones para añadir y quitar nodos del grafo y aquella que calcula la ruta más corta. Esta última función

calcula dicha ruta según el algoritmo de Dijkstra. El algoritmo de Dijkstra o de caminos mínimos determina el camino más corto desde un vértice hacia el resto de vértices de un grafo. Gracias a este algoritmo se pueden resolver topologías de red muy complejas.

3.6.2 IMPLEMENTACIÓN DEL ENFOQUE 3

En este punto se explicará la implementación que será utilizada durante las pruebas con los dispositivos embarcados. La herramienta explicada en el apartado anterior es fundamental en el desarrollo de esta solución.

Al igual que en los otros puntos de este trabajo, en la [Figura 24] se puede observar el diagrama de funcionamiento de la implementación. En este se pueden encontrar varias diferencias respecto a los diagramas de las soluciones anteriores. Por ejemplo, la tabla *MAC TO PORT* ya no es necesaria. El método de establecimiento de las rutas ya no es una a una cada vez que se va aprendiendo sino que al tener todo la información de la red disponible desde el principio el método de establecimiento de rutas es más rápido y directo.

La primera diferencia que se puede observar respecto los diagramas anteriores es que en este caso se produce un descubrimiento de la red al inicio de la configuración del sistema. La información obtenida de la red es añadida al grafo y éste entonces es construido con todos los nodos y todos los enlaces de la red.

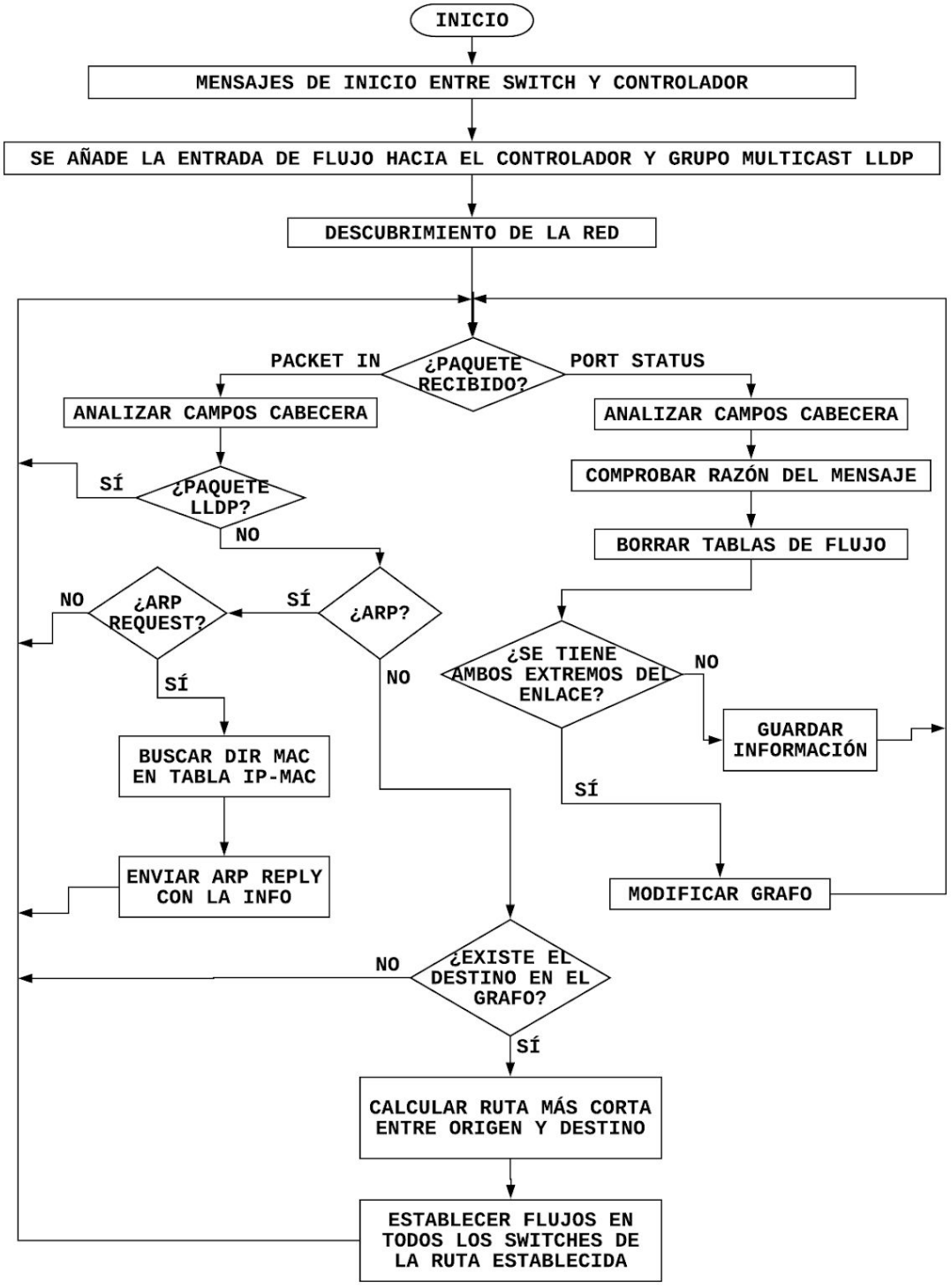
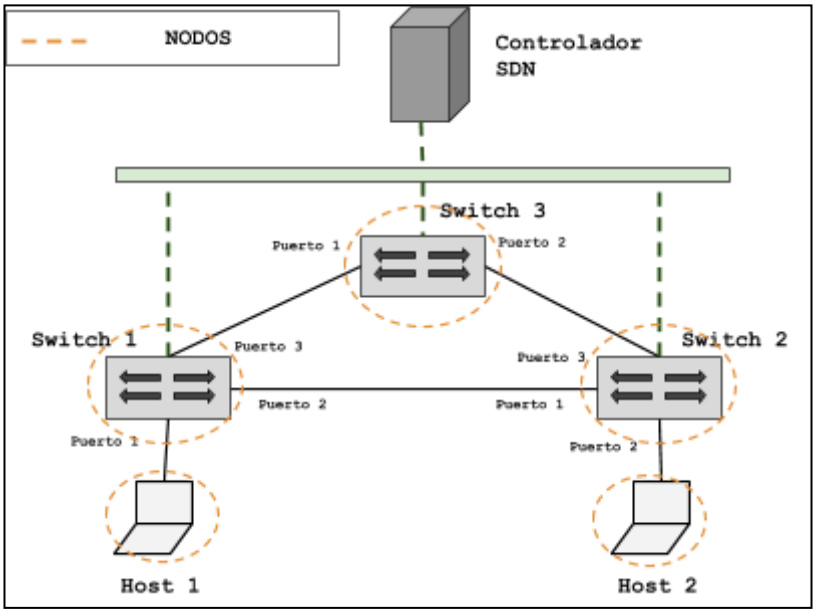


Figura 24 - *Diagrama de flujo de la solución final*

A continuación se detalla la información añadida al grafo. En la [Figura 25] se puede observar la configuración de la red y los nodos existentes inicialmente según el diagrama de *3 switches y 2 host* con el que se está trabajando en éste capítulo. También se detalla cómo se incluyen los enlaces en el grafo.



ID NODO 1	ID NODO 2	PUERTO
Host 1	Switch 1	
Switch 1	Host 1	Puerto 1
Host 2	Switch 2	
Switch 2	Host 2	Puerto 2
Switch 1	Switch 2	Puerto 2
Switch 2	Switch 1	Puerto 1
Switch 1	Switch 3	Puerto 3
Switch 3	Switch 1	Puerto 1
Switch 2	Switch 3	Puerto 3
Switch 3	Switch 2	Puerto 2

Figura 25 - *Configuración inicial de la red añadida al grafo*

Una vez se ha inicializado el controlador y se conoce la topología de red comienza el tráfico de datos. A continuación se va a explicar qué ocurre cuando el *host 1* comience la comunicación y cuales son las diferencias respecto a las soluciones anteriores. En la [Figura 26] se puede observar un esquema del tráfico de mensajes entre los dispositivos para un comando *ping*.

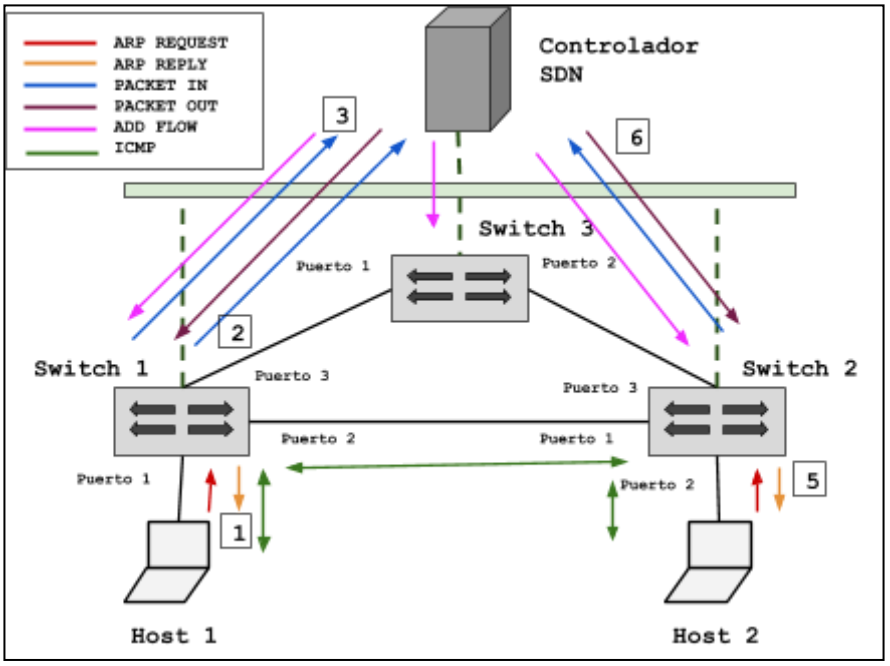


Figura 26 - *Conexión 3 switch 2 host solución final*

1. El host 1 envía un *ARP REQUEST*. Como la red acaba de iniciarse el primer mensaje será este en el que el host 1 pregunta la dirección Ethernet del host 2.
2. El switch 1 recibe el *ARP REQUEST*. Al no existir entrada con la que coincida en la tabla de flujo envía el paquete al controlador en un *PACKET IN*. El controlador analiza el paquete. Comprueba que se trata de una trama *ARP REQUEST* y consulta una tabla en la que se relacionan las direcciones IP con las direcciones Ethernet de los dispositivos. Envía un *PACKET OUT* al switch 1 que contiene una trama *ARP REPLY* con la información que necesita el host 1. El switch 1 envía el *ARP REPLY* al host 1.
3. El host 1 comienza el tráfico *ICMP* y envía un *ICMP REQUEST* con destino el host 2. El switch 1 recibe el *ICMP REQUEST*. Al no existir entrada con la que coincida en la tabla de flujo envía el paquete al controlador en un *PACKET IN*. El controlador analiza el paquete. Comprueba que no se trata de una trama *ARP*, entonces comprueba que si el destino de la trama es un nodo existente en el grafo. En este caso sí lo es y por lo tanto calcula la ruta más corta entre los dos dispositivos finales. Una vez calculada la ruta establece las entradas en la tabla

de flujo necesarias para la comunicación. Se configuran los switches 1 y 2 de la siguiente manera:

FLOW TABLE SWITCH 1			
IN PORT	**MAC SCR**	**OUT PORT**	**MAC DST**
2	mac host 2	1	mac host 1
1	mac host 1	2	mac host 2

FLOW TABLE SWITCH 2			
IN PORT	**MAC SCR**	**OUT PORT**	**MAC DST**
2	mac host 2	1	mac host 1
1	mac host 1	2	mac host 2

Las tablas quedan configuradas con los mismos datos que en las soluciones anteriores pero de una manera directa.

4. Una vez configurados los flujos el tráfico del host 1 llega al host 2. En este momento el host 2 necesariamente envía un *ARP REQUEST*. Se debe recordar que la respuesta al *ARP REQUEST* del host 1 la dió el mismo controlador y por lo tanto el host 2 no conoce nada sobre el host 1.
5. El switch 2 recibe el *ARP REQUEST*. Al no existir entrada con la que coincida en la tabla de flujo envía el paquete al controlador en un *PACKET IN*. El controlador analiza el paquete y realiza el mismo procedimiento que con el paquete del host 1. Envía un *PACKET OUT* al switch 2 que contiene una trama *ARP REPLY* con la información que necesita el host 2. El switch 2 envía el *ARP REPLY* al host 2.
6. El host 2 envía un *ICMP REPLY* con destino el host 1. El switch 2 recibe este paquete y no necesita mandarlo al controlador porque su tabla de flujo ya fue configurada y por lo tanto la comunicación entre los dos usuarios finales ya queda establecida.

Con este intercambio de mensajes queda establecido el tráfico de datos entre los dos dispositivos finales y por la ruta más corta. Con esta implementación se elimina el tráfico ARP de la red y por lo tanto el control de tramas repetidas que se había configurado en la solución anterior. Esto es posible gracias a que el controlador tiene una caché en la que están relacionados direcciones Ethernet con direcciones IP. Para esta solución, esta tabla es directamente introducida por el desarrollador, ya que las direcciones de los equipos finales no son descubiertas al comienzo. Pero como se verá

en el capítulo de trabajos futuros, el descubrimiento de direcciones por parte de los dispositivos de red podría ser algo real.

La segunda diferencia importante respecto de la solución anterior es el método de gestión de eventos. Ahora es más complejo porque es necesario modificar el grafo, y para ello se requiere tener una información de la red. Cuando un enlace se modifica se generan dos eventos, uno en cada extremos del enlace. Los mensajes que recibe el controlador como consecuencia, tienen información sobre la razón que produjo el evento. Es entonces necesario gestionar esto de una manera correcta para no introducir errores en el grafo. En la [Figura 27] se puede observar un diagrama más completo de la nueva implementación.

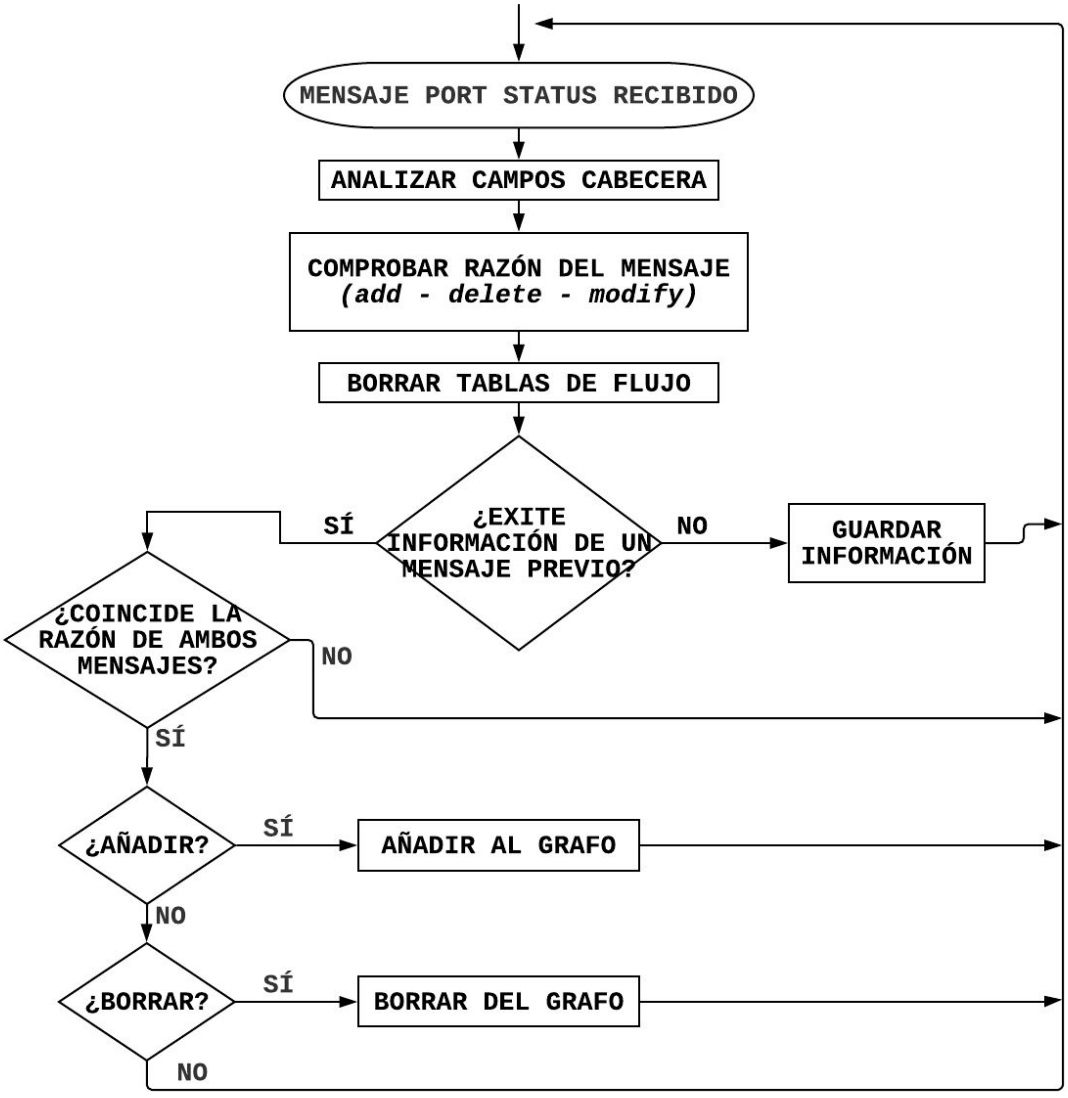


Figura 27 - *Diagrama de flujo de la gestión de eventos*

Con la nueva implementación se espera a tener la información de ambos extremos del enlace para modificar el grafo. Para borrar o añadir enlaces al grafo usamos las siguientes funciones:

*networkx.**remove_edge** (id_node, id_node)*

*networkx.**add_edge** (id_node, id_node, port)*

De esta manera el grafo queda actualizado después de cada cambio en la red y las rutas pueden recalcularse de una manera correcta.

3.6.3 PRUEBAS

Una vez se probó que la conexión mediante *Ping* funcionaba, se comenzaron a realizar pruebas de tráfico de datos con *Iperf* tanto sobre TCP como sobre UDP y los resultados obtenidos son los siguientes.

En la [Figura 28] se observa el resultado del tráfico TCP entre los dos dispositivos finales. El enlace entre los switch 1 y 2 se activa y desactiva cada 20 segundos aproximadamente. Al igual que en el apartado anterior, en cada modificación del enlace se puede apreciar una caída del ancho de banda que se recupera rápidamente.

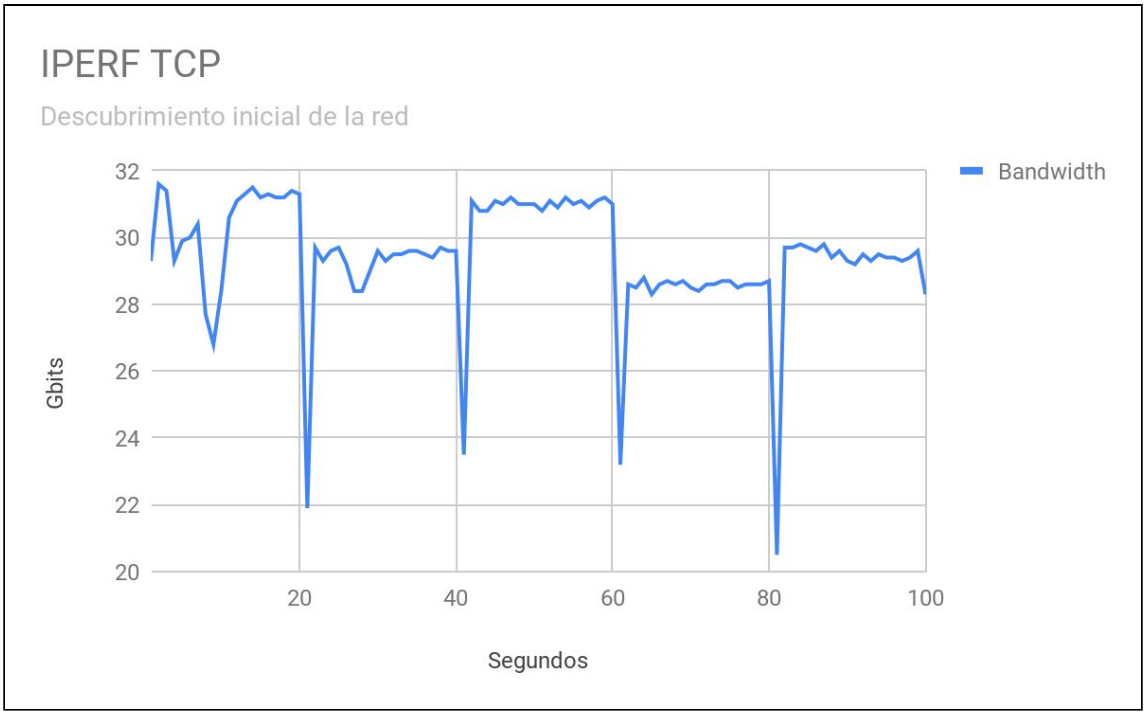


Figura 28 - *Iperf TCP descubrimiento inicial de la red*

Ahora es el momento de probar el tráfico sobre UDP. En el apartado anterior se pudo observar como para un ancho de banda de 100 Mbits/s la conexión se perdía durante varios segundos al realizar una modificación en el enlace. Por lo tanto, a continuación, en la [Figura 29] se puede observar el resultado de la ejecución de la misma prueba pero utilizando esta nueva configuración.

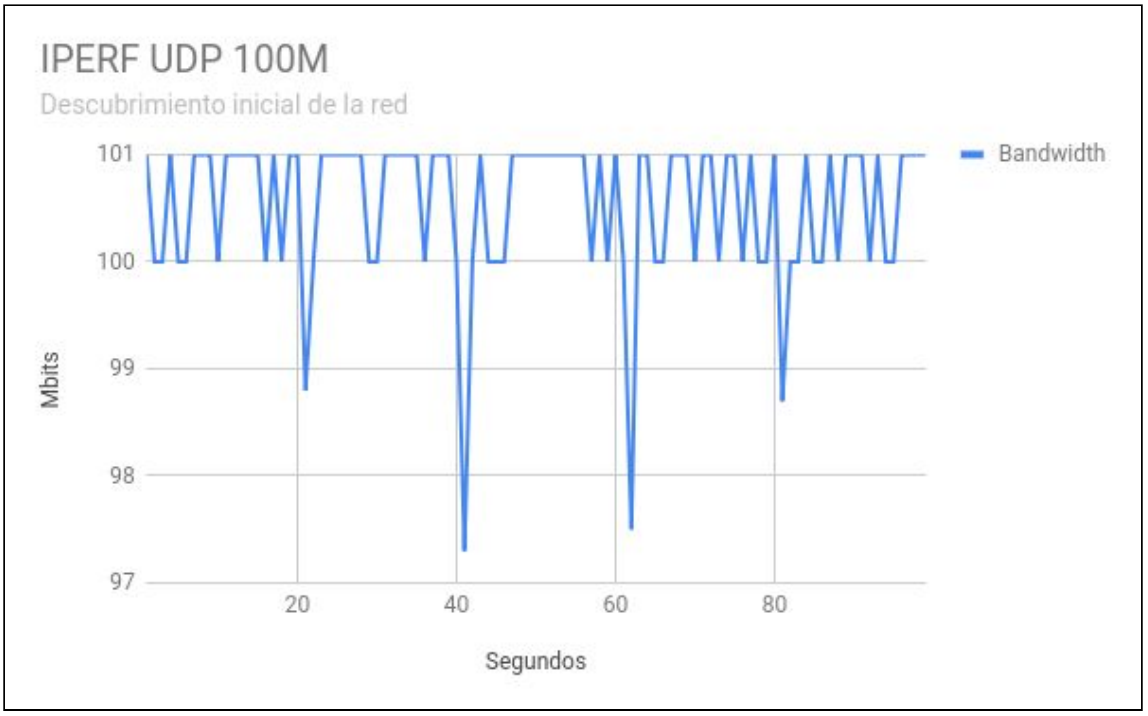


Figura 29 - *Iperf UDP descubrimiento inicial de la red*

Ahora se puede apreciar que aunque el enlace se modifique cada 20 segundos la recuperación es inmediata como en el caso de una conexión sobre TCP y que el ancho de banda se mantiene constante sobre los 100 Mbits por segundo especificados.

3.7 AMPLIACIÓN DEL ESCENARIO

Una vez se han obtenido unos resultados satisfactorios para el escenario de 3 switches y 2 host es momento de recordar que en la red puede haber muchos más nodos y enlaces. Por lo tanto en este apartado se va a probar el código en una topología aleatoria más extensa. Se van a ir eliminando enlaces para comprobar que la red se configura correctamente y que realmente se elige la ruta más corta. Es escenario utilizado en este apartado es el representado en la [Figura 30].

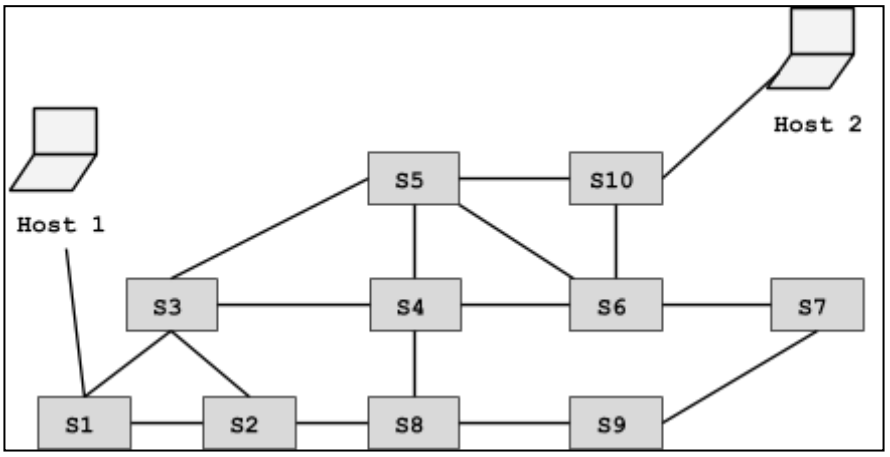


Figura 30 - *10 switches 2 hosts*

Según el escenario de la figura, la ruta más corta entre el host 1 y el host 2 es aquella que pasa por s1, s3, s5 y s10. No se puede ejecutar el comando *traceroute* para demostrar la ruta escogida puesto que los switches son dispositivos de nivel 2. Por lo tanto para demostrar que la ruta cambia según lo esperado vamos a sacar los datos del RTT de un comando *ping*.

En la [Figura 31] se puede observar unas variaciones del RTT durante una prueba de un minuto de duración. Los pasos que se siguieron durante la prueba fueron:

1. Todos los enlaces se encuentran activos y la ruta escogida es la más corta, es decir: s1, s3, s5 y s10.
2. En el segundo 20 cae el enlace entre los switches 5 y 10. La ruta más corta ahora es: s1, s3, s4, s6 y s10. Ahora existe un salto más en la ruta y se puede observar como el RTT ha aumentado.
3. En el segundo 40 cae el enlace entre los switches 4 y 6. La ruta más corta entonces es: s1, s3, s5, s6 y s10. En este caso se mantiene el número de saltos y no se observa ninguna variación significativa del RTT.
4. En el segundo 60 cae el enlace entre los switches 5 y 6. La ruta más corta ahora es: s1, s2, s8, s9, s7, s6 y s10. En este momento se añaden dos saltos y se puede observar que el RTT aumenta el doble de lo que aumentó la primera vez con sólo un salto.

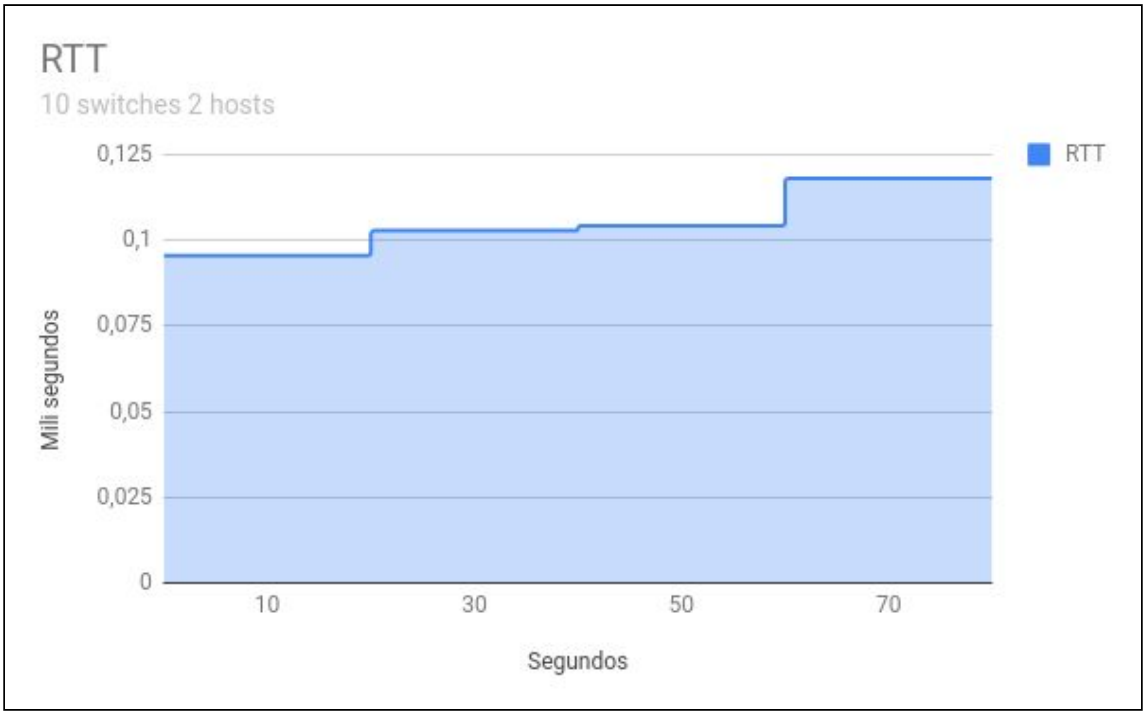


Figura 31 - *RTT 10 switches 2 hosts*

CAPÍTULO 4

ESCENARIO REAL

4. ESCENARIO REAL

Una vez se ha configurado el controlador de una manera correcta se comienza a configurar el escenario físico en el laboratorio. En este capítulo se explica el montaje del escenario en el laboratorio, la instalación y configuración de los equipos embarcados y los resultados obtenidos de las diferentes pruebas realizadas.

4.1 COMPONENTES DEL ESCENARIO

Empieza entonces el trabajo en el laboratorio. Para comenzar se expone el material del que se dispone y el uso que se va a dar a cada componente. Este escenario está compuesto por los siguientes componentes:

- 3 equipos SBC
- 3 PCs
- 1 Switch SMCFS8
- 8 dongles USB 3.0 a 10/100/1000 Gigabit Ethernet
- 9 cables Ethernet

A continuación, en la [Figura 32] puede observarse la distribución de los componentes.

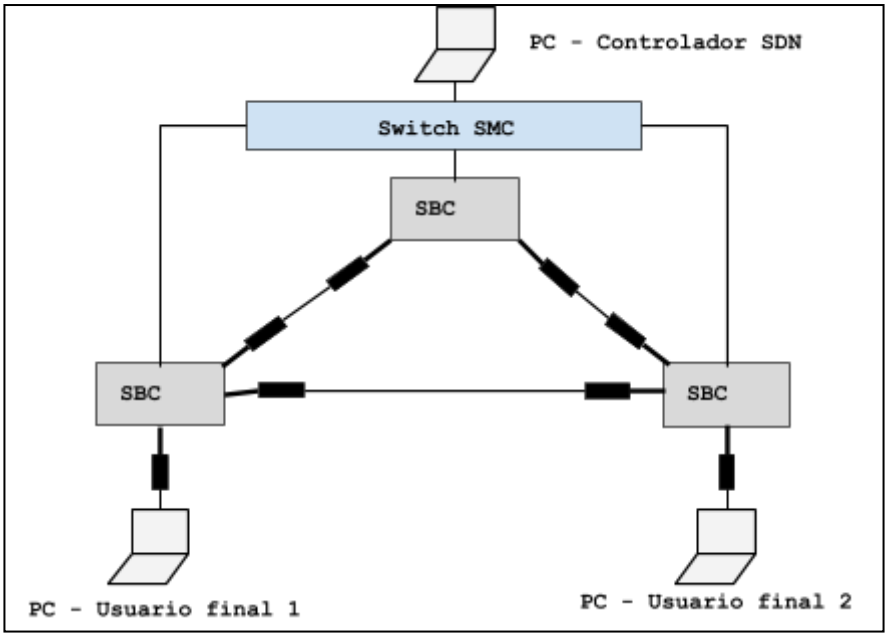


Figura 32 - *Montaje componentes escenario real*

4.2 OPEN VIRTUAL SWITCH

OVS (Open Virtual Switch) [16] será el software instalado en las Raspberry Pi para que éstas funcionen como switches OpenFlow y puedan comunicarse con el controlador SDN.

Open vSwitch es un software de código abierto bajo la licencia Apache 2.0. El objetivo de este software es que diferentes dispositivos sean capaces de funcionar como switches virtuales. Permite realizar un uso libre de la función de reenvío de una forma programable y está adaptado para funcionar con máquinas virtuales. Se trata también de una de las principales implementaciones que soportan el protocolo OpenFlow en todas sus versiones. Esta última característica es muy importante de cara a la red SDN que se quiere crear en este trabajo.

En la [Tabla 1] se pueden observar las características de los principales componentes de software de Open vSwitch.

Componente	Descripción
ovs-vswitchd	Es el componente principal. Implementa las funcionalidades del switch como comunicarse con el exterior mediante el protocolo OpenFlow, con los otros componentes y con el módulo del kernel.
ovsdb-server	Servidor de base de datos al cual ovs-vswitchd accede para obtener la configuración. Esta información se mantiene tras un reinicio del dispositivo. Los dos componentes se comunican mediante el protocolo OVSDB.
ovs-dpctl	Herramienta para la realizar la configuración del módulo del kernel del switch.
ovs-vsctl	Herramienta para realizar consultas y actualizaciones sobre la configuración de ovs-vswitchd.
ovs-appctl	Herramienta encargada de enviar los comandos para ejecutar los demonios de Open vSwitch.

Tabla 1 - *Componentes de Open Virtual Switch*

Para la configuración de los dispositivos se emplea este software y no otro que crea redes virtuales de nivel dos como *Linux Bridges* porque OVS sí soporta el protocolo OpenFlow.

4.3 CONFIGURACIÓN DE LOS EQUIPOS

Como ha sido explicado en capítulos anteriores, los equipos SBC embarcados son Raspberry Pi. Para la red SDN es necesario que estos dispositivos funcionen como switches OpenFlow y por lo tanto se debe instalar el software OVS.

En la Raspberry Pi se configura el sistema operativo Raspbian. Raspbian [17] es una distribución de Linux basada en Debian para las placas Raspberry.

Una vez se ha configurado el sistema operativo de la Raspberry se tendrá acceso a una interfaz gráfica similar a la de un PC común. Se inicia un terminal de comandos, y teniendo conexión a internet, se pueden utilizar los comando especificados en el [Anexo V] para realizar la instalación de OVS en la Raspberry.

Es entonces el momento de ejecutar los comandos necesarios para la configuración de OVS, pero para poder hacerlo correctamente antes se debe tener clara la arquitectura de una Raspberry y cómo qué función deben tener sus puertos. A continuación, en la [Figura 33], puede observarse la arquitectura de una Raspberry para este trabajo. En el puerto eth0, que conectará el dispositivo con la red de control SDN, necesita asignada una dirección IP perteneciente a dicha red. El resto de puertos serán incluidos dentro de un bridge OVS, esto es lo que les aportará la funcionalidad de switch.

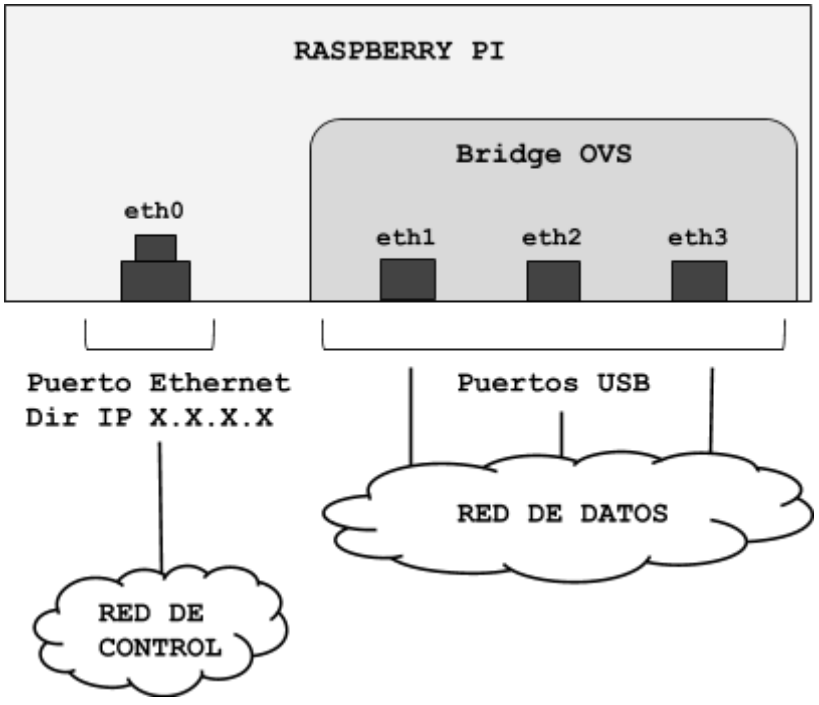


Figura 33 - *Distribución arquitectura Rpi*

Una vez ha quedado claro la distribución de los puertos se puede comenzar con la configuración el dispositivo. Lo primero es modificar el archivo en el que se configuran las interfaces que se encuentra ubicado en */etc/network/interfaces*. En el [Anexo VI] se encuentra un ejemplo de modificación del fichero para una Raspberry. Cada vez que el dispositivo se reinicie las interfaces quedan configuradas tal y como está establecido en el fichero. Se puede observar que la eth0 tiene asignada una dirección IP, que como hemos comentado es necesaria para la conexión del controlador. El resto de interfaces quedan configuradas de forma que se activen cuando reciban tráfico. No tienen dirección IP asignada puesto que trabajarán como puertos de un switch.

Seguidamente se debe configurar el bridge OVS. En el [Anexo VII] se encuentran comandos OVS necesarios para la configuración. Se necesita incluir las interfaces en el bridge, modificar la versión del protocolo OpenFlow que se desee para la conexión y conectar el bridge al controlador. Estos pasos se siguen en las tres Raspberry para que todas funcionen como switches OpenFlow.

4.4 MONTAJE DE LA RED

Es el momento entonces de montar la red física en el laboratorio. En la [Figura 34] queda reflejado un mapa de la configuración total de la red. Se especifican las interfaces y las direcciones IP en las interfaces que son necesarias.

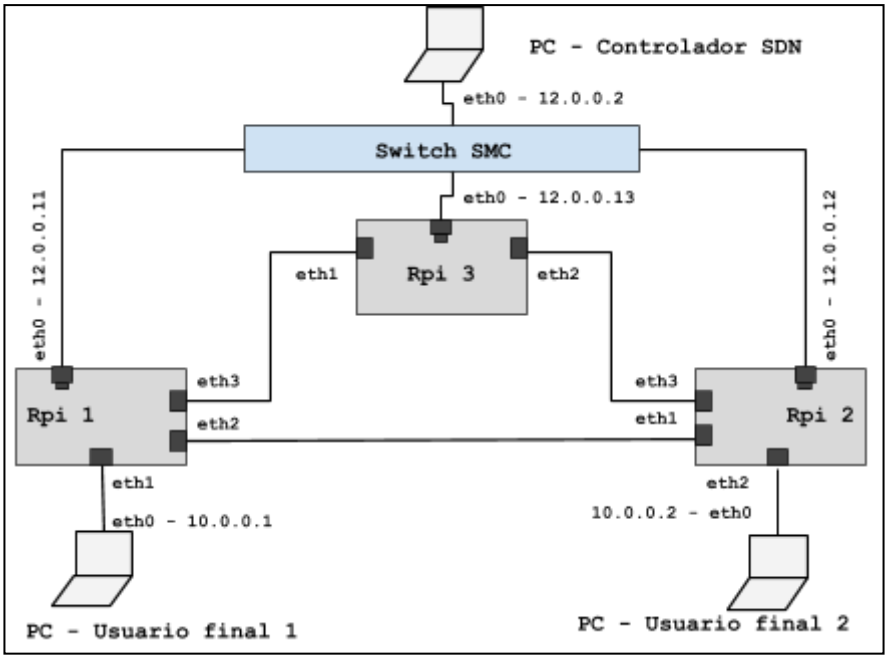


Figura 34 - *Configuración de interfaces y direcciones de la red*

Para comprobar la correcta configuración de la red se ejecuta un comando ping entre los usuarios finales. Durante esta conexión se prueba a desconectar y volver a conectar el enlace entre la Rpi 1 y la Rpi 2. La no existencia de cortes en la comunicación indica una correcta configuración y por lo tanto se puede comenzar las pruebas sobre el sistema.

4.5 PRUEBAS ESCENARIO CABLEADO

Ahora que el escenario se encuentra montado y configurado se comienzan a realizar distintas pruebas sobre él. Primeramente se analiza la comunicación establecida mediante el comando *ping*. De esta conexión es interesante analizar el RTT (*Round Trip Time)* de las tramas, es decir, el tiempo de ida y vuelta de un paquete. En la [Figura 35] se puede observar una gráfica de éste parámetro cuando el enlace entre la Rpi 1 y la Rpi 2 cambia de activo a inactivo y viceversa en varias ocasiones.

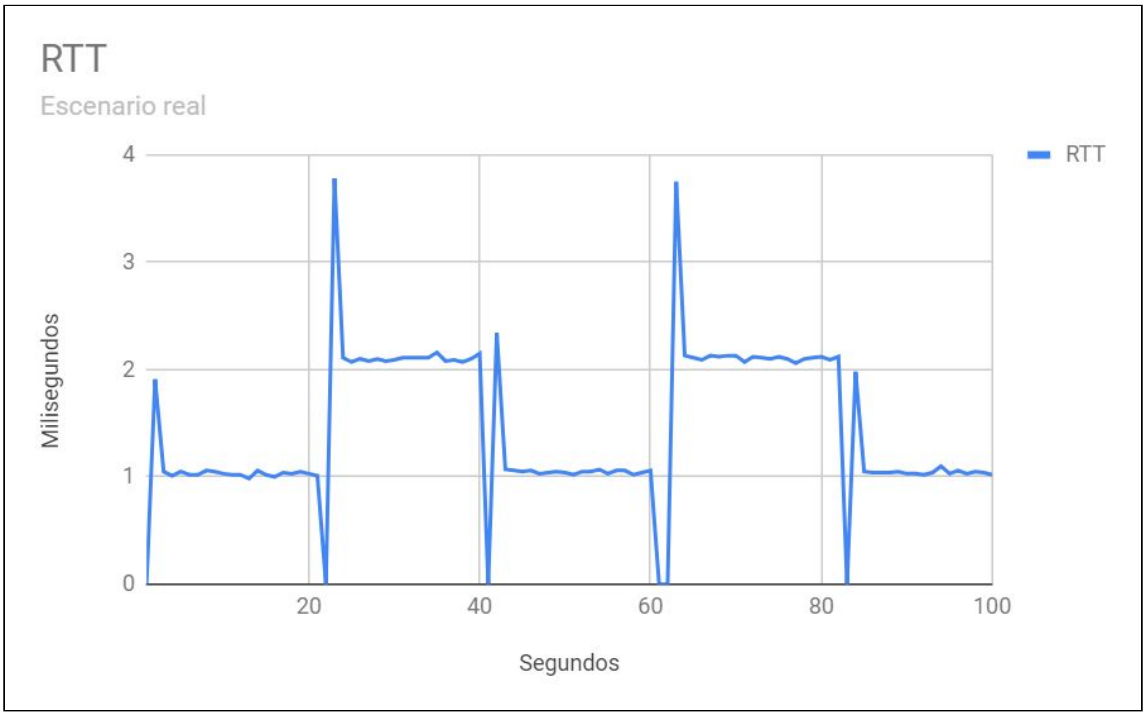


Figura 35 - *RTT escenario real con modificaciones del enlace*

La prueba consiste en modificar el enlace cada 20 segundos durante 100 segundos. Se puede observar a cada modificación un aumento puntual del RTT, debido a la reconfiguración de la red. También son fácilmente identificables los periodos en los cuales las tramas cursan el camino directo entre la Rpi 1 y la Rpi 2, o cuando el tráfico

es cursado a través de la Rpi 3. A través del camino corto el RTT tiene valores medios de 1 ms mientras que por el camino largo dobla su valor a los 2 ms.

Las siguientes pruebas que se realizan ya son simulando un tráfico de datos real con el comando *iperf*. Se procede a simular tráfico tanto sobre TCP como UDP. Primeramente en la [Figura 36] se observa el resultado de la simulación del tráfico sobre TCP.

En ella se observa la variación del ancho de banda de la comunicación cuando el enlace entre la RPi 1 y la RPi 2 está o no funcionando. Esta prueba se inicia con el enlace caído y puede observarse cómo en ancho de banda es algo menor que en los intervalos en los que el enlace sí está activo. Independientemente del estado del enlace se mantienen valores alrededor de los 90 Mbits/segundo.

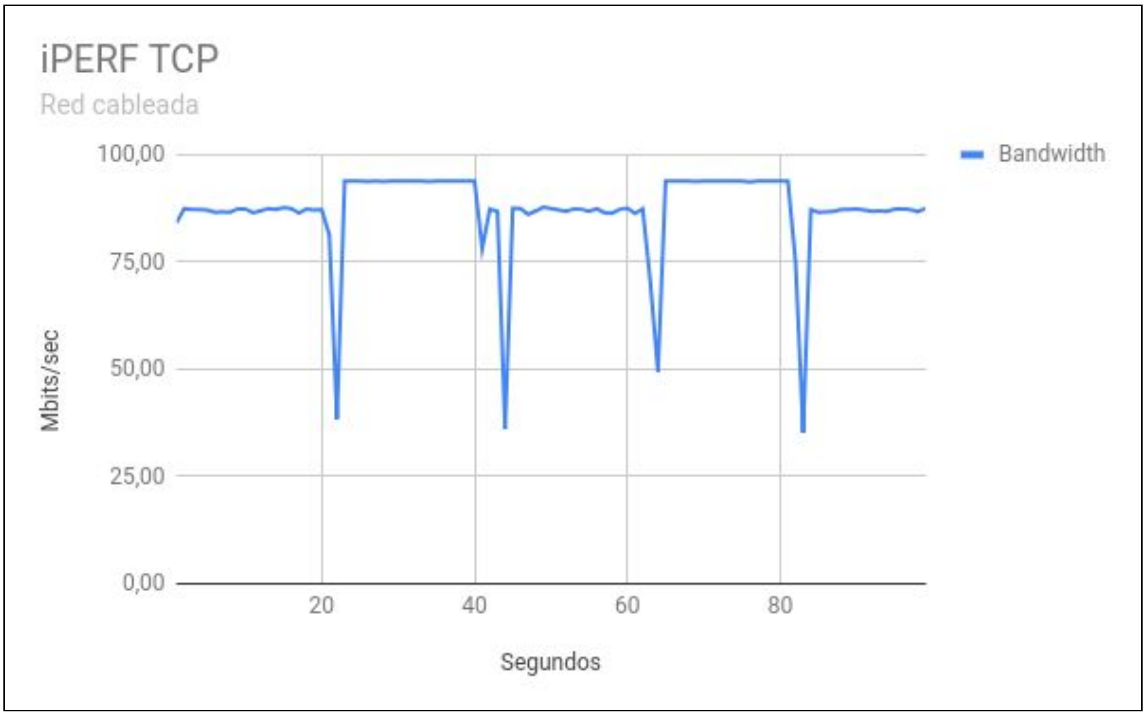


Figura 36 - *Iperf TCP escenario real cableado*

Seguidamente se realizó la misma prueba sobre UDP. En este caso se eligen de nuevo los anchos de banda de 50 y 100 Mbits/s, que se podrían comparar con los resultados obtenidos en el capítulo anterior en las pruebas realizadas simulando el escenario. En las [Figuras 37 y 38] a continuación se puede apreciar el comportamiento de la red para los diferentes anchos de banda.

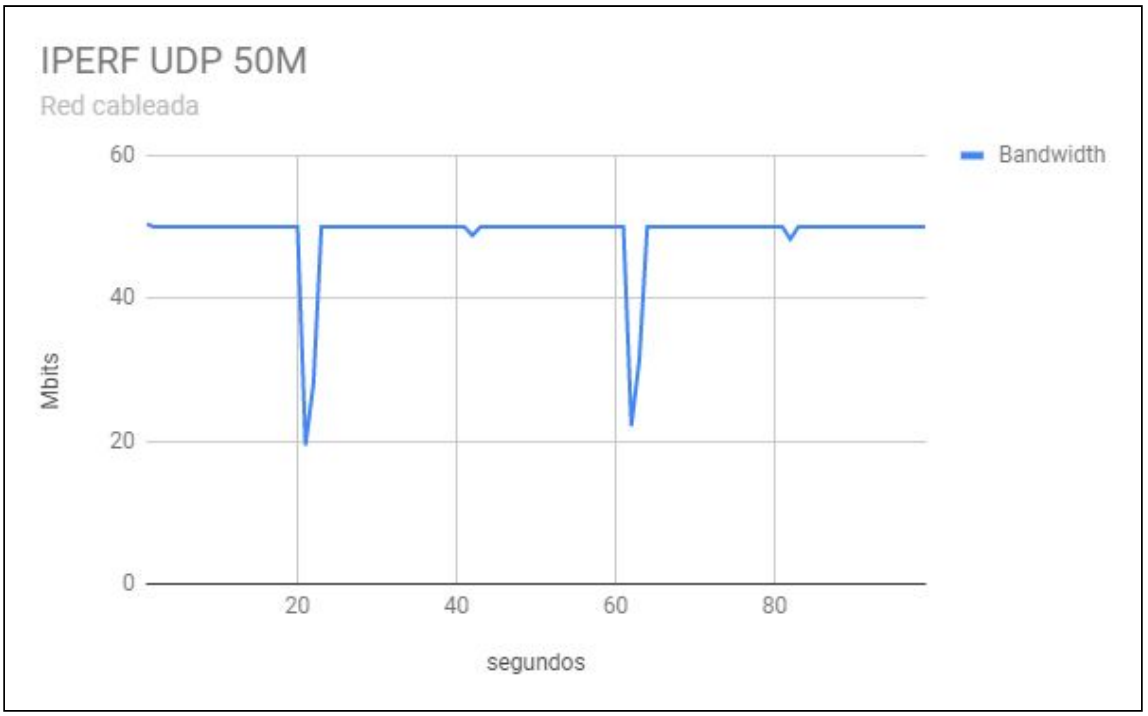


Figura 37 - *Iperf UDP 50M escenario real cableado*

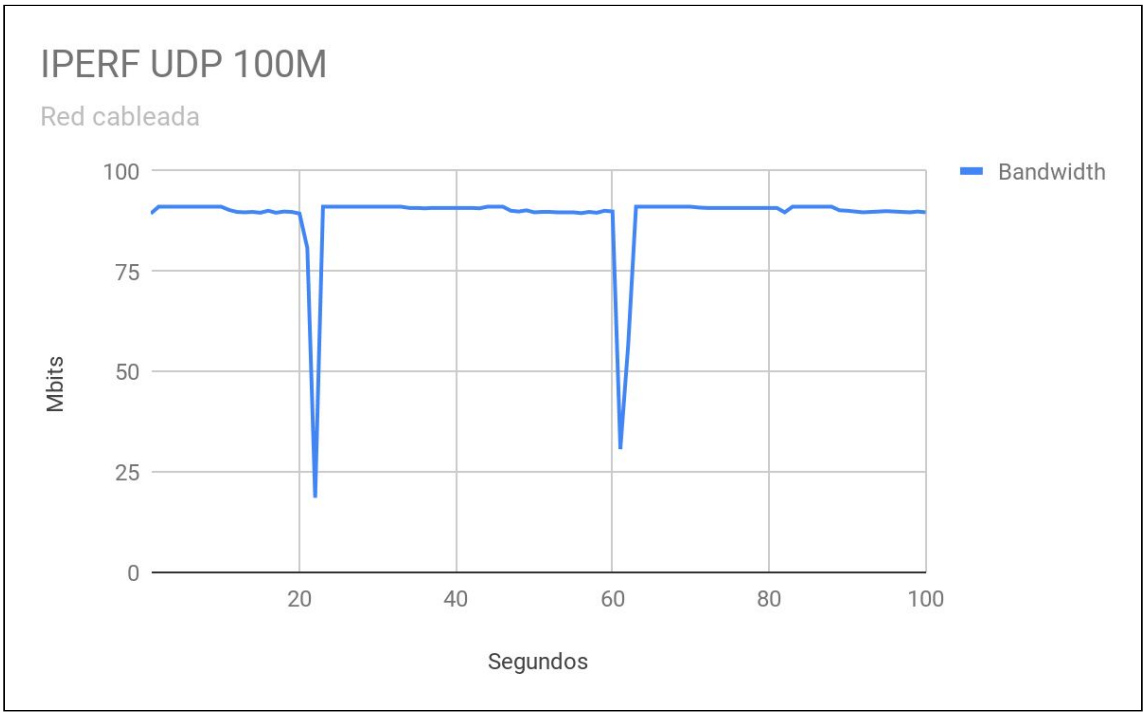


Figura 38 - *Iperf UDP 100M escenario real cableado*

Primeramente, para el ancho de banda de 50 Mbits/segundos, se modifica el enlace cada 20 segundos. Al comienzo de la prueba el enlace entre la Rpi 1 y la Rpi 2 se encuentra activo. En el gráfico pueden observarse los momentos de modificación del

enlace. Se puede observar también como la red no pierde la conexión en ningún momento, sino que el ancho de banda cae a valores cercanos a los 20 Mbits/segundo.

Seguidamente, para el ancho de banda de 100 Mbits/segundo, se modifica el enlace en los segundos 20 y 60. Las condiciones iniciales son las mismas que para el apartado anterior. Se puede observar como en los instantes de modificación el ancho de banda también cae a valores alrededor de los 20 Mbits/segundo.

Con estos resultados se concluye éste apartado de pruebas sobre la red cableada

4.6 INTERFAZ INALÁMBRICA

Una vez se ha probado la correcta conexión mediante cables, se va a proceder a una toma de contacto con las interfaz inalámbrica de las Raspberry. La motivación de la exploración de esta interfaz es obvia puesto que el trabajo está orientado a redes de UAVs.

Como se explicó cuando se detallaron las características de la Raspberry Pi 3, tienen integrada una antena para poder realizar conexiones inalámbricas. Para este trabajo se dispone de dos Raspberry Pi 3.

Primeramente hay que configurar la interfaz WiFi de los dispositivos. Para este trabajo resulta interesante que la comunicación entre los dispositivos sea en modo ad-hoc. La principal característica del modo ad-hoc es la ausencia de un punto de acceso en la red, es decir, la comunicación entre dos dispositivos es directa. Este hecho se considera necesario para el tipo de red de UAVs que se está tratando en este trabajo.

En el [Anexo VI] se puede observar dicha configuración la configuración realizada sobre la interfaz *wlan0* de las Raspberry Pi para crear una red ad-hoc inalámbrica. Para comprobar el correcto funcionamiento de la configuración se realizará una conexión entre ambas Raspberry asignado una dirección IP a ambas interfaces y se probará dicha conexión mediante el comando ping.

Una vez se comprobó que los dos dispositivos eran capaces de establecer una conexión inalámbrica se procede a configurar un escenario SDN. En la [Figura 39] se puede observar el escenario que se va a implementar para realizar la exploración de esta interfaz.

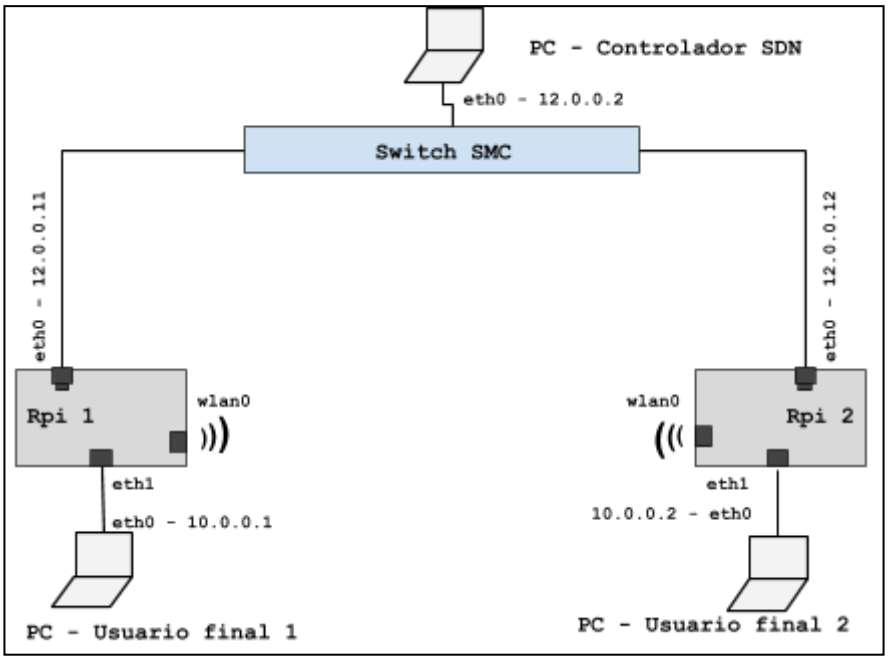


Figura 39 - *Configuración red inalámbrica laboratorio*

Para este escenario sólo tenemos dos dispositivos disponibles, y para la realización de algunas pruebas de análisis de prestaciones utilizaremos el script de *simple_switch_13.py,* cuyo funcionamiento fue ya explicado en apartados anteriores. No se emplea la solución final implementada por problemas de descubrimiento de enlaces inalámbricos.

La conexión será probada con el comando ping y en la [Figura 40] se puede observar una gráfica del RTT durante la ejecución de dicho comando. Se produce un intercambio de mensajes ICMP durante 100 segundos.

Se puede observar como el primer mensaje tiene un RTT de 100 ms, esto es porque como se ha explicado varias veces, la red debe configurarse. Después en media el RTT tiene un valor de 12 ms aproximadamente.

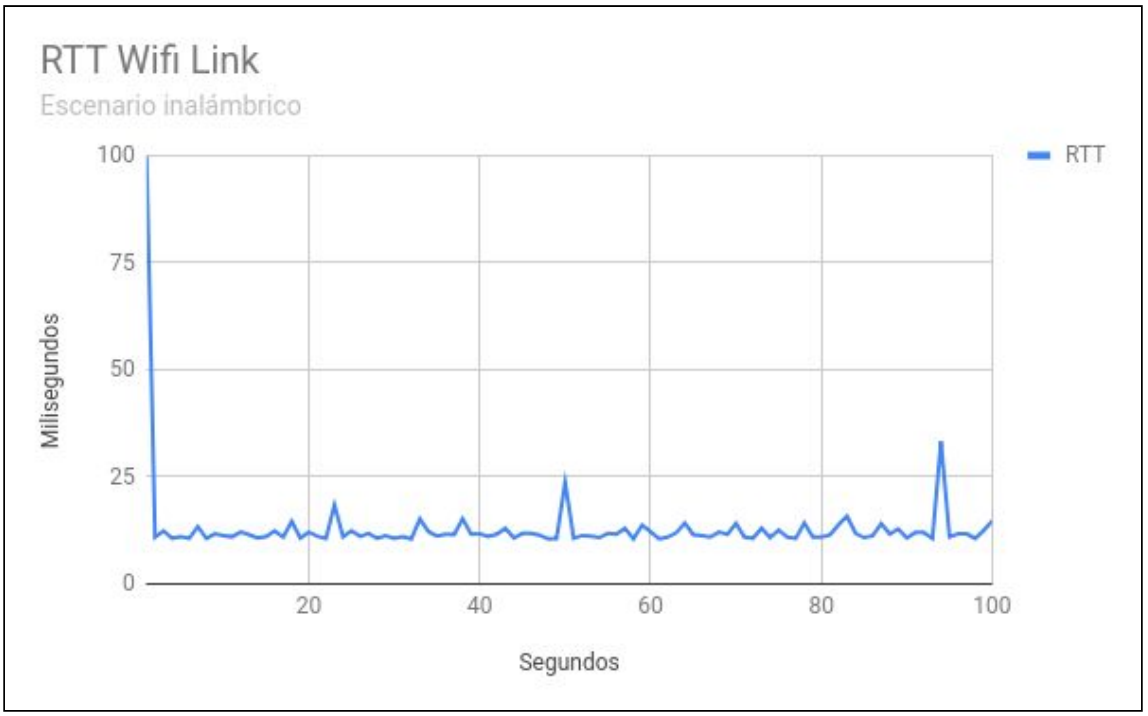


Figura 40 - *RTT escenario inalámbrico*

A continuación se procede a realizar pruebas de tráfico de datos en esta interfaz. Como en pruebas anteriores, se prueba el tráfico de datos sobre TCP con el comando iperf y los resultados obtenidos se pueden observar en la [Figura 41].

En dicha gráfica se observa un ancho de banda muy limitado alrededor de los 70 Kbits/s de media. Seguidamente en la [Figura 42], se realiza una prueba de tráfico sobre UDP con un ancho de banda de 100 Kbits/s y se observa el mismo comportamiento que en el caso de tráfico sobre TCP.

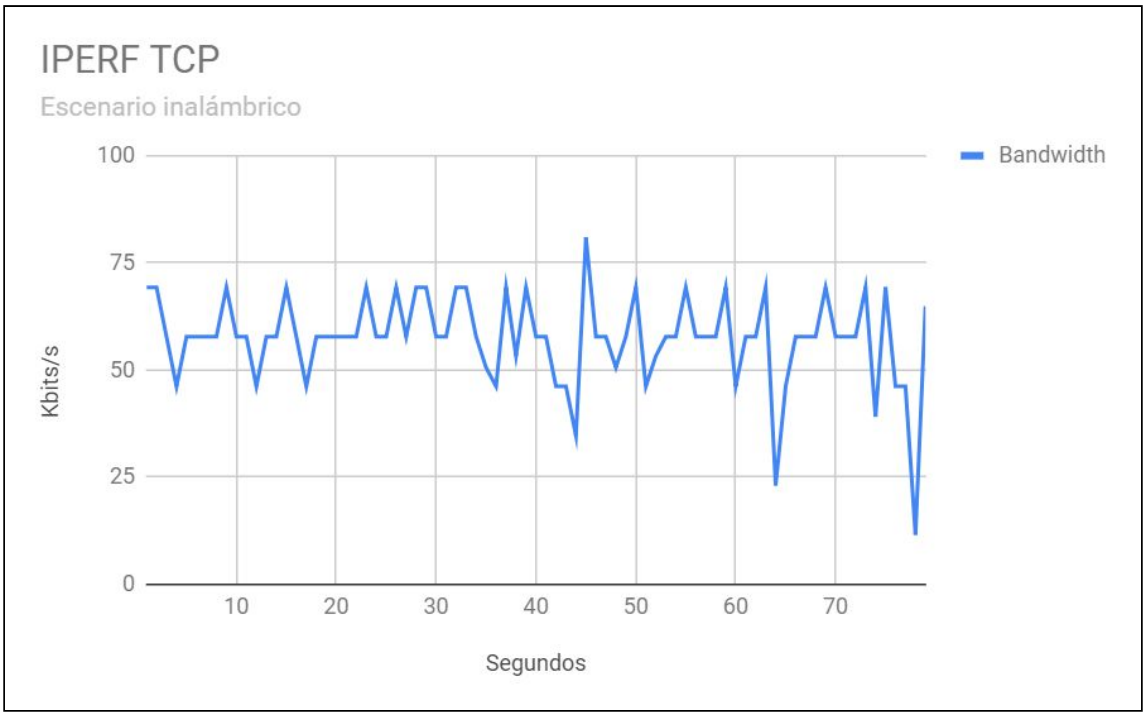


Figura 41 - *Iperf TCP interfaz inalámbrica*

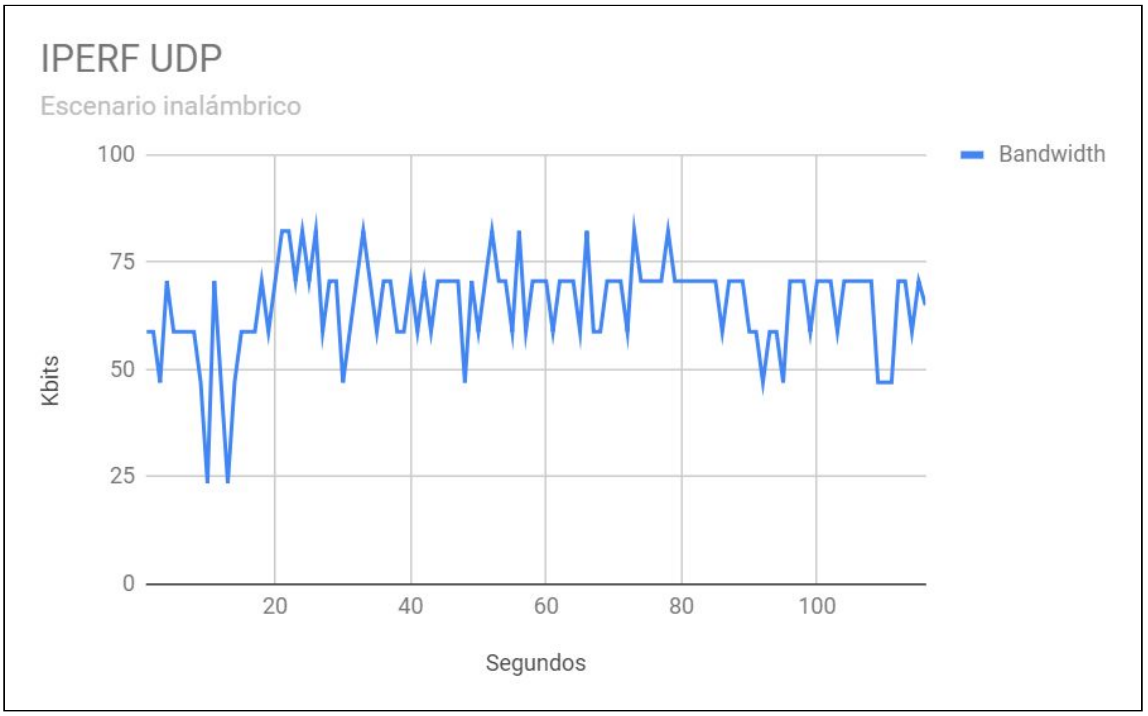


Figura 42 - *Iperf UDP interfaz inalámbrica*

Con estos resultados finaliza la fase de exploración de la interfaz inalámbrica.

CAPÍTULO 5

CONCLUSIONES Y TRABAJOS FUTUROS

5. CONCLUSIONES Y TRABAJOS FUTUROS

En este capítulo se exponen las conclusiones del trabajo realizado y una propuesta de los futuros desarrollos que podrían ser implementados para la continuación del mismo.

5.1 CONCLUSIONES

Como se ha podido observar, en este Trabajo de Fin de Grado se han desarrollado dos principales tareas diferenciadas: el desarrollo y configuración del controlador SDN y la experimentación y estudio de la tecnología SDN con las Raspberry Pi. El trabajo realizado sobre el controlador es totalmente necesario para poder realizar pruebas con las Raspberry Pi. Sin embargo, previamente al uso del controlador sobre estos dispositivos hubo que realizar un estudio y configuración de los mismos.

Antes de comenzar con el desarrollo de estas tareas se realizó un trabajo fundamental. Este trabajo es el estudio de las redes de UAVs. Resulta necesario analizar las características y necesidades de este tipo de redes y por qué la tecnología SDN puede ser una buena solución para las problemáticas que presentan. Así se pudo realizar posteriormente unas correctas configuraciones e implementaciones.

Durante todo el proyecto se ha demostrado la importancia de una correcta configuración del controlador, pues es el cerebro de la red y por lo tanto es necesario que se ajuste totalmente a las necesidades de la misma. Primeramente se ha realizado un trabajo de comprensión del mismo con diferentes topologías gracias a la herramienta Mininet. Después se identificaron problemas y se establecieron soluciones para poder lograr la configuración deseada. Finalmente ha sido llevado a cabo un trabajo en el laboratorio, para realizar una prueba de concepto sobre un dispositivo SBC. Estas pruebas realizadas utilizando Raspberry Pi, se han realizado sobre escenarios cableados y se ha terminado haciendo un primer uso de la interfaz inalámbrica.

Aunque se pueda concluir que se han cumplido los objetivos de configuración de una red SDN sobre los equipos Raspberry Pi, es una configuración que tiene varias limitaciones, tanto en el desarrollo del código del controlador como en la interfaz inalámbrica de las Raspberry Pi. Las actividades futuras que pueden desarrollarse para intentar solucionar estas limitaciones se detallan en el siguiente punto.

5.2 FUTUROS TRABAJOS

En el apartado anterior se ha comentado que a la finalización de este proyecto se han encontrado diferentes limitaciones en el mismo. Por lo tanto, en este punto se proponen diferentes tareas que podrían solucionar los problemas detectados. Los trabajos propuestos son los siguientes:

- Desarrollo de un cambio en el controlador para la detección de enlaces ad-hoc en una interfaz inalámbrica. La actual implementación utiliza el protocolo LLDP para descubrimiento de vecinos, pero este protocolo es útil en redes cableadas. En la parte de exploración de la red inalámbrica se comprobó este hecho y por lo tanto se propone el cambio de protocolo de descubrimiento de vecinos a uno que sí funcione en redes ad-hoc.
- Continuación de los trabajos sobre la interfaz inalámbrica. Como se ha podido ver los resultados obtenidos sobre el plano de datos deben ser mejorables. Actualmente se presenta un ancho de banda muy bajo. El objetivo de este proyecto es conseguir una red SDN funcional sobre una red de UAVs y por lo tanto un correcto funcionamiento de la interfaz inalámbrica resulta fundamental.

Este sería un resumen de los principales trabajos que se deberían realizar en un futuro inminente y que son totalmente necesarios para conseguir una red de drones funcional utilizando la tecnología SDN.

No terminar sin recordar que estos trabajos están totalmente enmarcados en una primera exploración de la tecnología SDN para configurar el reenvío de datos en dispositivos SBC. Pero otros trabajos como el control de vuelo de los drones o la implantación de los dispositivos SBC en los equipos voladores son fundamentales para que en un futuro pueda observarse en el aire el trabajo realizado.

ANEXOS

ANEXO I - PLANIFICACIÓN

En este punto se desarrolla una organización y planificación del proyecto. Se detallan las distintas fases en las que el trabajo ha sido dividido, el desarrollo en el tiempo de dichas fases y su presupuesto de realización.

Se han definido varias fases para la realización del proyecto:

- ANÁLISIS Y OBJETIVOS: Primeramente, se deben plantear cuáles van a ser las partes fundamentales del proyecto y establecer los objetivos del mismo. Esta fase tiene una gran importancia y efecto en el desarrollo general del proyecto. Un análisis incorrecto de las necesidades y objetivos puede suponer una pérdida de tiempo y dinero.

- ESTUDIO FUNDAMENTOS TEÓRICOS: Se debe realizar un entendimiento de las bases teóricas de las ideas principales del proyecto. Se realiza un correcto estudio de los conceptos para poder realizar un buen desarrollo posteriormente.

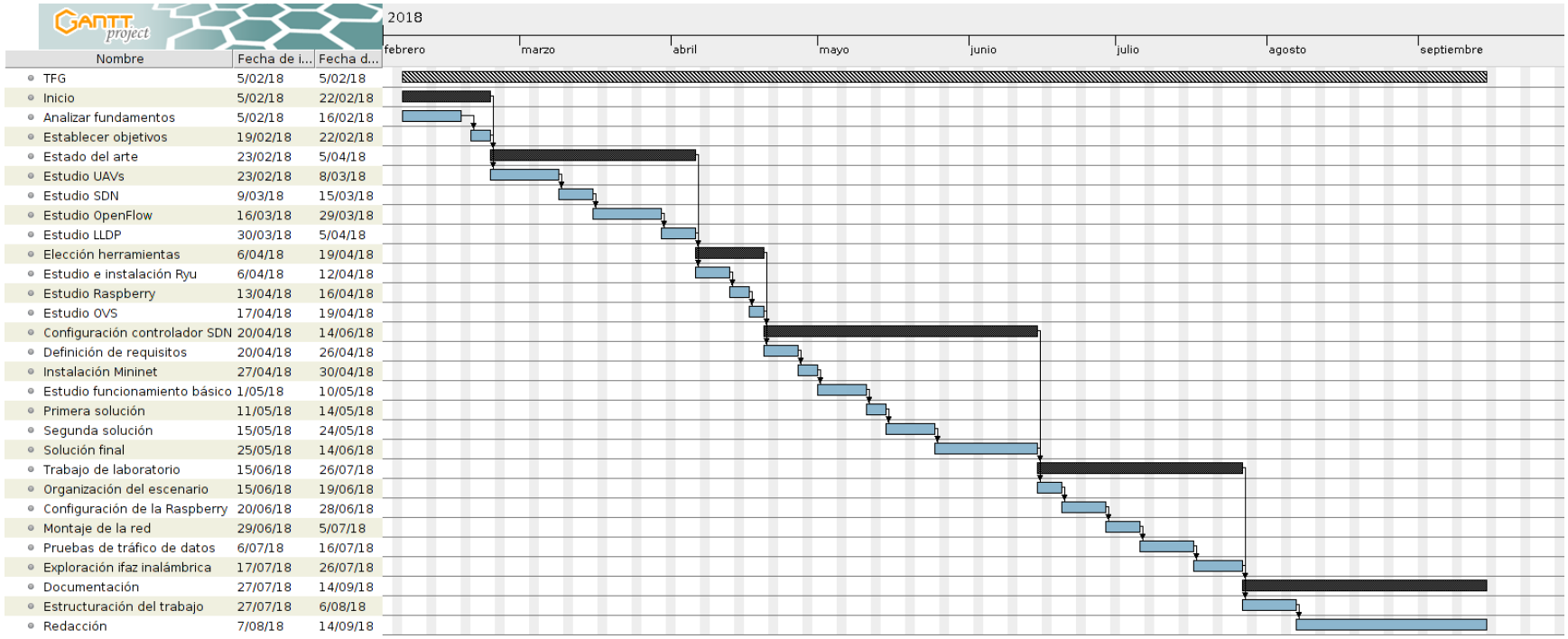
- ELECCIÓN DE LAS HERRAMIENTAS Y DISPOSITIVOS: Una vez se realizó un correcto entendimiento de las ideas y teniendo los objetivos presentes se lleva a cabo la elección del material necesario para el desarrollo.

- DISEÑO Y SIMULACIÓN: Se comienza el desarrollo del proyecto y antes de implementarlo en un escenario real se realizan pruebas en simulación para comprobar un funcionamiento adecuado. Esto se lleva a cabo de esta manera para evitar posibles errores de una implementación real durante el desarrollo del código.

- IMPLEMENTACIÓN REAL Y PRUEBAS: En esta fase el trabajo se realiza en laboratorio y se procede a la implementación de un escenario físico. Se realizan diferentes pruebas sobre dicho escenario y los resultados de las mismas son analizados.

- EVALUACIÓN DEL TRABAJO: Tras el fin de las pruebas en el escenario real y con un análisis de las mismas se procede entonces a la realización de un análisis general de todo el trabajo realizado. También se completa un estudio de los trabajos futuros de mejora y ampliación.

En la [Figura 43] se puede observar un diagrama de Gantt donde queda detallada la distribución en el tiempo de todas las fases.



Nombre	Fecha de i...	Fecha d...
TFG	5/02/18	5/02/18
Inicio	5/02/18	22/02/18
Analizar fundamentos	5/02/18	16/02/18
Establecer objetivos	19/02/18	22/02/18
Estado del arte	23/02/18	5/04/18
Estudio UAVs	23/02/18	8/03/18
Estudio SDN	9/03/18	15/03/18
Estudio OpenFlow	16/03/18	29/03/18
Estudio LLDP	30/03/18	5/04/18
Elección herramientas	6/04/18	19/04/18
Estudio e instalación Ryu	6/04/18	12/04/18
Estudio Raspberry	13/04/18	16/04/18
Estudio OVS	17/04/18	19/04/18
Configuración controlador SDN	20/04/18	14/06/18
Definición de requisitos	20/04/18	26/04/18
Instalación Mininet	27/04/18	30/04/18
Estudio funcionamiento básico	1/05/18	10/05/18
Primera solución	11/05/18	14/05/18
Segunda solución	15/05/18	24/05/18
Solución final	25/05/18	14/06/18
Trabajo de laboratorio	15/06/18	26/07/18
Organización del escenario	15/06/18	19/06/18
Configuración de la Raspberry	20/06/18	28/06/18
Montaje de la red	29/06/18	5/07/18
Pruebas de tráfico de datos	6/07/18	16/07/18
Exploración ifaz inalámbrica	17/07/18	26/07/18
Documentación	27/07/18	14/09/18
Estructuración del trabajo	27/07/18	6/08/18
Redacción	7/08/18	14/09/18

Figura 43 – *Diagrama de Gantt*

ANEXO II - ENTORNO SOCIO-ECONÓMICO

En este punto se expone un presupuesto desarrollado para la realización de este proyecto. En dicho presupuesto se incluye un desglose de todos los gastos necesarios, tanto humanos como materiales.

GASTOS MATERIALES

En este apartado se detalla el material necesario para la realización del proyecto y los costes asociados a dicho material. Los gastos materiales están compuestos en su totalidad por equipos físicos puesto que todo el software usado es de código libre y por lo tanto no genera coste alguno. Los materiales necesarios son los siguientes:

- 2 PCs Clevisa winec mod. Intel I3-8100 a 3.6 GHz
- 1 PC Asus F552L Intel Core I7-4510U a 3.1 GHz
- 1 switch SMC FS8
- 2 Raspberry Pi 3 model B
- 1 Raspberry Pi 2 model B
- 8 dongles adaptadores de USB 3.0 a 10/100/1000 Gigabit Ethernet.
- 9 cables Ethernet
- 3 pantallas de 15’’ con cable HDMI.
- 3 packs ratón + teclado.

Recurso	**Unidades**	**Coste unidad (€)**	**Coste total (€)**
PC Clevisa Winec Mod.	2	400	800
PC Portátil Asus	1	750	750
Switch SMC FS8	1	13,47	13,47
Raspberry Pi 3 model B	2	28,26	76,52
Raspberry Pi 2 model B	1	36,00	36,00
Dongles	8	13,99	111,92
Cables	9	5,86	52,74
Pantalla + HDMI	3	125	375

Ratón y teclado	3	14,44	43,32
		TOTAL	**2.258,97**

Tabla 2 – *Costes dispositivos hardware*

El coste asociado al hardware del trabajo es entonces de dos mil doscientos cincuenta y ocho euros con noventa y siete céntimos. Este es, a su vez, el coste total asociado a los gatos materiales.

GASTOS DE PERSONAL

En este apartado de detalla los costes asociados al trabajo realizado por las personas. Para este trabajo es necesario el trabajo del estudiante que lo realiza y del tutor responsable del mismo.

Persona	**Horas proyecto**	**Coste hora(€)**	**Coste total (€)**
Estudiante	676	13,55	9.159,80
Tutor	120	19,66	2.359,20
		TOTAL	**11.519**

Tabla 3 – *Coste de personal*

Las horas son calculadas en base a que como se pudo ver en el [Anexo 1] el proyecto tiene una duración de siete meses y medio. Se estima un trabajo de 4 horas diarias por parte del alumno los días laborables. Por parte del profesor se estima una dedicación de cuatro horas semanales. El coste asociado entonces al gasto de personal durante siete meses y medio de trabajo es de once mil quinientos diecinueve euros.

RESUMEN

En este apartado se detalla el total de los gastos y por lo tanto el presupuesto final del proyecto.

Resumen	Coste total (€)
Material	2.258,97
Personal	11.519
TOTAL	**13777,97**

Tabla 4 - *Costes totales del proyecto*

Se concluye entonces que el proyecto tiene un coste total asociado de trece mil setecientos setenta y siete euros con noventa y siete céntimos.

ANEXO III - MARCO REGULADOR

El 29 de diciembre de 2017 fue publicado en el BOE el nuevo marco de regulación para el uso civil de vehículos aéreos no tripulados en el territorio nacional [18]. Introduce cambios respecto a la anterior ley de julio de 2014 [19]. Se especifican puntos importantes como qué se necesita para pilotar drones de una manera profesional o de una manera recreativa y qué características deben tener los dispositivos para cada tipo de vuelo.

En el decreto se imponen numerosas normas respecto a las características del dispositivo, la regulación de los vuelos y la ocupación del espacio aéreo, la formación de los pilotos y las licencias necesarias. Sin embargo, no se encuentra una limitación legal para el alcance de este trabajo puesto que está enfocado a la configuración de unos equipos embarcados en la propia infraestructura del dispositivo y no existe especificación alguna sobre este punto.

ANEXO IV - INSTALACIÓN Y EJECUCIÓN MININET

En este anexo se detalla el código usado en Mininet para las simulaciones. El primero configura 3 switches y 2 hosts. El segundo configura una red más grande con 10 switches y 2 hosts. Al final también se indica el comando necesario para poder ejecutar los códigos.

```
"""
```

```
3 switches 2 hosts
"""

from mininet.topo import Topo

class MyTopo( Topo ):

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

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        leftHost = self.addHost( 'h1' )
        rightHost = self.addHost( 'h2' )
        leftSwitch = self.addSwitch( 's1' )
        rightSwitch = self.addSwitch( 's2' )
        centralSwitch = self.addSwitch( 's3' )

        # Add links
        self.addLink( leftHost, leftSwitch )
        self.addLink( leftSwitch, rightSwitch )
        self.addLink( rightSwitch, rightHost )
        self.addLink( centralSwitch, leftSwitch )
        self.addLink( centralSwitch, rightSwitch )


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

```
"""
10 switches 2 hosts
"""

from mininet.topo import Topo

class MyTopo( Topo ):

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

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        host1 = self.addHost( 'h1' )
        host2 = self.addHost( 'h2' )
        switch1 = self.addSwitch( 's1' )
        switch2 = self.addSwitch( 's2' )
```

```
        switch3 = self.addSwitch( 's3' )
        switch4 = self.addSwitch( 's4' )
        switch5 = self.addSwitch( 's5' )
        switch6 = self.addSwitch( 's6' )
        switch7 = self.addSwitch( 's7' )
        switch8 = self.addSwitch( 's8' )
        switch9 = self.addSwitch( 's9' )
        switch10 = self.addSwitch( 's10' )


        # Add links
        self.addLink(switch1, host1)
        self.addLink(switch10, host2)
        self.addLink(switch1, switch2)
        self.addLink(switch1, switch3)
        self.addLink(switch2, switch3)
        self.addLink(switch3, switch5)
        self.addLink(switch3, switch4)
        self.addLink(switch2, switch8)
        self.addLink(switch5, switch4)
        self.addLink(switch4, switch8)
        self.addLink(switch5, switch10)
        self.addLink(switch5, switch6)
        self.addLink(switch4, switch6)
        self.addLink(switch8, switch9)
        self.addLink(switch6, switch7)
        self.addLink(switch9, switch7)
        self.addLink(switch10, switch6)



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

```
 sudo mn --custom file_name.py --topo mytopo --switch ovsk
--mac --controller=remote,ip=controler_ip -x
```

ANEXO V - INSTALACIÓN OPEN VIRTUAL SWITCH EN RPI

En este anexo se indican los comandos utilizados para la instalación de Open Virtual Switch en las Raspberry Pi.

```
$ sudo apt install openvswitch-switch
$ sudo apt install openvswitch-common
$ sudo apt install bridge-utils
```

ANEXO VI - FICHERO CONFIGURACIÓN DE RED DE RPI

En este anexo se indica un ejemplo de configuración de las interfaces de una Raspberry. Este código se encuentra escrito en el fichero */etc/network/interfaces*. El fichero es modificable las veces que se necesite, es decir, se pueden configurar las interfaces que se vayan a utilizar.

```
# Loopback network interface
auto lo
iface lo inet loopback

allow-hotplug eth0
iface eth0 inet static
      address 12.0.0.10
      netmask 255.255.255.0
      network 12.0.0.0
      broadcast 12.0.0.255

auto eth1
iface eth1 inet manual

auto eth2
iface eth2 inet manual

auto eth3
iface eth3 inet manual

auto bridge_rpi
allow-ovs bridge_rpi
iface bridge_rpi inet manual
ovs_type OVSBridge

auto wlan0
iface wlan0 inet manual
```

ANEXO VII - COMANDOS OPEN VIRTUAL SWITCH

En este anexo se indican los comandos de Open Virtual Switch necesarios para la configuración de las Raspberry Pi.

Configuración del Bridge

```
sudo ovs-vsctl add-br bridge_rpi
sudo ifconfig bridge_rpi up
sudo ovs-vsctl add-port bridge_rpi eth1
sudo ovs-vsctl add-port bridge_rpi eth2
sudo ovs-vsctl set Bridge bridge_rpi protocols=OpenFlow13
sudo ovs-vsctl set-controller bridge_rpi
tcp:IP_CONTROLLER:6633
```

Visualización de las tablas de flujo

```
sudo ovs-ofctl -O OpenFlow13 dump-flows br1
```

ANEXO VIII - *EXTENDED ABSTRACT*

This chapter explains the motivation, objectives and conclusions of this End-of-Grade Work.

MOTIVATION

UAVs (Unmanned Aerial Vehicles) began their history during the 19th century for military purposes. The first ones that were built were very expensive and really large size vehicles. The continuous development of electronics and the miniaturization of components has allowed these devices to evolve into what are commonly known as drones. The SUAVs (Small Unmanned Aerial Vehicles) or MAVs (Micro Air Vehicles) have been part of our daily lives for several years and it seems that they are here to stay for a long time.

The MAVs are much smaller than other air vehicles or other UAVs such a Predator, Siva or Milano. Thanks to this feature, MAVs provide accessibility in hostile locations, have a lower price, reduce human risk and can be controlled very precisely. They also reduce the time needed to carry out different operations because they do not need previous pre-flight tasks and can take off or land almost anywhere.

As can be seen, the advantages offered by unmanned aerial vehicles are considerable and, as a result, various missions are currently being carried out in both the military and civil fields. They are able to make many different operations. They perform rescue and risk control operations, transmit multimedia traffic and can carry out the transport of small goods.

Another major advantage of these small devices is the ability to carry out operations using several UAVs at the same time. The first missions completed with this devices use only one of them, but the research developed on drone networks is becoming more and more extensive. The use of a greater number of devices means a lower probability of failure of a mission and a saving of time in it. At the same time, they are less expensive, as several cheaper devices can be used instead of one that is more expensive.

FANETs (Flying Ad Hoc Networks) are grouped within VANETs (Vehicle Ad Hoc Networks) since drones are considered vehicles. The VANETs belong to the large group of MANETs (Mobile Ad Hoc Networks). Both MANETs and VANETs have been highly developed in recent times and for their similarities try to use many of the protocols created for them in the FANETs. Even so, there are many differences that exist and therefore the need arises to look for alternatives for the control of these

networks. Therefore, it has been thought that one of the options that can be used to perform an effective management is the use of SDN.

The SDN (Software Defined Networking) technology has a main characteristic. This characteristic is that there is a separation between the data layer and de control layer. This separation leads to the appearance of a new figure in the network: the SDN controller, which is, as its name suggests, will be in charge of the control layer. From this device it will be possible to control the configuration of the rest of the devices of the network and manage it in a centralized way.

In the union of these two concepts is where the motivation of this End-of-Grade Work resides mainly. The SDN technology can be a good solution to manage data networks between drones with a great efficiency, because the controller can have a real-time image of the network and make a quick recognition of it. The use of this technology in these networks is a challenge of great complexity and therefore it is necessary to make a first exploration.

OBJECTIVES

The main objective of this End-of-grade Work is to perform a first scan of SDN technology to configure information forwarding. This explanation will be carried out first in a simulation environment and later in the laboratory with physical devices.

This overall objective can be divided into the following points:

- Study de MAVs networks. Understand their main characteristics and needs.
- Study the SDN technology. Understand its main characteristics and advantages that it can bring. Understand the architecture of SDN networks and their main components.
- Study of different SDN controllers taking into account their main characteristics and choice of an SDN controller for the network.
- Study of the functioning of the OpenFlow protocol, fundamental in SDN networks. Study of the operation and characteristics of OpenFlow switches. Perform an analysis of the most important messages exchanged between the switches and the controller that will be fundamental in the development of this work.
- Carrying out tests on the chosen controller simulating different network topologies witch the Mininet tool.

- Identification of different problems in the configuration of the controller and creation of solutions for these problems.
- Study the characteristics of Raspberry Pi devices and configure them to function as OpenFlow switches.
- Testing in a fixed and wired scenario with the previous configuration of the controller and Raspberry Pi devices.
- Exploration of the wireless interface of the devices.

JOB PROGRAMME

Once the motivation and objectives of the work have become clear, the development of the project begins. First the state art state is analyzed and explained, then an understanding and configuration of the controller is made and finally an exploration in the laboratory is done with physical devices.

ART STATE

In this chapter a study of different theoretical concepts necessary for the development of the project is carried out. The objectives of study of MAV networks, SDN technology and OpenFlow protocol are fulfilled in this chapter. But other necessary tools such as the LLDP protocol or Raspberry Pi devices are also discussed.

USE OF SDN TECHNOLOGY IN MULTI-HOP NETWORKS

This is the most extensive chapter of the work. It begins with a first exploration of the chosen controller that helps us to identify the problems. Once the problems for our type of multi-hop SDN have been identified, we proceed to detail the solutions that are being proposed. At the same time it is explained which are the characteristics of these solutions and if they are suitable for the present problems. All the tests carried out during this section are executed in a simulated fixed scenario.

REAL SCENARIO

In this chapter, once an SDN controller is configured, a real scenario with the SBC equipment in the laboratory begins to be configured too. The distribution of the components and how they are configured is specified in this chapter. The tests carried out in this section consist of two parts. First tests on the wired scenario with the controller configured and a first exploration of the wireless interface of the SBC devices.

CONCLUSIONS

As it has been observed, in this End-of-Grade Work two main differentiated tasks have been developed: the development and configuration of the SDN controller and the experimentation and study of the SDN technology with the Raspberry Pi devices. The work done on the controller is absolutely necessary to be able to perform tests with the Raspberry Pi. However, prior to the use of the controller on these devices had to make a study and configuration of them.

Before beginning with the development of these tasks a fundamental work was made. This work is the study of UAV networks. It is necessary to analyze the characteristics and needs of this type of networks and why SDN technology can be a good solution for the problems they present. In this way, correct configurations and implementations could be carried out later on.

During the whole project the importance of a correct configuration of the controller has been demonstrated, because it is the brain of the network and therefore it is necessary that it adjusts totally to the needs of the network. First of all, the Mininet tool was used to understand the network with different topologies. Afterwards, problems were identified and solutions were established in order to achieve the desired configuration. Finally, work was carried out in the laboratory to carry out a test of concept on an SBC device. These tests, which were done using Raspberry Pi, have been carried out on wired scenarios and a first use of the wireless interface has been completed.

Although it can be concluded that the objectives of configuring an SDN network on Raspberry Pi equipment have been met, it is a configuration that has several limitations, both in the development of the driver code and in the wireless interface of the Raspberry Pi.

It is important to continue with the development of a change in the controller for the detection of ad-hoc links in a wireless interface. The current implementation uses the LLDP protocol for neighbor discovery, but this protocol is useful in wired networks. In the exploration part of the wireless interface this fact was verified and therefore it is

proposed to change the neighbor discovery protocol to one that works in ad-hoc networks.

The continuation of the work on the wireless interface is also very important. As it has been possible to see the results obtained on the data plane must be improved. Currently there is a very low bandwidth. The aim of this whole project is to achieve a functional SDN network over a network of UAVs and therefore a proper functioning of the wireless interface is essential.

These two points would be the main works that should be carried out in an imminent future and that are totally necessary to achieve a functional network of drones using SDN technology.

Not to finish without remembering that the work made in this End-of-Grade Work is totally framed in a first exploration of SDN technology to configure data forwarding in SBC devices. But other works such as the flight control of drones or the implementation of SBC devices in flying equipment are essential so that in the future can be observed in the air the work done.

ANEXO IX - BIBLIOGRAFÍA

[1] **L. Gupta, R. Jain, G. Vaszkun**. *Survey of Important Issues in UAV Communication Networks.* 2016. [En línea] https://ieeexplore.ieee.org/abstract/document/7317490/ Último acceso: Julio de 2018.

[2] **J. Velasco, S. García-Nieto, G. Reynoso-Meza, J. Sanchis.** *Desarrollo y evaluación de una estación de control de tierra para vehículos aéreos no tripulados*. 2012. [En línea] https://riunet.upv.es/bitstream/handle/10251/17704/Velasco%20et%20Al.pdf?sequence=1 Último acceso: Junio de 2018.

[3] **I. Bekmezci, O.K. Sahingoz, S. Temel.** *Flying Ad-Hoc Networks (FANETs): A survey.* 2013. [En línea] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.404.23&rep=rep1&type=pdf Último acceso: Julio de 2018.

[4] **E. Haleplidis, K. Pentikousis, S. Denazis, J. Hadi Salim, D. Meyer, O. Koufopavlou.** *Software-Defined Networking (SDN): Layers and Architecture Terminology.* RFC 7426 .2015. [En línea] https://tools.ietf.org/html/rfc7426 Último acceso: Julio de 2018.

[5] **Open Networking Foundation.** *Software- Defined Networking: The New Norm for Nertworks.* 2012. [En línea] http://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf Último acceso: Julio de 2018.

[6] **Ashton, Metzler & Associates.** *Ten things to look for in an SDN Controller.* [En línea] https://www.necam.com/docs/?id=23865bd4-f10a-49f7-b6be-a17c61ad6fff Último acceso: Julio de 2018.

[7] **A. García, C.M. Rodríguez, C. Arias, F.C. Casmartiño.** *Controladores SDN, elementos para su selección y evaluación.* 2014. [En línea] https://www.researchgate.net/profile/Caridad_Anias_Calderon/publication/320711755_Controladores_SDN_elementos_para_su_seleccion_y_evaluacion/links/59f719770f7e9b553ebd5609/Controladores-SDN-elementos-para-su-seleccion-y-evaluacion.pdf Último acceso: Junio de 2018.

[8] **Ryu.** *Ryu SDN Framework.* [En línea] https://osrg.github.io/ryu-book/en/Ryubook.pdf Último acceso: Junio de 2018.

[9] **Open Networking Foundation.** *Open Networking Foundation.* [En línea] *https://www.opennetworking.org/* Último acceso: Agosto de 2018.

[10] **Open Networking Foundation.** *OpenFlow Switch Specification. Version 1.3.0.* 2012. [En línea] https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf Último acceso: Agosto de 2018.

[11] **P. Congdon, P. Blatherwick.** *802.1AB Overview Link Layer Discovery Protocol.* 2004. [En línea] http://www.ieee802.org/3/frame_study/0409/blatherwick_1_0409.pdf Último acceso: Agosto de 2018.

[12] **Raspberry Pi Foundation.** *What is a Raspberry Pi?* [En línea] https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/ Último acceso: Agosto de 2018.

[13] **Mininet Team.** *Mininet.* [En línea] http://mininet.org/ Último acceso: Mayo de 2018.

[14] **Cisco Systems, Inc.** *Understanding and Configuring Spanning Tree Protocol (STP) on Catalyst Switches.* 2006. [En línea] https://www.cisco.com/c/en/us/support/docs/lan-switching/spanning-tree-protocol/5234-5.pdf Último acceso: Julio de 2018.

[15] **NetworkX Developers.** *NetworkX.* 2004 - 2018. [En línea] https://networkx.github.io/ Último acceso: Junio de 2018.

[16] **Linux Foundation.** *Open Virtual Switch.* [En línea] https://www.openvswitch.org/ Último acceso: Agosto de 2018.

[17] **Raspberry Pi Foundation.** *Raspbian.* [En línea] https://www.raspbian.org/ Último acceso: Junio de 2018.

[18] **Boletín Oficial del Estado.** *Real Decreto 1036/2017, de 15 de diciembre.* [En línea] https://www.boe.es/boe/dias/2017/12/29/pdfs/BOE-A-2017-15721.pdf Último acceso: Septiembre de 2018.

[19] **Boletín Oficial del Estado.** *Real Decreto 8/2014, de 4 de julio.* [En línea] https://www.boe.es/boe/dias/2014/07/05/pdfs/BOE-A-2014-7064.pdf Último acceso: Septiembre de 2018.