Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería de Telecomunicación: Sistemas de Telecomunicación

Trabajo Fin de Grado

Implementación de una aplicación para la gestión de la información topológica basada en un controlador SDN

Autor: Irene Villa Murillo

Tutor: Isaías Martínez Yelmo

Cotutor: Joaquín Álvarez Horcajo

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Grado en Ingeniería de Telecomunicación: Sistemas de Telecomunicación

Trabajo Fin de Grado

Implementación de una aplicación para la gestión de la información topológica basada en un controlador SDN

Autor: Irene Villa Murillo

Tutor: Isaías Martínez Yelmo

Cotutor: Joaquín Álvarez Horcajo

TRIBUNAL:

Presidente: Elisa Rojas Sánchez

Vocal 1º: José Manuel Arco Rodríguez

Vocal 2º: Isaías Martínez Yelmo

FECHA: ……

A mi familia, amigos y pareja por animarme y apoyarme en este proyecto.

Agradecimientos

En primer lugar, se lo quiero dedicar a los pilares fundamentales de mi vida, mis padres, sin ellos, yo no estaría aquí, gracias por apoyarme, aconsejarme y animarme a ser cada día una mejor persona y gracias por estar siempre que os necesito, no dudéis que siempre que me necesitéis a mi, aquí me tendréis, sois las personas que más quiero en el mundo.

En segundo lugar, se lo quiero dedicar a mi pareja, pobrecillo, lo que ha tenido que aguantar, gracias por tener esa paciencia infinita que tienes conmigo y por sacarme una sonrisa o darme un abrazo siempre que lo necesito, y por acompañarme tanto en los momentos buenos como malos.

En estos agradecimientos no pueden faltar mis amigos de la universidad, con los que he compartido tantos agobios, lloros, días de biblioteca, pero con los que también he compartido muchísimos momentos buenos que no voy a olvidar jamás, sobre todo a Marta, María y Andrea, os llevaré siempre siempre conmigo y doy gracias al destino por habernos juntado.

En último lugar, se lo quiero agradecer a mis amigos de Meco, a los cuales quiero un montón porque son unas personas increíbles y siempre están para los momentos buenos y malos.

Gracias a todos.

Resumen

El objetivo de este Trabajo de Fin de Grado, es reimplementar una aplicación que procese la información provista por el protocolo eHDDP, para que un sistema basado en Open Network Operating System (ONOS), pueda obtener la información topológica de una red híbrida SDN, compuesta por dispositivos compatibles con eHDDP para el plano de control SDN. Esta aplicación adoptará el formato para su desarrollo y compilación mediante Bazel. Por último, se explicarán las conclusiones que se han obtenido al desarrollar y evaluar la aplicación dentro del entorno propuesto y los posibles trabajos futuros que se pueden desarrollar a partir de este proyecto.

Palabras clave: SDN, Openflow, ONOS, topología, protocolos de descubrimiento.

Abstract

The objective of this Final Degree Project is to reimplement an application that processes the information provided by the eHDDP protocol, so that a system based on Open Network Operating System (ONOS), can obtain the topological information of a hybrid SDN network composed with devices that are compliant with the protocol eHDDP for the SDN control plane. This application will adopt the format for its development and compilation using Bazel. Finally, the conclusions obtained from developing and evaluating the application, within the proposed environment and possible future work that can be developed from this project, will be explained.

Keywords: SDN, Openflow, ONOS, topology, discovery protocols.

Resumen extendido

La aparición de avances informáticos, el aumento de tráfico de datos y el aumento de las nuevas tecnologías, son las pruebas, de que, en los últimos años, la tecnología no ha parado de progresar, y esta progresión ha sido posible gracias a nuevas arquitecturas que han ido surgiendo, como por ejemplo, la arquitectura de red definida por software (SDN), que es la topología de red en la que se va a centrar este proyecto. Debido a este progreso tecnológico y al aumento del tráfico, surgió la necesidad de crear este tipo de redes, para poder gestionar la red de una forma más sencilla y para mejorar el rendimiento de las redes.

El objetivo de este Trabajo de Fin de Grado, es reimplementar una aplicación que gestione la información proporcionada por el protocolo eHDDP, para que un sistema basado en Open Network Operating System (ONOS), pueda obtener la información topológica de una red híbrida SDN, compuesta por dispositivos compatibles con eHDDP para el plano de control SDN.

El código de la aplicación a implementar es el protocolo eHDDP y fue proporcionado por los tutores de este proyecto, dicho código fue necesario adaptar para poder construir la aplicación como un proyecto Bazel, puesto que la aplicación original se creó sobre un proyecto Maven, para poder adaptar la aplicación se utilizó la herramienta Intellij. Una vez que el código fue adaptado a Bazel y que se ejecutaba correctamente, la aplicación se implementó en ONOS para poder obtener, mediante el protocolo eHDDP, la información topológica de las redes híbridas SDN. Para comprobar que la implementación había sido satisfactoria se simularon distintas topologías de red y se comprobó que la aplicación descubría la topología de la red sin problemas. Por último, se escribieron las conclusiones obtenidas de este proyecto y los posibles futuros trabajos que se pueden investigar.

Índice

Lista de figuras

Lista de tablas

Lista de acrómicos

AOSS - ARP-Path OpenFlow Software Switch

API - Application Programming Interface

BEBA - BEhavioural BAsed forwarding

BOFUSS – Basic OpenFlow Software Switch

CDP: Cisco Discovery Protocol

CPqD - Centro de Investigación y Desarrollo de Telecomunicaciones de Brasil

eHDDP - Enhanced Hybrid Domain Discovery Protocol

IETF - Internet Engineering Task Force

ISP - proveedores de servicio de Internet

LLPD - Link Layer Discovery Protocol

MAC - Mandatory Access Control

ODL - OpenDaylight

ONF - Open Networking Foundation

ONOS - Open Network Operating System

OSI - Open Systems Interconnection Model

SDN - Software-Defined Networking

TCAM - Ternary Content-addressable Memory

TCP – Transmission Control Protocol

TLS - Transport Layer Security

Capítulo 1: Introducción

El mundo de las telecomunicaciones no ha dejado de avanzar en ningún momento, y como prueba de ello, se tienen todos los hallazgos realizados desde el desarrollo del telégrafo hasta el 5G, el cual proporciona una mayor seguridad de red, menor retardo y una mayor velocidad [1]. Estos avances han sido posibles gracias a grandes descubrimientos que se han hecho a lo largo de la historia, como por ejemplo, el Sistema inalámbrico Mundial [2], inventado por Nikola Tesla, con este hallazgo, Tesla intentó hacerse una idea de cómo utilizando dicho invento, a través de señales inalámbricas, se podrían enviar mensajes desde cualquier parte del mundo. Los avances tecnológicos también han sido posibles debido a las nuevas arquitecturas que han aparecido, como la arquitectura definida por software (SDN), este tipo de arquitecturas aparecieron para satisfacer ciertas necesidades, como por ejemplo, hacer que la gestión de las redes sea más sencilla y que las redes sean más eficientes.

La arquitectura SDN divide el Plano de Datos del Plano de Control. En el Plano de Control se encuentra la Capa de Aplicación y la Capa de Control. En la Capa de Aplicación los administradores de red fijan, establecen y ponen en marcha las distintas aplicaciones y módulos que monitorizan y coordinan la red, en la Capa de Control, se encuentran definidos todos los servicios y funciones que los administradores de red poseen para poder gestionar y controlar la red.

El controlador SDN se comporta como un mediador entre los elementos que forman la red y los administradores [3]. El controlador SDN, está formado por una serie de módulos que realizan las funciones de almacenar datos, controlar la red, gestionar paquetes, incorporar nuevos protocolos, etc. Además, las aplicaciones de red y los nodos, que se encuentran en la capa superior e inferior respectivamente, reciben información del controlador SDN puesto que este se encarga de transmitírsela.

Los nodos que forman la red se encuentran en la Capa de Infraestructura, dichos nodos pueden utilizar el protocolo OpenFlow[4] o P4Runtime para poder comunicarse con el controlador SDN, pero este proyecto se centrará en OpenFlow, el cual es un estándar abierto, que establece el pipeline de los nodos y que paquetes debe de utilizar el controlador para poder comunicarse con los nodos. Además, mediante reglas, este protocolo establece cómo los dispositivos que se encuentran entre diferentes capas, se pueden comunicar entre ellos.

Existen protocolos de descubrimientos de topologías, para poder conocer cómo es la estructura de una red, como por ejemplo Link Layer Discovery Protocol (LLPD), los protocolos de descubrimientos son necesarios para la Capa de Control, puesto que es necesario que conozcan cómo está organizada la red, para poder realizar sus funciones correctamente.

En este proyecto, se va a implementar una aplicación basada en el controlador SDN ONOS, que tiene la capacidad de gestionar todos los datos que recibe por parte de la red sobre su topología.

1.1 Motivación

En este proyecto, se va a utilizar el protocolo de descubrimiento de topologías conocido como Enhanced Hybrid Domain Discovery Protocol (eHDDP) [5], para conocer cómo es la topología de redes basadas en un controlador SDN, el lenguaje de programación que se va a utilizar es Java.

eHDDP es un protocolo de descubrimiento de topologías, desarrollado por el grupo de investigación NETIS, perteneciente a la Escuela Politécnica Superior de la Universidad de Alcalá de Henares. El objetivo de este proyecto, es implementar una aplicación para gestionar la información topológica obtenida de distintas estructuras de redes. Este Trabajo de Fin de Grado, se ha realizado bajo la supervisión de Isaías Martínez Yelmo y Joaquín Álvarez Horcajo, este documento es un resumen de todos los conocimientos necesarios para llevar a cabo el proyecto, los pasos que se han seguido para realizarlo correctamente y las conclusiones obtenidas. Este proyecto es la última etapa necesaria para poder obtener el Grado en Ingeniería de Sistemas de Telecomunicación en la Universidad de Alcalá de Henares.

Algunas de las motivaciones que me han llevado a realizar dicho proyecto son:

- → Socioeconómicas: Hallar un protocolo que sea eficiente y rentable en el momento de descubrir topologías híbridas, conlleva que se pueden ir incorporando de forma gradual nuevos elementos SDN en las redes, lo que permite que las redes se actualicen y se pueda seguir progresando en tecnología, lo que conllevará que se progrese en otros ámbitos como la educación, sanidad, etc.
- → Técnicas: Estudiar protocolos de descubrimiento, para poder identificar en una red SDN híbrida, elementos SDN y no SDN es necesario, para poder aumentar las aplicaciones

del descubrimiento de topologías en la extensión de soluciones SDN. Adicionalmente, estudiar el comportamiento de un protocolo de descubrimiento de topologías en estructuras de red muy diferentes, es un tema interesante de investigar.

1.2 Objetivos

El objetivo de este Trabajo de Fin de Grado, es la implementación de una aplicación basada en Open Network Operating System (ONOS), que permita gestionar la información topológica obtenida por el protocolo eHDDP [6] en el plano de control SDN.

Para poder llevar a cabo el objetivo principal, es necesario cumplir con una serie de objetivos secundarios: estudiar y comprender la arquitectura SDN, además de analizar las ventajas y las limitaciones que ofrece la arquitectura, entender cómo el protocolo OpenFlow, a través de un conjunto de normas y procesos, logra conectar el Plano de control con los dispositivos de la Capa de Infraestructura. Entender el protocolo de descubrimiento LLDP (Link Layer Discovery Protocol). Aprender a utilizar y a configurar Mininet, una herramienta versátil que se utilizará para realizar las simulaciones de las diferentes topologías. Además, es necesario familiarizarse con el lenguaje de programación Java, puesto que es el lenguaje utilizado en la programación del protocolo eHDDP. Para gestionar la información topológica obtenida por dicho protocolo es necesario saber usar ONOS, puesto que se utilizará para representar distintas topologías.

1.3 Estructura y contenidos de la memoria

Este proyecto está estructurado en los cinco capítulos que se indican a continuación:

→ **Capítulo 1 - Introducción:** Se presenta la descripción del proyecto, lo que se va a hacer, una introducción para poder establecer un contexto, los objetivos a los que se quiere llegar y la motivación para llevar a cabo el proyecto.

→ **Capítulo 2 - Base Teórica:** En este capítulo, aparecen detallados todos los conceptos teóricos necesarios para poder realizar el proyecto. Algunas de los temas explicados en este proyecto son: las arquitecturas de redes, las redes SDN, cómo están conectadas las

distintas capas que forman dicha red, cuáles son sus desventajas, se comenta más detalladamente el protocolo eHDDP, el funcionamiento y las características de OpenFlow, etc.

→ **Capítulo 3 - Desarrollo:** Este puede ser uno de los capítulos más importantes del proyecto, puesto que se muestra cómo se ha realizado toda la parte práctica del proyecto. Aquí se muestra, cómo se compila, se instala y se crea un proyecto Bazel, cómo debe de ser la estructura de la aplicación, cómo se ha migrado la aplicación, cómo funcionan los distintos módulos que forman la aplicación a implementar y la implementación del protocolo eHDDP en ONOS.

→ **Capítulo 4 - Resultados:** Se mostrarán los resultados obtenidos al realizar con el protocolo eHDDP diferentes topologías.

→ **Capítulo 5 - Conclusiones:** Se realiza un análisis de los resultados obtenidos en el apartado anterior y se presentan ideas que se podrían estudiar en el futuro para continuar con el estudio de descubrimiento de topologías.

Capítulo 2: Estado del arte

2.1 Redes Definidas por software

2.1.1 Arquitecturas de red

La arquitectura de red está compuesta por un conjunto de protocolos de comunicación, dispositivos, programas e infraestructura que permite la conexión y el flujo de datos de distintos dispositivos, a través de la red [7]. Internet está todo el tiempo disponible para los millones de personas que están conectados a él, es por eso que para tener una buena arquitectura de red, se deben de tener en cuenta ciertas características, en este proyecto se van a explicar las cinco características básicas para disponer de una buena arquitectura de red.

En primer lugar, se dispone de la tolerancia frente a fallos, es decir, en caso de que ocurra un error software o hardware, la red tenga la capacidad de recuperarse de forma rápida, de manera que el usuario no note que haya ocurrido un error. En segundo lugar, se tiene la escalabilidad, esta característica se refiere a la capacidad que tiene una red, en adaptarse ante situaciones sin perder calidad, también se refiere, a cuando en la red se unen nuevos usuarios y la red se adapta para seguir proporcionando los servicios que ofrecía sin perder calidad. Además, para que una arquitectura de red sea buena, tiene que ofrecer una buena calidad de servicio y ser segura, puesto que mantener la confidencialidad de los datos de un usuario o de una empresa es muy importante. Por último, se tiene la gestión de la red, es decir, administrar y monitorizar los recursos que dispone la red para que funcione correctamente, además de controlar el flujo de datos que viajan a través de la red.

A lo largo del tiempo, han surgido diferentes tipos de arquitectura de red, como por ejemplo la arquitectura de red conocida como Ethernet, la cual es más rápida que la red wifi, debido a que en condiciones ideales en el estándar Gigabit Ethernet, se ha llegado a alcanzar una velocidad de 300 Mbps, otro ejemplo, es la arquitectura estrella, las redes que tienen este tipo de arquitectura están compuestas por dispositivos que se conectan directamente a un punto central, donde todas las comunicaciones se deben de realizar a través de este [7], en el caso en el que un dispositivo se estropee, sólo se verá afectado dicho dispositivo, y la red podrá seguir funcionando. Debido a la evolución tecnológica, han ido apareciendo nuevas arquitecturas de red, como por ejemplo las redes SDN que se explicarán con más detenimiento a continuación.

2.1.2 Evolución hacia las redes definidas por software

Las telecomunicaciones han sufrido, desde los años ochenta, una evolución de carácter exponencial, por lo tanto, las redes se han tenido que transformar para poder llevar a cabo el intercambio de información entre los distintos equipos. Es más, en los últimos años han surgido nuevos problemas que han impulsado a desarrollar redes más complejas, como por ejemplo el aumento de la demanda de tráfico de datos.

En los últimos años, han surgido nuevos servicios que han hecho que el tráfico aumente mucho, como por ejemplo, la necesidad de analizar cantidades muy grandes de datos, más conocido como Big Data, el número de datos es tan grande que las aplicaciones de software que se utilizan para procesar datos, no son capaces de trabajar con tanta rapidez, es por eso que para poder llevar a cabo el procesamiento de los datos, es necesario estar conectados a servidores especializados en el análisis de datos. Otro ejemplo, es el internet de las cosas, que nos permite conectar a la red desde elementos físicos cotidianos, como por ejemplo una persiana eléctrica, hasta dispositivos médicos. Gracias a esta conectividad se puede controlar desde un dispositivo que por ejemplo la persiana de nuestra casa suba o baje. Como último ejemplo se tienen los proveedores de servicio de Internet (ISP) también conocidos como proveedores de acceso a Internet, dichos proveedores empezaron a aparecer a finales de los años 80 y son las organizaciones o empresas que suministran acceso a Internet y servicios similares a los usuarios. Normalmente, los ISO son compañías que facilitan servicios de telecomunicaciones, como por ejemplo conexión telefónica.

2.1.3 Introducción a las redes definidas por software

Existen dos características claves que definen el comportamiento de las redes SDN [5]. En primer lugar, la existencia de un controlador software programable, es decir un elemento externo en el que se reúne la lógica de control. El controlador SDN, conoce cómo está estructurada la red y se comporta como un mediador entre los administradores de red y los dispositivos que forman la red, además es el encargado de enviar información a las aplicaciones de red de la capa superior y a los nodos que se encuentran en la capa inferior. En segundo lugar, se tiene la separación entre el plano de datos y de control, es decir, la abstracción de software sobre la disposición física de la red, esto permite que el plano de control sea programable [5], dicho plano se encarga del enrutamiento, y el plano de datos es el encargado de transmitir los mensajes entre distintas interfaces. Esta abstracción no es algo exclusivo de las redes SDN,

esta idea ya había sido planteada en otros escenarios, como por ejemplo, en la creación del IETF de Forces [5].

La arquitectura de las redes SDN está dividida entre las siguientes capas:

Capa de Aplicación

La Capa de Aplicación es la capa de mayor nivel en el modelo OSI, desde esta capa, el gestor establece las diferentes reglas que debe de seguir la red. Para que la Capa de Aplicación sea capaz de comunicarse con la Capa de Infraestructura, se necesita un "middleware", también conocido como controlador, dicho "middleware" se encuentra en la Capa de Control. Con la interfaz conocida como "Northbound API", que SDN establece, se consigue que las aplicaciones se puedan comunicar con la capa de control.

Capa de Control

La Capa de Control permite, a través de los protocolos de comunicación, configurar y gestionar la Capa de Infraestructura. Esta capa, conocida como controlador, hace más sencilla la interconexión entre la Capa de Aplicación e Infraestructura.

Capa de Infraestructura

La Capa de Infraestructura está compuesta por los diferentes nodos multicapa SDN [9]. En esta capa, es importante señalar que solamente se consideran dispositivos de la Capa de Infraestructura, a aquellos elementos que son capaces de interactuar con el protocolo OpenFlow.

Otra característica que se debe destacar, es que en esta capa no es necesario que todos los dispositivos sean del mismo fabricante, únicamente, cada nodo debe disponer del soporte del protocolo de comunicación, que permite a cada nodo la conexión con el controlador.

En la ilustración 1 se muestra una representación visual de la arquitectura SDN.

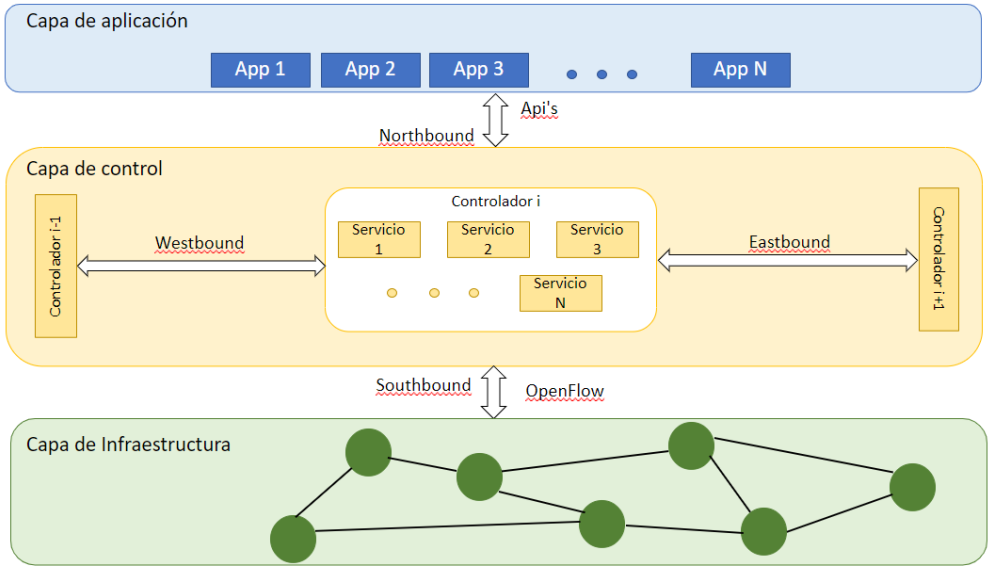


Ilustración 1:Arquitectura SDN

Interfaces de conexión entre capas

1. Eastbound y Westbound API

Las interfaces Eastbound y Westbound son de uso único para el controlador, son propias del estándar SDNi, dicho estándar soluciona el problema de la escalabilidad en SDN, permitiendo que los controladores puedan comunicarse entre ellos.

2. Southbound API

La Capa de Control e Infraestructura, están interconectados gracias a la interfaz Southbound. Esta interfaz se encarga de traducir a los switches SDN, los mensajes establecidos por el administrador, dichos mensajes pueden contener información sobre el reenvío de paquetes o incluso información propia de los switches, solicitada por alguna aplicación del controlador.

3. Northbound API

La interfaz Northbound, tiene como función principal interconectar la Capa de Aplicación con la del controlador. Dicha interfaz es la más crítica de las redes SDN puesto que da soporte a un gran número de aplicaciones y servicios.

Desventajas de las arquitecturas SDN actuales [8]

La arquitectura SDN, se ha expandido a redes de alto rendimiento, debido a las ventajas que esta proporciona como por ejemplo, el control de flujo, la configuración múltiple, es decir, permite configurar más de un switch a la vez, la programación de red, la pluralidad del hardware y la convivencia entre redes experimentales y productivas.

El problema de las arquitecturas SDN, es que tienen un gasto muy elevado puesto que el controlador puede sufrir una sobrecarga, debido a que posee un gran control sobre la red. Para eliminar este problema, Kandoo propone repartir el trabajo entre diversos controladores. Otra solución es asignar parte del estado de la red entre los conmutadores, como en DevoFlow [9], OpenState [10], o Difane [11].

Para supervisar los enlaces de una red, se utiliza el mecanismo LLPD [12], este mecanismo es un inconveniente, puesto que la sobrecarga producida en el controlador produce que no escale adecuadamente, lo que hace que las recuperaciones en este tipo de redes sean más lentas de lo esperado.

Otra desventaja de este tipo de arquitecturas, es la cantidad de flujos que puede recibir un switch, esto hace que el número de entradas en las tablas de los switches, sea muy grande, lo que hace imposible que esas tablas se puedan introducir en los TCAM actuales. Como TCAM tiene esta restricción se ha propuesto SRAMs, lo que hace que la escalabilidad mejore en este tipo de redes.

2.2 Protocolo OpenFlow

2.2.1 Funcionamiento OpenFlow

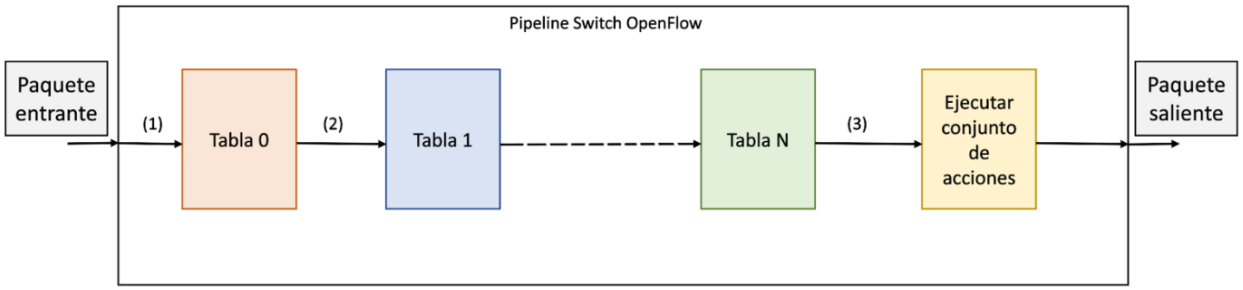
El protocolo OpenFlow [13] es un protocolo que establece una serie de normas, procesos y paquetes que hacen posible que los dispositivos que forman la Capa de Control y la Capa de Infraestructura, se puedan comunicar de manera correcta y sencilla, dicha comunicación se realiza a través de la interfaz Southbound API.

Adicionalmente, para que los switches SDN se integren correctamente en una red SDN, el protocolo OpenFlow establece una serie de normas para que la integración se realice correctamente.

OpenFlow Pipeline

El proceso Pipeline[8] es uno de los momentos más críticos del protocolo OpenFlow, puesto que este proceso se va a llevar a cabo sobre cada uno de los paquetes entrantes en los switches OpenFlow. Para poder llevarse a cabo, el proceso Pipeline utiliza una serie de tablas, organizadas de manera ordenada en cascada, las cuales pueden ser consultadas o modificadas por el controlador, además el controlador tiene la habilidad de crear nuevas tablas si es preciso.

El proceso que se lleva a cabo cuando un paquete entra en un switch OpenFlow, se basa en comparar de manera ordenada las reglas guardadas en las distintas tablas y almacenar las acciones asociadas a las reglas donde se haya producido una coincidencia, por último, todas las acciones almacenadas serán ejecutadas en el orden en el que se han ido guardando.



(1) El paquete tiene asociado un Puerto de entrada y un conjunto de acciones vacío.

(2) Se pasa el paquete junto a el puerto de entrada y un conjunto de metadatos. El conjunto de acciones puede ya no estar vacío.

(3) Se pasa el paquete y el conjunto de acciones a realizar.

Ilustración 2:OpenFlow Pipeline

En conclusión, Pipeline es un proceso compuesto por pasos repetitivos, es decir, cuando un paquete entra, se compara dicho paquete con la primera tabla, si se obtiene alguna coincidencia, se guarda la acción asociada y a continuación se pasa a la siguiente tabla para realizar los mismos pasos que se han hecho con la tabla uno, en otras palabras, este procedimiento se va a realizar con todas las tablas restantes. Cuando ya se han comparado todas las tablas, se ejecutan las acciones que han sido almacenadas de manera ordenada, para poder tramitar el paquete.

Tablas OpenFlow

El protocolo OpenFlow define un conjunto de tablas necesarias, para que el protocolo funcione correctamente y para almacenar diferentes datos. Las tablas pueden ser agrupadas en tres grupos [8]: en primer lugar, disponemos de la Tabla de Flujos, en ella se guarda toda la información vinculada al procesamiento, tratamiento y modificaciones necesarias sobre los paquetes recibidos o flujos que atraviesan el switch. En segundo lugar, tenemos la Tabla de Grupos, aquí se define cuando una misma acción puede ser realizada por diferentes dispositivos. Por último, se tiene la Tabla de Métricas, que se utiliza para llevar a cabo el control de los paquetes entrantes, su principal objetivo es producir una buena calidad del servicio, a través de la elaboración, configuración y supervisión de los servicios de red.

Tablas de flujos

Para que la Tabla de Flujo cumpla con sus objetivos, el protocolo OpenFlow establece cuales son los parámetros que debe de contener la tabla:

- → Campos de coincidencias: Estos parámetros se ocupan de guardar los datos significativos de los paquetes, por ejemplo, la dirección Media Access Control (MAC) origen, puerto de entrada, cabeceras, la dirección MAC de destino, incluso se pueden llegar a almacenar metadatos definidos por el desarrollador o por el protocolo OpenFlow.
- → Contadores: Son parámetros estadísticos que hacen referencia al conteo de los diferentes paquetes o flujos que son procesados por el switch.
- → Instrucciones: Constituyen las acciones o procesos que se deben efectuar en el caso en el que haya un ‘match’ con alguna regla.

Para agilizar la compresión de la Tabla de Flujos, se muestra un ejemplo en la tabla 1. En el ejemplo se puede observar cómo el controlador ha guardado en la Tabla de Flujos un conjunto de reglas para llevar a cabo cuando sea conveniente. Por ejemplo, en esta tabla se puede observar cómo en caso de que el EtherType sea igual a 00:54:35, el paquete tiene que ser encaminado hacia el controlador.

Campos de Match	Prioridad	Contadores	Instrucciones	Tiempo de espera	Cookie

Ilustración 3: Definición de Tabla de flujos

Campos de cabecera	Contadores	Acciones	Prioridad
Si puerto de entrada == 4		Tirar el mensaje	24135
Si puerto de entrada == 2		Reenviar por puerto 5	36547
Si EtherType == 00:54:35		Encaminar hacia el controlador	40000
Si IP Origen == 192.268.43.22		Actualizar el valor por 10.1.3.31 y reenviar por 5	32768
Si MAC Destino == AD:E2:2F:14:34:78		Añadir VLAN 14 y reenviar por puerto 2	56321

Tabla 1: Ejemplo de Tabla de Flujos.

Tabla de Grupos

Para poder ahorrar memoria y recursos en los sistemas, surgió la Tabla de Grupos, en ella se guardan las reglas que establecen como se debe de comportar un conjunto de elementos ligados a un grupo, de esta forma, el controlador puede reunir bajo unos mismos requisitos, a aquellos dispositivos que efectúen las mismas acciones. La estructura de la Tabla de Grupos se muestra a en la ilustración 4.

Identificador de grupo	Tipo de grupo	Contadores	Conjunto de acciones

Ilustración 4: Estructura Tabla de Grupos.

La Tabla de Grupos está compuesta por los siguientes indicadores:

→ *Identificador de Grupo*: Es el valor numérico que define de manera unívoca a cada grupo.

→ *Tipo de Grupo*: Especifica el Tipo de la agrupación. Dentro de este parámetro hay diferentes Tipos, tres obligatorios, denominados Todos, Seleccionado e Indirecto y otro opcional, Conmutación por error rápida.

- *Todos*: Se usa para transmitir tráfico broadcast y multicast, además establece que todas las acciones fijadas para los grupos son ejecutadas por los dispositivos del grupo.
- *Seleccionado*: Sólo se lleva a cabo una acción del grupo.
- *Indirecto*: Es similar al Seleccionado, pero la acción que se va a efectuar está definida en este grupo, es decir, que esa acción está dentro del conjunto de acciones posibles para el grupo en cuestión.
- *Conmutación por error rápida*: Sin consultar con el controlador, este parámetro es capaz de variar el reenvío de paquetes.

→ *Contadores*: Se utilizan para realizar datos estadísticos.

→ *Acciones asociadas*: Es un grupo de procesos y acciones que se deben de realizar a los paquetes que coincidan con las reglas del grupo.

Identificador de Grupo	Tipo de Grupo	Contadores	Conjunto de acciones
3			Tirar paquete
9			Puertos de salisa a, b
13			ID de Grupo E
17			Puertos de salida c, d
5			ID de Grupo G

Tabla 2: Ejemplo de Tabla de Grupos.

Ejemplo de datos en tablas [13]

En la siguiente figura se muestran diferentes ejemplos sobre las reglas que deben de seguir los switches a la hora de enviar paquetes en la red SDN.

Conmutación

Puerto switch	MAC origen	MAC destino	EtherType	ID VLAN	IP origen	IP destino	IP prot	TCP sport	TCP dport	Action
*	*	00:2E:EE ...	*	*	*	*	*	*	*	Puerto 4

Firewall

Puerto switch	MAC origen	MAC destino	EtherType	ID VLAN	IP origen	IP destino	IP prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	*	*	*	*	Tirar paquete

Ilustración 5: Ejemplo de reglas.

Tipos de Mensajes

Los mensajes de control, son los mensajes que los dispositivos intercambian en el caso de que estén utilizando el mismo protocolo, en el caso del protocolo OpenFlow, se definen tres tipos de mensajes de control: los Mensajes de Controlador a Switch, es decir aquellos mensajes que siempre se envían desde el controlador hacía un switch de la red. Este tipo de mensajes tienen como objetivo configurar o preguntar por reglas o estados de un switch SDN. En este caso, la comunicación es comenzada por el controlador y no tienen por qué recibir una respuesta por parte del switch, por ejemplo, puede tratarse de la instalación de una regla, en este caso, el controlador no recibirá ningún mensaje como respuesta.

A continuación, se tienen los Mensajes Asíncronos, este tipo se refiere a los mensajes recibidos en los controladores siempre por parte de los switches, en este caso el mensaje siempre es del tipo request - reply, además el switch SDN es siempre el que comienza la comunicación con el controlador. Estos mensajes pueden realizar diferentes funciones, es por ello, que se va a enfatizar en alguna de ellas, como, por ejemplo, para poder realizar ciertas acciones el switch SDN, puede solicitar información o informar de que el switch o la red ha sufrido alguna modificación en el estado.

En la ilustración 6, se puede observar cómo h1 envía un paquete al switch SDN s1, este paquete es desconocido por s1, por ello, el switch le pide información al controlador a través de un mensaje Packet_In, para saber hacia donde tiene que encaminar el paquete. Cuando el controlador recibe este tipo de mensajes realiza un análisis para saber que se tiene que hacer con ese paquete, una vez que se ha realizado dicho análisis, el controlador envía un Packet_out en el que aparecen, todas las acciones que se deben de realizar cuando aparece un paquete de ese tipo. Si al switch SDN s1, le llega el paquete desde h2 se seguirá el mismo procedimiento.

Por último, se tiene los Mensajes Simétricos, en este caso los mensajes pueden ser enviados tanto por los switches como por el controlador [8], la comunicación puede ser iniciada por el switch o por el controlador de red. Al igual que en el caso de los Mensajes Asíncronos, estos mensajes son del tipo request - reply.

Los Mensajes Simétricos tiene en dos tipos de mensajes: el primero de ellos son mensajes con los cuales el controlador llega a conocer las propiedades físicas y lógicas de un puerto, un switch o incluso puede llegar a recibir datos estadísticos almacenados. El segundo tipo de mensajes, son conocidos como Keep-alive, este tipo permite al controlador saber que switches están activados.

En la siguiente figura, se ha representado los mensajes más característicos de los Mensajes Simétricos. En primer lugar, se dispone de un mensaje Hello, el cual es un ejemplo de un mensaje Keep-Alive. Posteriormente, se tienen dos mensajes Feature_Request y Feature_Reply, que proporcionan al controlador información sobre las propiedades físicas de un switch. Finalmente, se tienen los mensajes Port Desc Stats Request y Port Desc Stats Reply, que permiten al controlador conocer información relacionada con las interfaces y puertos de un switch.

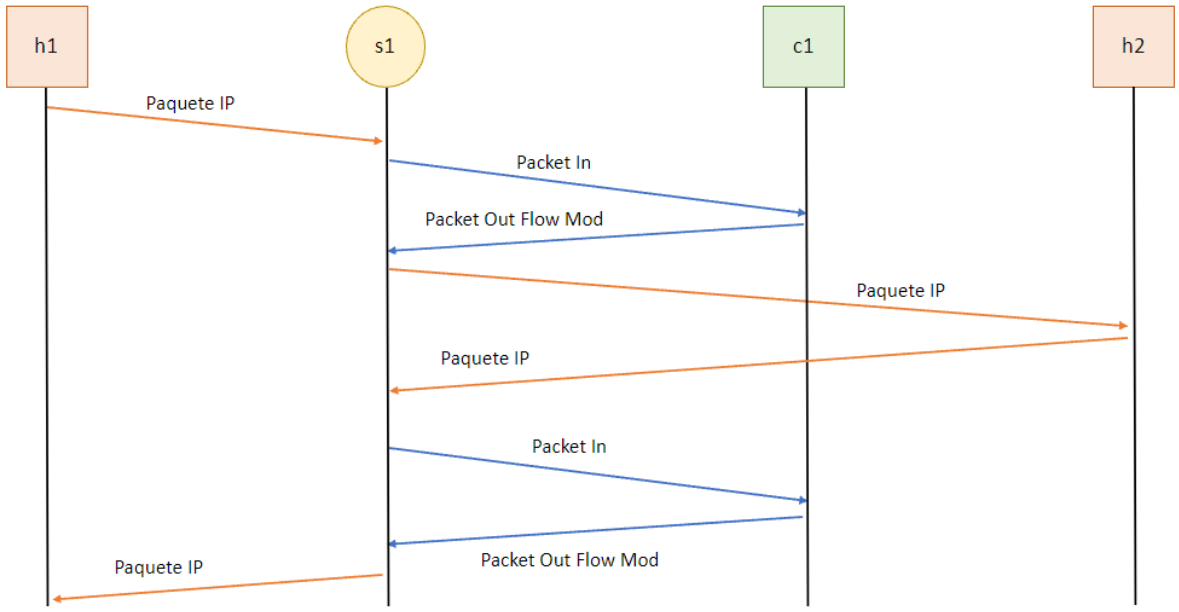


Ilustración 6: Intercambio de mensajes asíncronos.

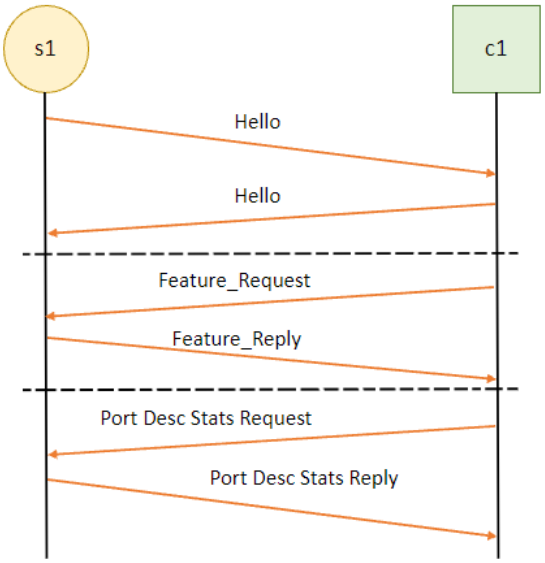


Ilustración 7: Intercambio de mensajes simétricos.

2.2.2 Switch OpenFlow

Como se ha citado anteriormente, el protocolo OpenFlow, establece las normas y características necesarias que debe seguir un dispositivo para poder ser considerado un switch OpenFlow [8] y además ser compatible con el protocolo.

Las características que un switch OpenFlow debe de seguir son las siguientes. Para empezar, se debe de tener una tabla de flujos, es decir una tabla donde el controlador tenga la capacidad de introducir reglas que permitan asociar una o más acciones a un cierto tipo de paquetes. A continuación, el switch tiene que ser capaz de crear una comunicación con el controlador. La comunicación que el switch establece con el controlador se puede hacer a través de mecanismos Transport Layer Security (TLS) para poder proporcionar seguridad extra y siempre debe de hacerse en un canal TCP. Por último, el switch debe de disponer de una tabla de grupos, es decir, una tabla donde se guarden una serie de reglas establecidas para los grupos.

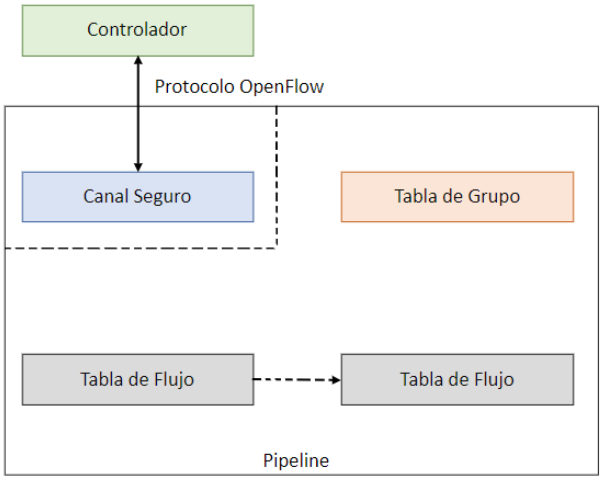


Ilustración 8: Funcionamiento del canal OpenFlow.

Lógica funcional de un switch OpenFlow [8]

En la siguiente figura, se enseña el diagrama de flujo funcional del switch OpenFlow 1.3, en ella se puede ver cómo, para empezar, el switch realiza un análisis y lectura de los datos, a continuación, estos datos se guardarán en la cabecera de la trama entrante, algunos ejemplos de los datos guardados son, las direcciones MAC de la cabecera de enlace o los puertos origen y destino de la cabecera de transporte. Después, estos datos se introducirán en las estructuras internas del switch para poder llevar a cabo las acciones necesarias. A continuación, el switch realiza una comparación entre las reglas que se encuentran en las tablas de los dispositivos y los datos recibidos de las cabeceras de la trama.

Existe cierta posibilidad de poseer dos o más tablas, donde el controlador sea capaz de incluir sus reglas o localizar distintas reglas para el mismo tipo de paquete, en caso de que esto ocurra, el switch analiza las reglas producidas según su orden de prioridad. Al finalizar dicha comparación, si se haya alguna regla válida, la acción ligada a la regla se incluirá en una lista de acciones a llevar a cabo con o sobre la trama. En el caso en el que no se encuentre ninguna coincidencia, quiere decir que no hay ninguna acción en la lista, por lo tanto, se empieza a buscar en la tabla conocida como 'Miss-Table', en esta tabla, el controlador define la acción que se debe efectuar, en el caso de que tampoco se pueda llevar a cabo ninguna de las acciones establecidas en la tabla 'Miss-Table', la trama se descartará.

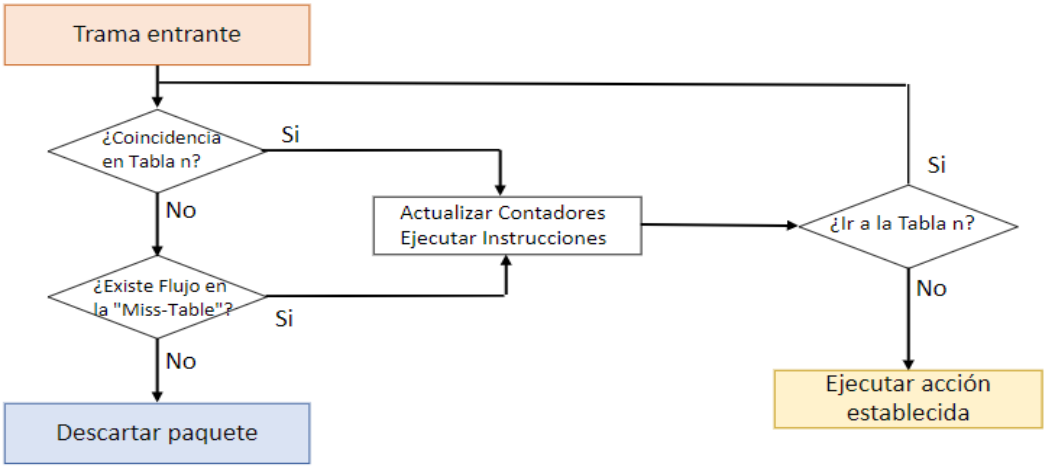


Ilustración 9: Diagrama de flujo switch OpenFlow 1.3

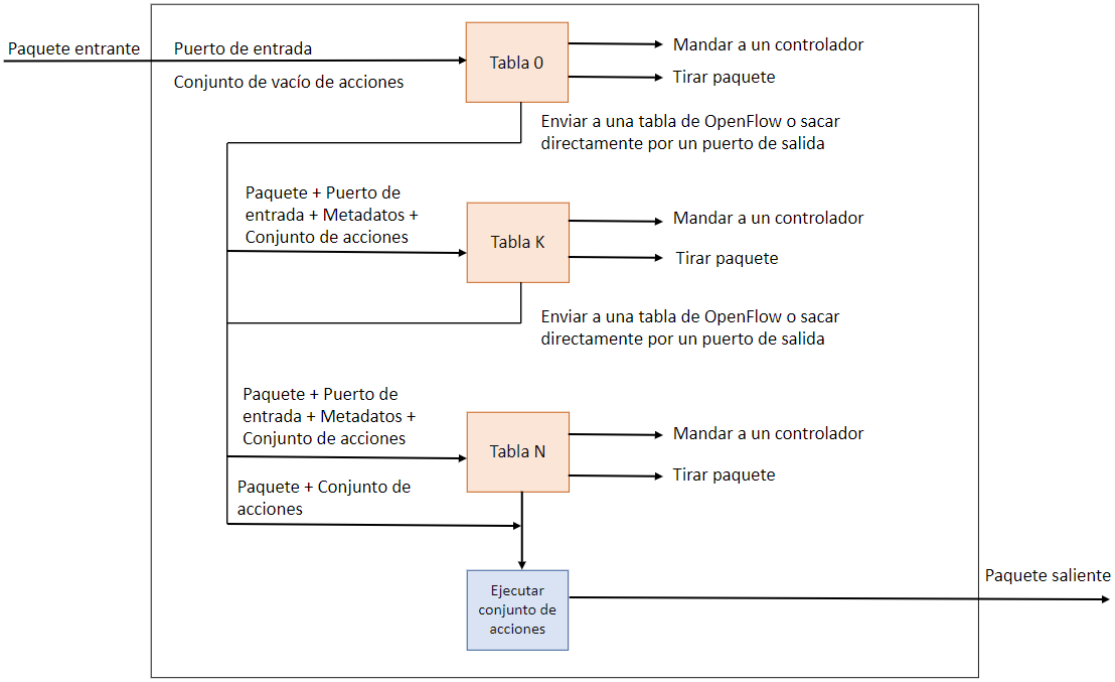


Ilustración 10: Procesado interno de una trama switch SDN.

Basic OpenFlow Software Switch (BOFUSS)

Hoy en día, hay disponibles numerosas implementaciones software de dispositivos que son compatibles con la versión 1.3 del protocolo OpenFlow. Este TFG se va a centrar en Basic OpenFlow Software Switch (BOFUSS) [14], un switch software puesto en marcha por

el Centro de Investigación y Desarrollo de Telecomunicaciones de Brasil (CPqD), dicho switch intenta llevar a cabo las especificaciones proporcionadas por la versión 1.3 de OpenFlow.

BOFUSS es un switch software completamente desarrollado en espacio de usuario de Linux [8]. El principal objetivo es desarrollar un switch software que sea lo más exacto posible a las distintas propiedades establecidas por el protocolo OpenFlow 1.3. Para poder llevar a cabo el objetivo principal, se ha buscado una implementación sencilla y accesible, el lenguaje utilizado para la implementación es C/C++.

BOFUSS está compuestos por tres bloques. El primer bloque es el responsable de parsear la trama, el siguiente bloque contiene el funcionamiento del protocolo OpenFlow y el último elemento contiene las tablas donde se almacena la información de los puertos, grupos o de los flujos.

BOFUSS comienza a trabajar, cuando a través de los puertos de entrada, el switch recibe una trama, en ese momento, el sistema empieza a efectuar llamadas periódicas a los buffers de recepción, dichas llamadas se hacen a través de 'Sockets RAW'. Si en los buffers de recepción se encuentra algún objeto se traslada al segundo bloque, en este caso la librería NetBee Link debe transportar la trama desde el Espacio Kernel, hasta el espacio de Usuario, es decir desde la interfaz de red, hasta el espacio de operaciones asequible por BOFUSS, a continuación, NetBee Parser coge los datos de las cabeceras de los paquetes y debe leer y deserializar dichos datos para poder introducirlos dentro del switch. Posteriormente, comienza un proceso controlado por el bloque de control de OpenFlow, en el que se comparan los valores obtenidos anteriormente, con las distintas reglas almacenadas dentro de la Tabla de Flujos. Cuando el proceso de comparación finaliza, la Tabla de Medidas del switch se actualiza con los nuevos datos obtenidos del paquete procesado.

En la Ilustración 11 se muestra una representación visual de la arquitectura BOFUSS.

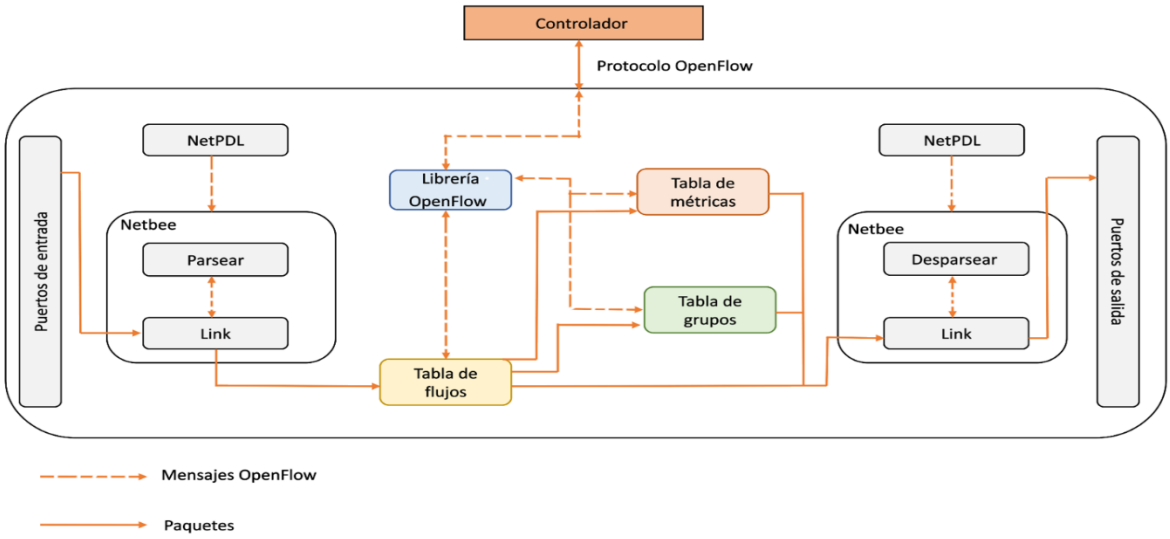


Ilustración 11: Arquitectura BOFUSS.

***Casos de uso de BOFUSS** [8]*

BOFUSS es un switch que solamente actúa en espacio de usuario, además su diseño y su programación son muy sencillos, lo que facilita a la comunidad científica que se puedan hacer ciertas modificaciones sin mucho esfuerzo. Por estas razones, en diferentes estudios y desarrollos este switch ha sido seleccionado para llevar a cabo dichos estudios.

A continuación, se van a nombrar algunos desarrollos:

→ BEhavioural BAsed forwarding (BEBA) [15], pertenece al grupo de proyectos europeos H2020. En este proyecto se logró tener una capacidad alrededor de los 1,19 Gbps desde una tasa de transferencia media de 185 Mbps, con dicho incremento, se consiguió que el switch BOFUSS fuera más eficiente.

→ ARP-Path OpenFlow Software Switch (AOSS) [13]. En este desarrollo se tiene un switch híbrido SDN que se ha programado para poder efectuar el protocolo ARP-Path [16]. Con el protocolo ARP-Path, el número de flujos que el controlador tiene que tratar es menor, debido a que el switch puede generar caminos de manera autómata, esto tiene un inconveniente, y es que en este caso el controlador no va a conocer de forma directa cómo se ha repartido el tráfico en la red, puesto que no puede revisar las tablas de reenvío de ARP-Path.

→ OnLife [17], es un desarrollo perteneciente al proyecto CORD [18] de Telefónica. Este proyecto tiene dos objetivos: mejorar la calidad de los servicios de red y crear una red

en la que no haya protocolos heredados. Dichos objetivos se consiguen acercando al usuario final los servicios de red.

2.2.3 Controlador SDN

El plano de control de la red está formado por controladores SDN. A través de las interfaces Southbound y Northbound API, los controladores se comunican con las aplicaciones de red y con los equipos del plano de datos. En el caso de las redes SDN, el enrutamiento de los elementos que forman la red en inicializado y configurado por el controlador, es decir, en el caso de estas redes, la inteligencia de red es trasladada al controlador.

Para que la red funcione correctamente, hay que tener en cuenta ciertas propiedades del controlador, debido a que lleva a cabo funciones importantes, por ejemplo, gestión de la red y los dispositivos, guardar datos, enrutamiento, descubrimiento de la topología de la red, etc. Las propiedades a tener en cuenta son las siguientes [19]:

- → Funcionamiento correcto de las interfaces Southbound y Northbound.
- → Buen rendimiento.
- → Programabilidad.
- → Soporte por parte de OpenFlow a la red.
- → Mediante un lenguaje de programación flexible se conseguirá compatibilidad con otras plataformas.

Hoy en día, se dispone de una gran variedad de controladores SDN, debido a que con ellos se puede gestionar la red de manera más sencilla. A continuación, se muestra en una tabla algunos ejemplos de los controladores más significativos. Los más importantes son: el controlador ODL [20], que tiene la habilidad de en un mismo cluster, unir la inteligencia de distintos controladores [5], también tenemos el controlador Beacon [21], el cual se desarrolló para que la red fuera más eficiente. Otro controlador importante es el que se va a desarrollar a continuación y con el que se va a trabajar en este proyecto: Open Network Operating System (ONOS).

	BEACON	ODL	ONOS
Versión OpenFlow	v1.0	v1.0, v1.2, v1.3	v1.0, v1.2, v1.3
Lenguaje	Java	Java	Java
Plataforma	Linux	Linux	Linux
Soporte OpenStack	Si	Si	Si
Mutiproceso	Si	Si	Si
Código abierto	Si	Si	Si
API REST	Si	Si	Si

Tabla 3: Ejemplos controladores

ONOS

En este proyecto se ha elegido ONOS debido al alto rendimiento y escalabilidad, además también posee una gran capacidad para trabajar con redes SDN híbridas.

ONOS es un sistema operativo de red, diseñado para administrar elementos como routers o ordenadores, es decir, elementos pertenecientes del plano de control. ONOS es un proyecto de código abierto, que se pone en marcha en el plano de control para las redes SDN, este proyecto es encabezado por la ONF.

La arquitectura de ONOS fue creada con la idea de mantener la reconfigurabilidad y el agnosticismo del protocolo, y la modularidad del código [5]. La estructura del controlador está dividida según los distintos niveles de funcionamiento, conocidos como servicios. Los servicios están compuestos por un grupo de elementos conocidos como subsistemas. Cada uno de los servicios, realiza diferentes funciones, como por ejemplo, la enumeración de equipos. En la siguiente figura se muestra la arquitectura del controlador ONOS.

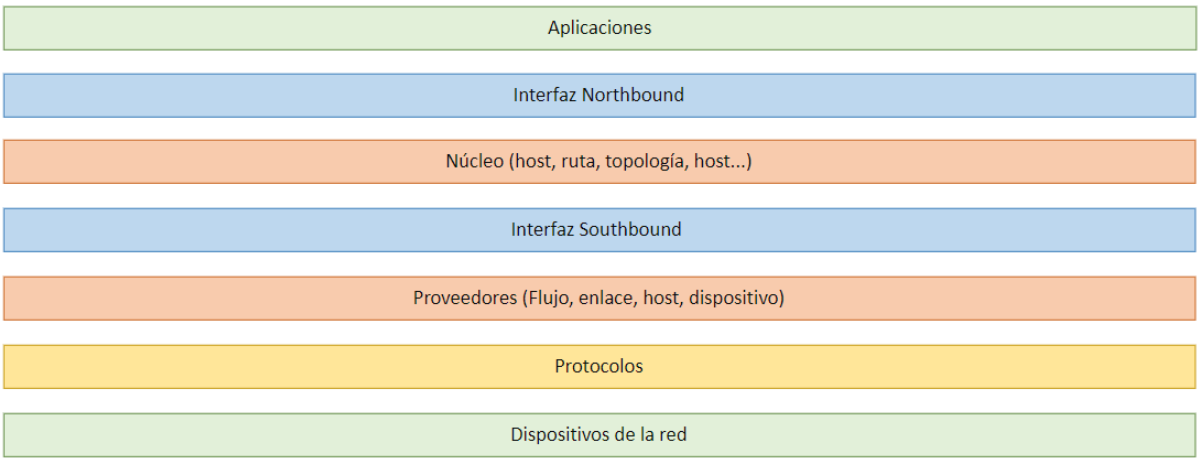


Ilustración 12: Estructura de ONOS.

ONOS puede realizar diseños de planos de transferencia a gran escala y comportarse como un sistema distribuido en múltiples servidores para poder funcionar en numerosas plataformas. Además, mediante la interfaz Northbound, ONOS se enlaza con la Capa de Aplicación, lo que consiente que las aplicaciones dejen de estar unidas a la Capa de Aplicación, esto conlleva que las aplicaciones también se desacoplen de los protocolos de comunicación. Esto les proporciona a las aplicaciones trabajar de manera independiente de la interfaz Southbound y de plataformas del plano de datos contiguas.

2.3 Protocolos de descubrimiento de topologías

A continuación, se van a explicar dos protocolos de descubrimiento de topologías, uno de ellos es el protocolo eHDDP, que es en el que más se va a centrar este apartado y el protocolo LLDP (Link Layer Discovery Protocol) puesto que este protocolo es una de las aplicaciones que dispone ONOS.

2.3.1 LLDP: Link Layer Discovery Protocol

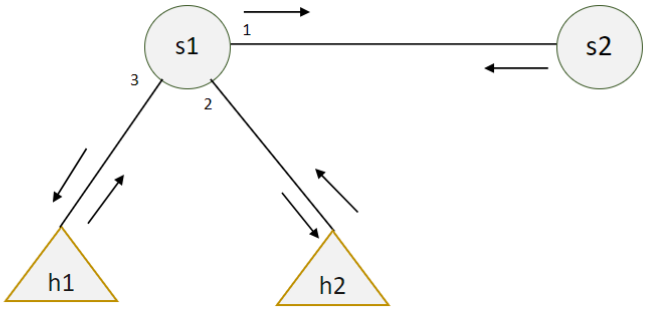
El protocolo LLDP tiene como propósito descubrir los vecinos que están conectados directamente a un dispositivo de la red. Su lógica es la siguiente, cada dispositivo que utiliza dicho protocolo intercambia información con cada uno de sus vecinos, la información recibida se reúne en la memoria interna del dispositivo. Para que el protocolo pueda funcionar, es necesario enviar una trama conocida como LLDP Frame, esta trama está compuesta por el identificador del dispositivo, identificador del puerto, tiempo de vida del paquete y del

identificador de final de trama LLDP [8]. Además, la trama puede contener más información opcional, como por ejemplo: la descripción del puerto, funciones del dispositivo de red, el nombre, etc. [8]

Se puede llegar a la conclusión de que el protocolo LLDP es un protocolo de descubrimiento de vecinos en vez de descubrimiento de topologías, pero con la información que los vecinos proporcionan se puede obtener la información necesaria para conocer la topología de la red.

Identificador del dispositivo	Identificador del puerto	Tiempo de vida del paquete	Información adicional	Final de trama

Ilustración 13: Trama LLPD.



Id puerto	Dispositivo	Información
1	s2	Conmutador
2	h2	Equipo final
3	h1	Equipo final

Ilustración 14: Funcionamiento protocolo LLPD.

2.3.2 eHDDP: Enhanced Hybrid Domain Discovery Protocol

eHDDP [23] es un protocolo capaz de descubrir de manera eficiente, la topología completa de los despliegues híbridos SDN, es decir, de elementos SDN y no SDN, así como la interconexión establecida entre dichos elementos.

Una de las ventajas del protocolo eHDDP, es que el controlador dispone de una visión real de la red, independientemente de los elementos que formen la red, además con dichas topologías, se puede garantizar el descubrimiento completo de subtopologías compuestas por dispositivos no SDN.

Otra de las ventajas de eHDDP, es que el número de anomalías que hay en la red puede disminuir y además, puede realizar diferentes tareas de gestión de tráfico en la red, debido a que, con este protocolo, se puede supervisar la movilidad de los sensores.

En conclusión, eHDDP puede considerarse un protocolo innovador y efectivo en el descubrimiento de topologías de redes híbridas.

Definición de eHDDP

En el descubrimiento de topologías, eHDDP es capaz de detectar elementos no SDN en una red, pero como dichos dispositivos no pueden estar conectados con el controlador, dependen de los dispositivos SDN para reenviar su información al controlador.

eHDDP es un protocolo que se basa en el intercambio de información mediante mensajes de control eHDDP Request y eHDDP Reply, a continuación, se muestra su estructura:

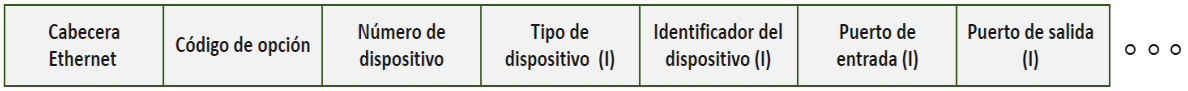


Ilustración 15: Mensaje de control eHDDP.

Los campos que contienen los mensajes eHDDP Request y eHDDP Reply recopilan la información necesaria para que el controlador pueda reconstruir la topología de la red híbrida, dichos campos, deben de ser completados por los nodos a medida que se realiza el proceso de descubrimiento.

eHDDP se implementa en la Capa de Infraestructura, es por eso que la estructura de los mensajes se basa en la longitud mínima de las redes Ethernet, aunque depende de la infraestructura en la que se aplique el protocolo, como por ejemplo en redes inalámbricas.

El funcionamiento del protocolo se muestra en la ilustración 16, aunque puede resumirse como:

(1). Cada uno de los nodos SDN recibe un mensaje request del controlador SDN, dicho mensaje viene encapsulado en un PACKET_OUT.

(2). Cuando los nodos SDN reciben el PACKET_OUT, lo desencapsulan y lo difunden por todos sus puertos excepto por el entrante.

(3). A continuación, el mensaje request será recibido por los nodos vecinos SDN y no SDN:

a. Los nodos no SDN que reciben el mensaje request, actualizan el contador de número de dispositivos y envían el mensaje por todos los puertos menos por el de entrada. Además, los nodos no SDN bloquean el puerto por el que ha entrado el mensaje request para evitar bucles.
b. Mediante un PACKET_IN, los nodos SDN reenvían al controlador el mensaje request que han recibido. El valor Num Device es utilizado por el controlador para describir si dos nodos SDN son vecinos o si hay dispositivos no SDN entre nodos SDN. En el caso de que no haya elementos no SDN NumDevide será igual a 1, en el caso en que si haya algún elemento no SDN Num Device será > 1. Cuando tengamos algún elemento no SDN se contesta con un mensaje eHDDP reply para hacer llegar la información de los dispositivos no SDN al controlador.

(4). En el caso en el que un dispositivo no SDN reciba, a través un puerto diferente al bloqueado, otro mensaje eHDDP request, se creará un mensaje eHDDP reply pero utilizando el mensaje eHDDP request recibido, a este mensaje se le añadirá tuplas de información al final del paquete y se cambiará el valor de los campos Opcode y NumDevice. Por último, el nodo enviará el mensaje eHDDP reply a través del puerto bloqueado.

(5). Cuando el mensaje eHDDP reply llega a un nodo de la red:

a. Si se tiene un nodo no SDN, el valor del campo NumDevice aumentará una unidad, es decir, dicho campo aumenta a medida que pasa por cada dispositivo no SDN de la red.

Una vez que aumente el valor del campo NumDevice, se añadirá información y se reenviará el mensaje a través del puerto bloqueado.

b. Si se tiene un nodo SDN, el dispositivo enviará el paquete eHDDP reply al controlador mediante un paquete Packet_IN.

Con este último paso, el controlador sabrá cómo es la topología de la red híbrida.

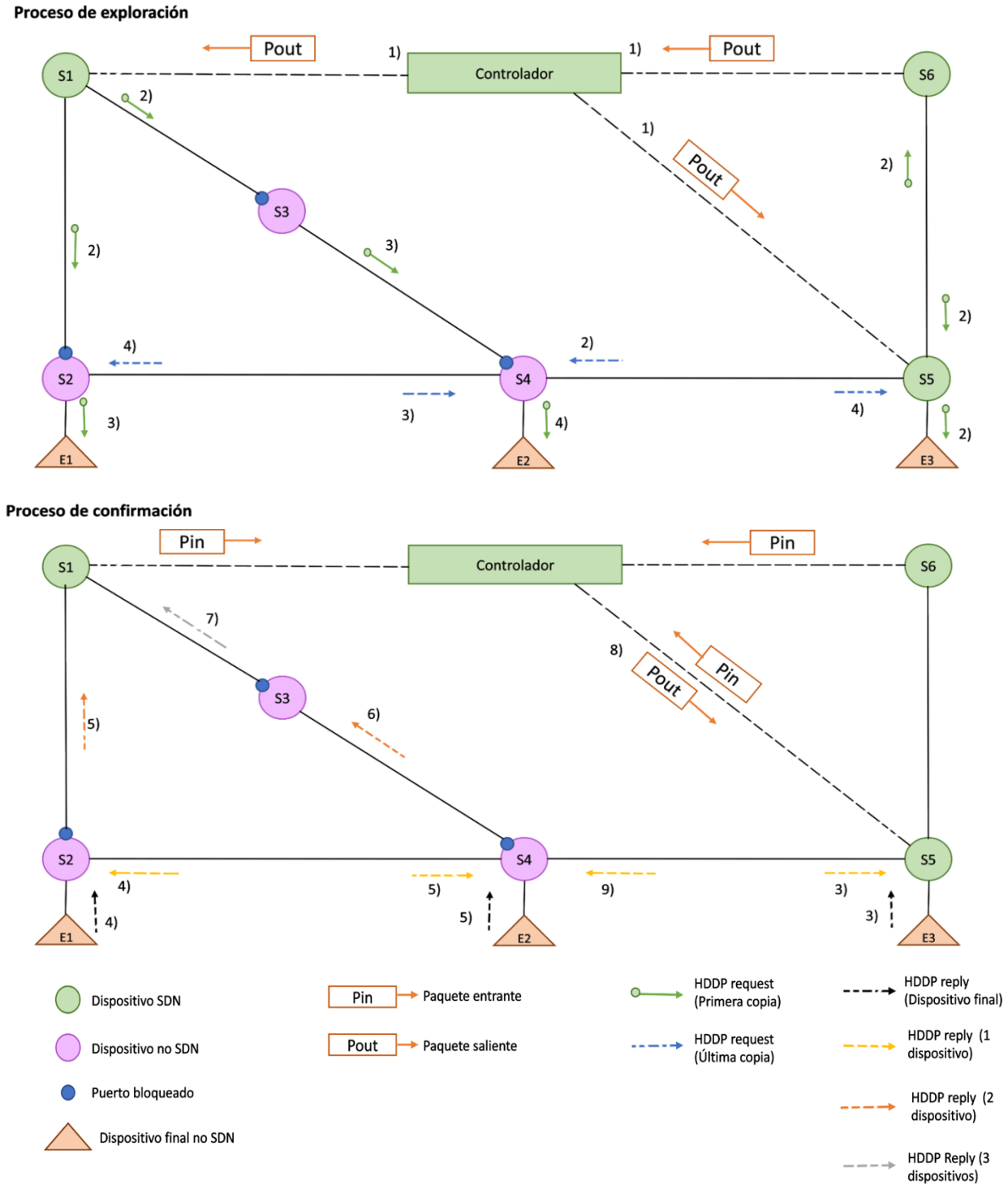


Ilustración 16: Funcionamiento protocolo eHDDP.

El controlador ONOS en eHDDP

En el protocolo eHDDP se dispone, en primer lugar, del mensaje eHDDP request, el cual se utiliza para explorar la red, en segundo lugar, se tiene el mensaje eHDDP reply que se utiliza para poder recoger información sobre los elementos que forman la red. Dichos mensajes, son reenviados por cada uno de los nodos no SDN hacia un nodo SDN, para que este los envíe al controlador para su posterior análisis.

A través de la API OpenFlow, el controlador ONOS, es capaz de comunicarse con los nodos SDN. El controlador, es responsable de crear los paquetes de control eHDDP, y de reenviar y recibir dichos paquetes de los switches de la SDN. A continuación, con la información recibida a través de los paquetes, el controlador podrá conformar la topología de la red.

ONOS trabaja con eHDDP usando una serie de programas que han sido desarrollados anteriormente. En primer lugar, se ejecuta un código escrito en Java para controlar, verificar y configurar el servicio eHDDP

Para que el controlador ONOS pueda construir los mensajes eHDDP request y eHDDP reply, el controlador ejecuta el programa DHTpacket.java. En la figura 15, se muestra cómo están estructurados los campos de los mensajes eHDDP, además, se incluyen también ciertos campos adicionales, necesarios para poder implementar el protocolo. Es importante destacar que, aunque se está hablando todo el rato de dos tipos de mensaje, request y reply, el controlador ONOS únicamente crea un tipo de mensaje y los campos que lo forman son los que caracterizan si el mensaje es de solicitud o respuesta.

Los campos fijos que forman el paquete eHDDP son los siguientes:

- → Mac_ant: Dirección física del dispositivo anterior. Este parámetro solamente se utiliza en redes inalámbricas.
- → Last_mac: Dirección física del último dispositivo que actualiza el paquete. Únicamente se utiliza en redes inalámbricas.
- → Src_mac: Dirección física del dispositivo que crea el paquete. Solamente se usa en redes inalámbricas.
- → Num_hops: Número de nodos por los que ha pasado el paquete.
- → Num_sec: Número de secuencia, en esta implementación se usa además para identificar el momento de lanzamiento desde el controlador.

→ Opcode: Identifica el tipo de paquete. En el caso de que Opcode sea igual a 1, se dispondrá de un mensaje eHDDP request, cuando Opcode sea igual a 2 será un mensaje eHDDP reply.

→ Time_block: Tiempo de bloqueo parametrizable por el controlador.

Los campos de tupla eHDDP:

→ Type_devices: Contiene el tipo de dispositivos.

→ Outports: Dispone de los puertos de salida.

→ Inports: Campo que contiene los puertos de entrada.

→ Num_ack: Número aleatorio utilizado para realizar las sumas de comprobación.

→ Bidirectional: Es un campo en el que se indica la conexión bidireccional con el nodo anterior

→ Id_mac_devices: En este campo se dispone de los ID de los dispositivos.

En primer lugar, el controlador ONOS agrupa los campos de cabecera fijos de los paquetes eHDDP y se dispone de una tupla de diferentes campos de cabecera de la matriz, vacía al principio. Cada vez que un conmutador no SDN ejecuta el agente eHDDP se añadirá al final de la trama[5].

En el caso de esta implementación, se tiene una limitación con el parámetro MAX_DEVICE, dicha limitación fue encontrada por los desarrolladores cuando se estaba creando la aplicación del controlador ONOS. Los paquetes que envíe el controlador ONOS contienen el parámetro MAX_DEVICE que va a ser igual a 31, este valor muestra el número máximo de nodos contiguos no SDN que se van a poder descubrir, es decir el número máximo de tuplas que puede tener un paquete.

El controlador ONOS iniciará el proceso de descubrimiento enviando el paquete PACKET_OUT que tendrán encapsulados los mensajes eHDDP request a los conmutadores SDN que forman parte de la red, dichos paquetes podrá recibirlos de vuelta en el formato PACKET_IN.

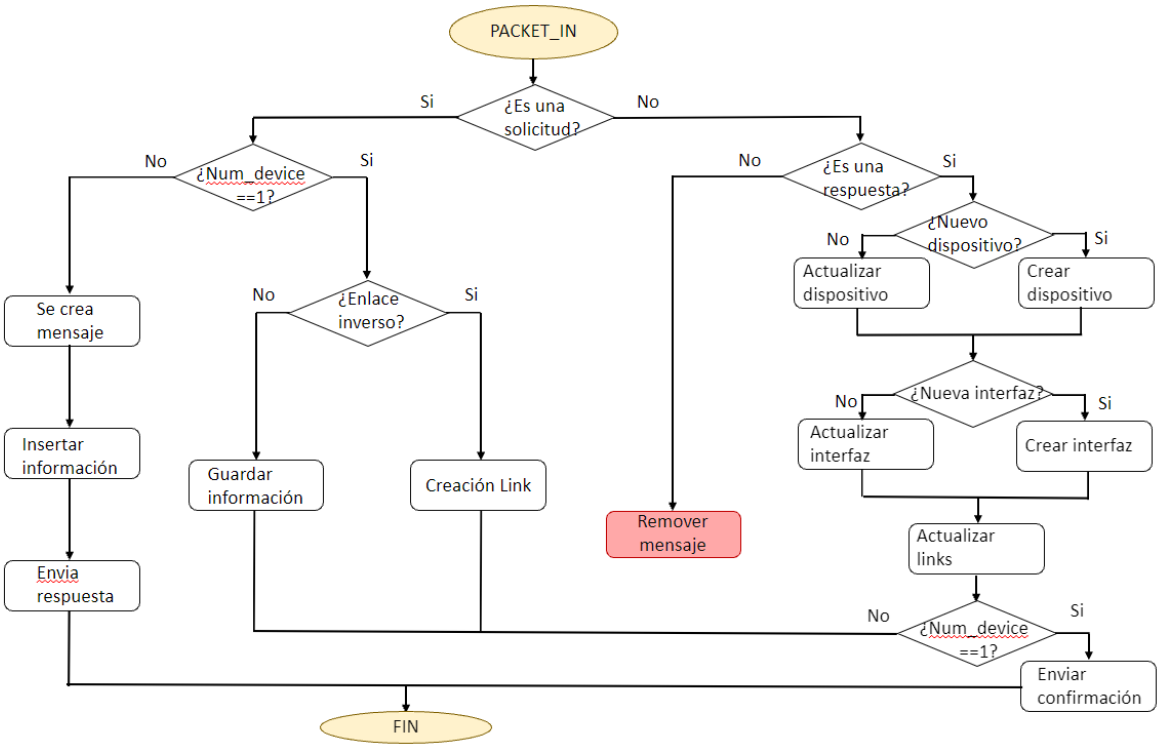


Ilustración 17: Aplicación lógica del controlador.

Una vez que el controlador ONOS ya tenga descubierta la topología de la red híbrida, se utiliza Unit Interface (GUI) de ONOS. GUI es la interfaz gráfica de ONOS que nos va a permitir tener una imagen en pantalla de la topología de red.

2.4 Mininet

Mininet [24] es una plataforma de redes que permite emular enlaces y dispositivos de red, para emular las distintas redes, Mininet utiliza software en vez de hardware.

Propiedades de Mininet:

- → Flexible: interpreta nuevas funcionalidades y topologías.
- → Escalabilidad: soporta la escalabilidad de un gran número de switches.
- → Realista: Un dispositivo simulado en Mininet, funciona muy similar a un dispositivo real.
- → Interactivo: La simulación se realiza en tiempo real, es decir, es como si se estuviera trabajando con una red real.
- → Fácil de compartir: Se pueden ejecutar pruebas que luego se pueden compartir de manera sencilla.

→ Código abierto: Los usuarios pueden corregir errores, examinar o modificar su código fuente, ya que se trata de un código abierto.

Mininet se puede utilizar en ámbitos como la investigación o la educación, debido a que en esta herramienta, se pueden emular redes complejas, es compatible con topologías personalizadas, incorpora un conjunto de comandos de diagnóstico y además tiene la capacidad de enviar una orden a un nodo de la red.

Hay disponibles herramientas similares a Mininet, por ejemplo, VM Nox o OpenFLowVMS pero Mininet proporciona una alta velocidad de arranque, mayor ancho de banda y permite mayor escalabilidad.

Capítulo 3: Desarrollo de la aplicación

3.1 Programa eHDDP

El programa utilizado en este TFG está formado por diferentes clases, debido a que la programación que se ha utilizado, es la programación orientada a objetos. Se ha tenido que utilizar este tipo de programación, debido a que ONOS está programado como un sistema que está orientado a eventos y a su vez a objetos. Se dice que está orientado a eventos porque cuando recibe un evento externo reacciona, es decir, en el caso de que el evento externo sea la llegada de un paquete, ONOS va a reaccionar respecto a la llegada de ese paquete, estos eventos hacen que ONOS distribuya los trabajos entre sus diferentes aplicaciones.

3.1.1 Módulo AppComponent

Este módulo, está compuesto por cuatro puntos importantes, la activación de la aplicación, la cual comienza con la función protected void activate, el cierre de la aplicación, definido por la función protected void deactivate. En tercer lugar, se dispone de la función ReactivePacketProcessor, que se encarga de procesar los paquetes de confirmación y de descubrimientos que han sido recibidos. Como último punto importante se tiene la función startDHTProcess que inicia el proceso de exploración del protocolo.

La función de activación lo único que hace es indicar a ONOS que elementos deben de ser activados por dicha función, entre ellos, los elementos más importantes son el DHTproviderlink donde se van a almacenar todos los enlaces que van a formar la topología, el DHTproviderdevice que se va a encargar de almacenar toda la información de los dispositivos, en nuestro caso con dispositivos hacemos referencia a los nodos que forman la topología y el DHTpacket que es el módulo que genera el paquete para poder realizar el descubrimiento de las topologías.

Una vez que se ha iniciado la aplicación y se han activado todos los providers correspondientes, comienza a ejecutarse la función startDHTprocess, la cual va a permitir que la aplicación siga ejecutándose hasta que se desactive, a su vez, se encarga de limpiar los enlaces de forma periódica para poder actualizar la topología y además se encarga de generar eHDDP request procedentes del controlador y con destino a cada uno de los nodos SDN de la topología, estas misiones se llevan a cabo de la siguiente manera: recorre todos los dispositivos que se

encuentran almacenados en ONOS y de esos dispositivos, selecciona únicamente a dispositivos OpenFlow, a estos elementos se les enviará paquetes eHDDP request para iniciar la fase de exploración. La creación de dichos paquetes se realizará mediante la función CreatePacketDHT y se enviará gracias a la función sendpacketwithDevice. A continuación, lo que realiza startDHTProcess es mantener de forma activa la aplicación.

Como se ha mencionado anteriormente, esta aplicación es reactiva, es decir reacciona a eventos y en este caso un evento es la llegada de un paquete, el manejo de dichos paquetes se lleva a cabo mediante la función ReactivePacketProcessor, este proceso se encarga, en primer lugar, de comprobar a que aplicación corresponde el paquete, mediante su DHT_ETHERNET_TYPE, si el DHT_ETHERNET_TYPE coincide con eHDDP el paquete será controlado por la aplicación implementada, en caso contrario, la aplicación no hará caso al paquete, esto sucede porque cuando ONOS recibe un paquete, ONOS lo pasa por todas las aplicaciones hasta que una de las aplicaciones confirma que ese paquete es suyo.

Una vez que se ha detectado, que un paquete es para nuestra aplicación, lo que se hace es leer los campos que conforman el paquete, como por ejemplo el option_code. En el caso de que se trate de un eHDDP request, pueden ocurrir dos cosas, en primer lugar, que el número de saltos que trae el paquete sea uno solo, es decir, que a la aplicación le llegue un paquete con un único salto, lo que significa que el nodo anterior al que ha llegado el paquete es un nodo SDN, por lo tanto, se trataría de una unión entre dos elementos SDN, es decir, si se tiene un nodo A y un nodo B y ambos nodos son SDN el nodo B también le enviará un paquete al nodo A lo que indica que se trata de un enlace bidireccional. Si no se trata de un paquete eHDDP request, se tratará de un eHDDP reply, este tipo de paquetes llevan la información topológica dentro del mismo, esa información es porque ha pasado por muchos nodos que no son elementos SDN y tienen el agente eHDDP incorporado, una vez que ese paquete ha llegado al nodo correspondiente, se actualiza al nodo.

Por lo tanto, en este caso lo que hace el sistema es leer todos los dispositivos que tienen el paquete eHDDP incorporado y comprueba que están en el sistema ONOS, si se da la circunstancia de que no están en ONOS, este los creará. En segundo lugar, el sistema lee cada de una de las interfaces por las que ha pasado el paquete, por ejemplo, el paquete pasa por la interfaz uno del nodo tres y el sistema comprueba que esa interfaz está adherida a ese nodo tres, en el caso de que esa interfaz no aparezca, el sistema creará esa API y se la asignará al nodo.

Por último, se crean los enlaces entre las interfaces creadas previamente siguiendo la información indicada en el paquete.

En resumen, el paquete hace una lista de nodos, el primer nodo será el que cree el mensaje eHDDP reply y este dispositivo sólo tendrá interfaz de salida, el siguiente nodo por el que pase eHDDP reply tendrá dos interfaces, una de ellas será la interfaz de entrada que estará conectada al nodo de salida del primer nodo y la interfaz de salida que estará conectada al siguiente nodo y este proceso se repite tantas veces como nodos que haya atravesado dicho paquete, el último nodo por el que pase el eHDDP reply sólo dispondrá de interfaz de entrada.

3.1.2 Módulo DHTpacket

El módulo llamado DHTpacket es el encargado de crear el paquete para realizar el descubrimiento de las topologías. Además, también en este módulo también se serializan los datos y se deserializan los datos que se reciben.

En DHTpacket también se disponen de funciones que son capaces de introducir datos en el buffer y de generar una dirección física válida mediante variables long.

3.1.3 Módulo DHTproviderdevices

El siguiente módulo se encarga de almacenar la información topológica, además, también se encarga de crear o de actualizar los dispositivos no SDN de la red. También comprueba que todos los dispositivos necesarios en la topología, existan y en el caso de no existir lo que hace este programa es crearlo, en el caso de que el dispositivo sea SDN, el nodo no será tratado debido a que el módulo Openflow propio de ONOS será el que gestione los elementos SDN, en el caso en que se trate de un elemento no SDN configurado o no configurado se deberá de actualizar el dispositivo, de esta misma forma se comprobará si existen todos los puertos necesarios, en caso de que no existan todos los puertos necesarios, estos serán creados. Una vez que todos los puertos y nodos han sido creados, se deben de configurar y actualizar los puertos de los dispositivos y esto será realizado por la función LinkPortDevice.

3.1.4 Módulo DHTproviderlink

Por último, se dispone del módulo DHTproviderlink, en este módulo, se van a almacenar todos los enlaces que han sido necesarios para formar la topología. En primer lugar, se dispone de la función linkbetweendevices la cual es encargada de crear enlaces entre dos dispositivos SDN, para poder llegar a dicho objetivo, se tiene que obtener la información del que va a ser el origen y el destino del enlace, y a continuación, es cuando se formarán los dos enlaces. En el caso en el que se tenga un enlace que esté formado por algún nodo SDN, se considerará que el enlace no es durable, si en los extremos del enlace se disponen de dos nodos se considerará que el enlace es durable. En último lugar, la función linkstopology se encargará de generar los enlaces definidos por el protocolo en la topología de ONOS.

3.2 Bazel

3.2.1 La herramienta Bazel

Google ha desarrollado una nueva herramienta llamada Bazel, que posibilita ser más fiables, rápidos y escalables a la hora de crear un software [30]. Bazel, fue creado para ser capaz de compilar muchos códigos fuentes basados en un único monorepos, es decir que todo el código de un programa se localice en un solo repositorio. Adicionalmente, Bazel proporciona cuatro ventajas a tener en cuenta: la primera de ellas la velocidad, Bazel es capaz de realizar distintas tareas de forma simultánea y además cuenta con un sistema de caché, lo que quiere decir que no lleva a cabo funciones innecesarias, como por ejemplo, si ya ha ejecutado unos archivos BUILD o test que no han sufrido ningún tipo de modificación, Bazel no volverá a ejecutar dichos archivos. Otra de las ventajas es que acepta que en un mismo proyecto los archivos que lo formen puedan estar programados en distintos lenguajes, como C++, Java, Python, etc. Además, Bazel tiene la habilidad de dar apoyo a los nuevos lenguajes de programación que van surgiendo, por último, es capaz de administrar información que se encuentra en uno o varios repositorios.

En Bazel es muy importante disponer de un directorio raíz llamado workspace, debido a que Bazel se organiza a través de ese directorio. El directorio debe de tener un archivo WORKSPACE en el que se introducen todas las dependencias externas necesarias para poder realizar el proyecto.

Posteriormente, se tienen los paquetes, que son la unidad primaria de organización del código en un WORKSPACE [31]. En dichos paquetes están agrupados los archivos relacionados para poder desarrollar la aplicación, para saber cómo están relacionados los archivos de un paquete se dispone del archivo BUILD. Los archivos BUILD, están formados por un número de reglas en lenguaje imperativo, que establecen como se debe de construir el paquete al que pertenecen, es decir, en estos ficheros se declaran las reglas, los ejecutables y los test necesarios para poder formar el paquete. La mayoría de las reglas están formadas por deps, es decir, las dependencias que posee el código, también por srcs que indica el código fuente y otras particularidades como origin, visibility, resources, etc. Además de los ficheros BUILD, también se disponen de archivos .bzl en los cuales también se definen reglas y se pueden importar mediante el comando load en el WORKSPACE.

3.2.2 Configuración de Bazel

El entorno que se ha utilizado para instalar y formar el proyecto Bazel, es el programa Intellij, en este caso se ha utilizado la versión 2020.2.4 debido, a que no todas las versiones de Intellij disponían de la capacidad de realizar un proyecto Bazel,

A continuación, se muestra cómo es el entorno de Intellij y de cómo hay que proceder para poder disponer de Bazel.

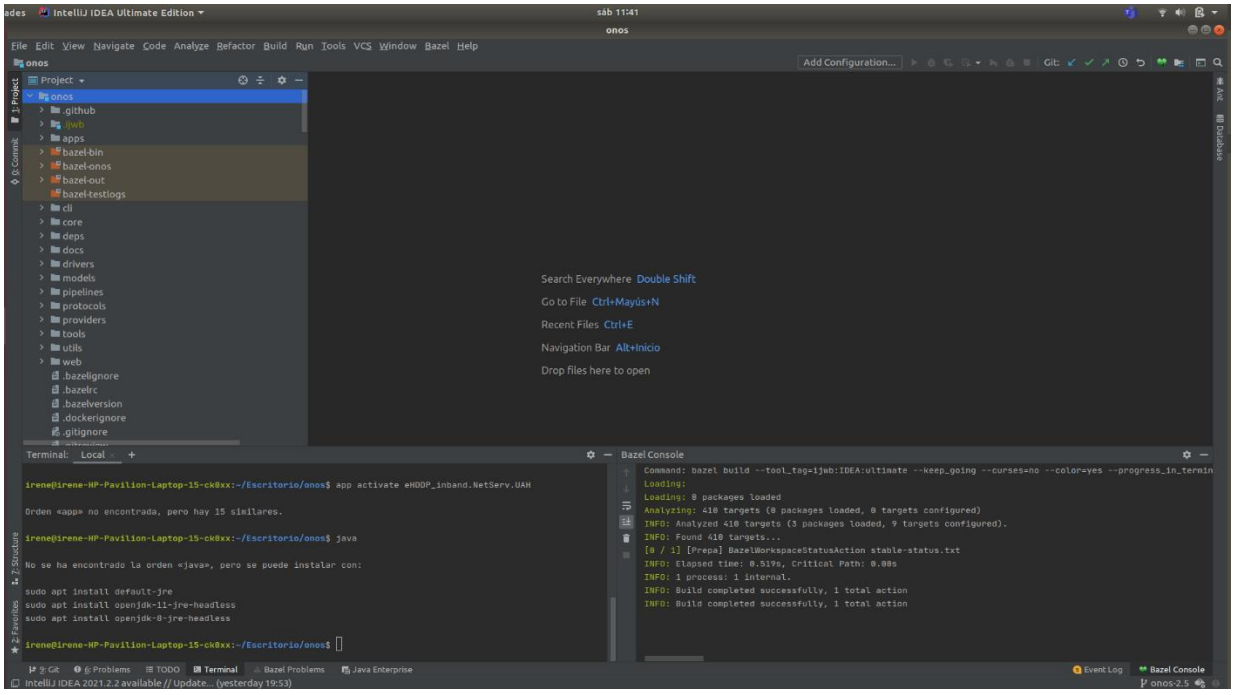


Ilustración 18:Entorni Intellij

Para configurar Bazel en Intellij se debe de elegir la opción File y a continuación se selecciona Settings. Una vez hemos llevado a cabo los pasos anteriores, se mostrará la ventana Settings, en ella se debe de elegir Plugins, y dentro de Plugins se disponen de dos opciones: Marketplace, en donde aparecen todas las aplicaciones disponibles que se pueden instalar en Intellij y la segunda opción, Installed en donde se muestra todo lo que se tiene instalado en Intellij. Se selecciona la opción Marketplace y en el buscador se escribe la aplicación que se desea, en este caso Bazel. Una vez que se hayan realizado los pasos mencionados correctamente, ya se dispondrá de Bazel.

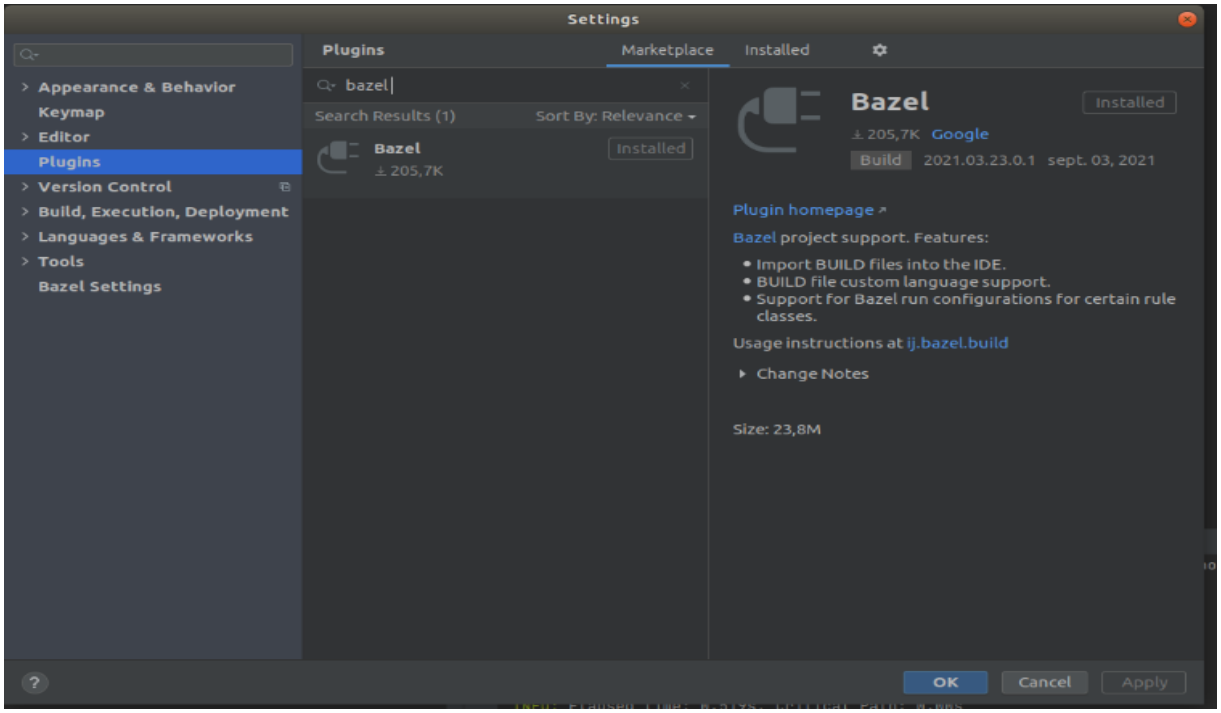


Ilustración 19: Instalar Bazel

Además de configurar Intellij con Bazel, también es necesario instalar Bazel en Ubuntu, para ello se debe de abrir un terminal e introducir el siguiente comando:

```
irene@irene-HP-Pavilion-Laptop-15-ck0xx:~$ sudo apt update && sudo apt install bazel
```

Ilustración 20: Instalación Bazel en Ubuntu

Una vez que se ha instalado Bazel se puede actualizar a la última versión disponible, es decir la versión más nueva. Dicha versión se puede conseguir mediante el comando que se muestra a continuación:

```
irene@irene-HP-Pavilion-Laptop-15-ck0xx:~$ sudo apt update && sudo apt full-upgrade
```

Ilustración 21: Obtener última versión Bazel

En el caso de este proyecto, no se deseaba tener instalada la última versión de Bazel, si no que se deseaba la versión 3.7.2. En el caso de que se quiera una versión específica, se puede conseguir mediante la siguiente línea de comando:

```
irene@irene-HP-Pavilion-Laptop-15-ck0xx:~$ sudo apt install bazel-3.7.2
```

Ilustración 22:Obtener una versión Bazel específica

3.2.3 Creación de un proyecto en Bazel

Para crear un proyecto Bazel se debe de ir a la opción File y elegir la opción Import Project from Bazel y se debe de indicar el directorio del proyecto que se desea que sea un proyecto bazel, en el caso de este proyecto, el directorio es el siguiente: /home/irene/Escritorio/onos.

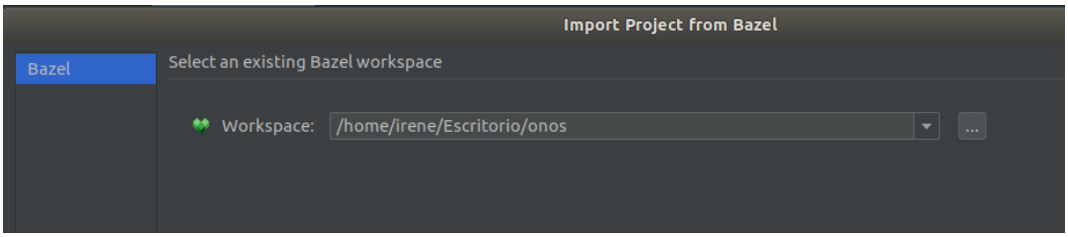


Ilustración 23: Import Project from Bazel

A continuación, se mostrará una pantalla en la que se da a elegir que Proyect view se quiere, es decir, que vista del proyecto se desea. En este caso, se va a elegir la opción: Generate from BUILD file, se elige dicha opción, debido a que cuando se han descargado los archivos ONOS para poder desarrollar la aplicación, se dispone de un archivo BUILD, el cual es el archivo principal de todo el proyecto.

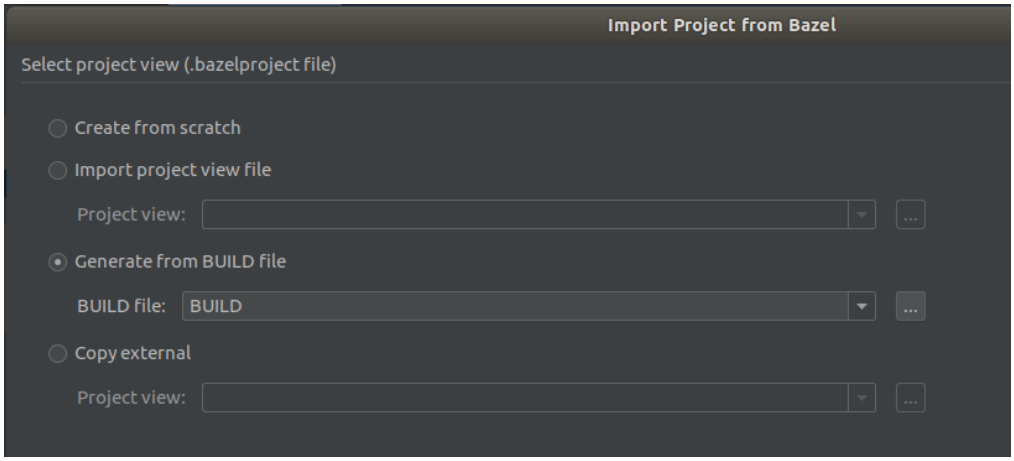


Ilustración 24: Generate from BUILD file.

El siguiente paso es generar el archivo .bazelproject donde aparecen los directorios, objetivos y lenguajes adicionales para poder soportar la aplicación. Este archivo se autogenera una vez que se ejecuta el archivo onos-gen-bazel-project mediante el siguiente comando:

```
irene@irene-HP-Pavilion-Laptop-15-ck0xx:~/Escritorio/onos/tools/dev/bin$ ./onos-gen-bazel-project > /tmp/onos_bazelproject
```

Ilustración 25: Generar archivo .bazelproject

Una vez que se tiene dicho archivo se copia y se pega en el Project View, a continuación, se le pulsa a terminar y el proyecto Bazel será creado.

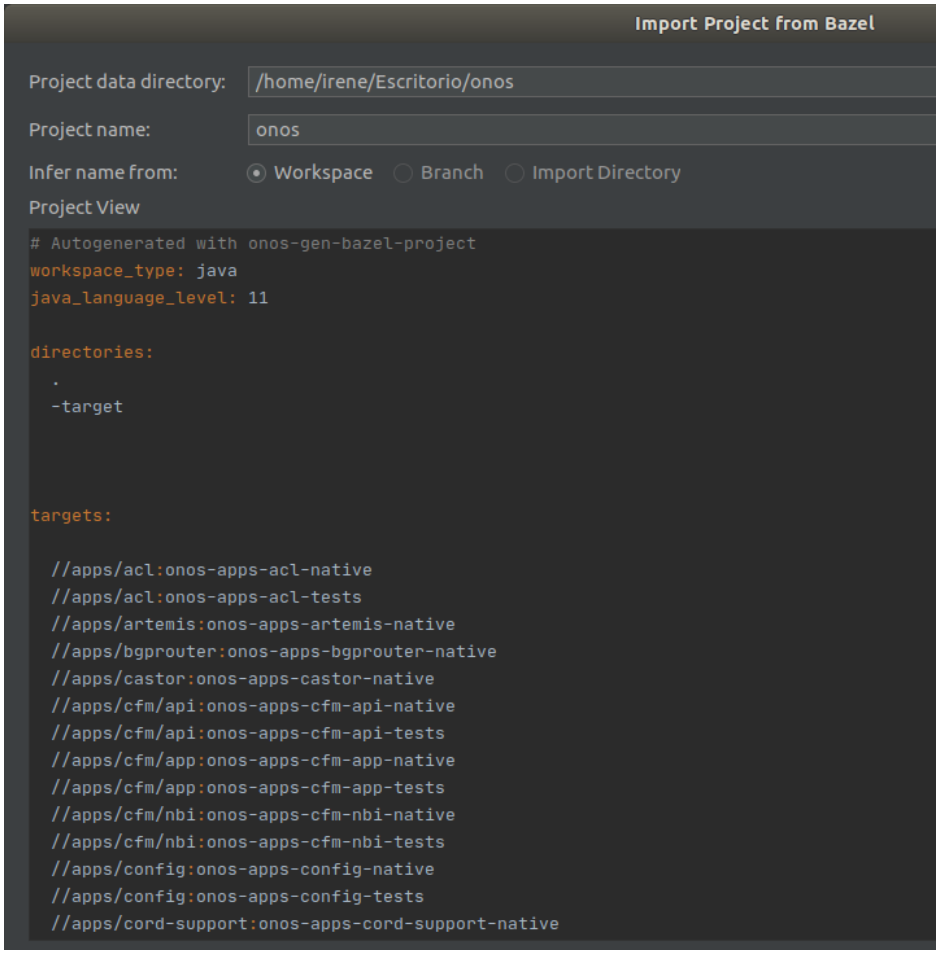


Ilustración 26: Directorios, objetivos y lenguajes adicionales.

3.2.4 Compilación de una aplicación en Bazel

Para compilar el proyecto, el primer paso que se debe de realizar el sincronizar los archivos BUILD, para realizar dicha sincronización se debe de ejecutar Bazel > Sync > Sync Project with BUILD files, si en la consola Bazel, no aparece ningún error, se continúa compilando el proyecto Bazel > Build > Compile Project.

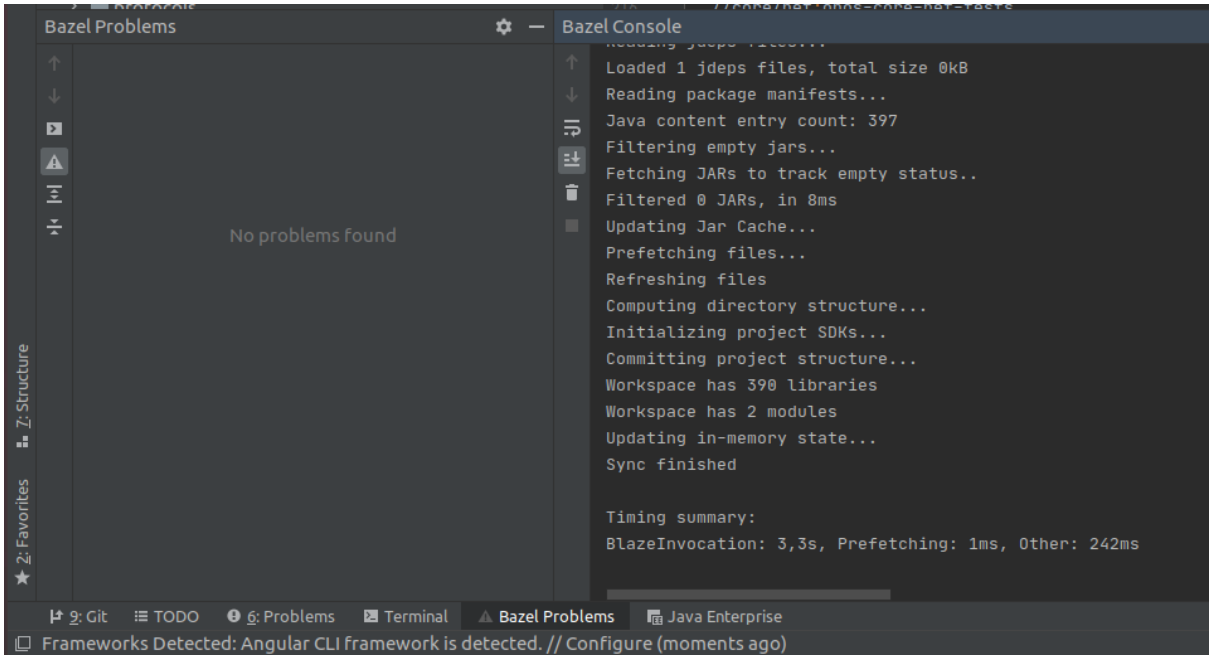


Ilustración 27: Sincronización archivos BUILD.

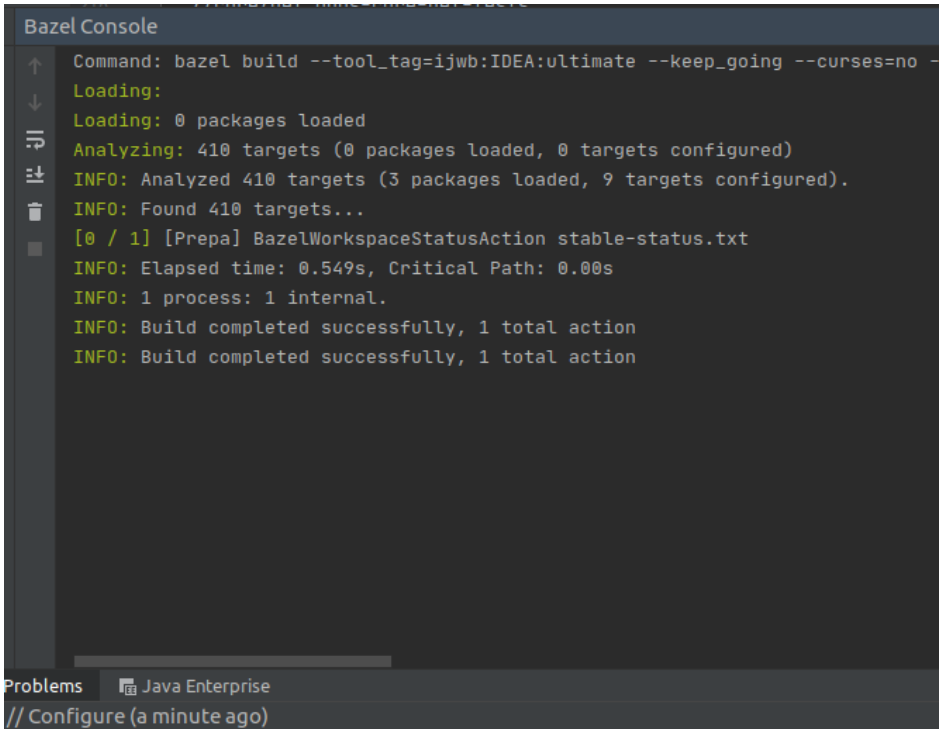


Ilustración 28: Compilación proyecto Bazel

3.2.5 Estructura de la aplicación

Una vez que se tiene la carpeta ONOS con los archivos correspondientes y que se ha creado el proyecto Bazel se procede a introducir los archivos proporcionados por los tutores de este

proyecto, estos archivos se han introducido en una carpeta llamada *aplicacion* y dicha carpeta se ha introducido dentro de la carpeta apps de ONOS. Para que la aplicación funcione correctamente, es muy importante que la estructura de los archivos sea la correcta. Para saber cómo debían de estar organizados los archivos de *aplicacion*, se tomó como referencia la carpeta fwd y se observó que dentro de la carpeta *aplicacion* debía de haber una carpeta src y un archivo BUILD, en el cual se definen una serie de reglas de compilación. Dentro de la carpeta src se encuentra otra carpeta llamada main y en el interior de la carpera main se encuentra la carpeta java y así sucesivamente hasta la última carpeta llamada *aplicacion*, donde se encuentran todos los archivos.

En la ilustración 29, se puede ver de forma más detallada cómo quedó la estructura de la aplicación:

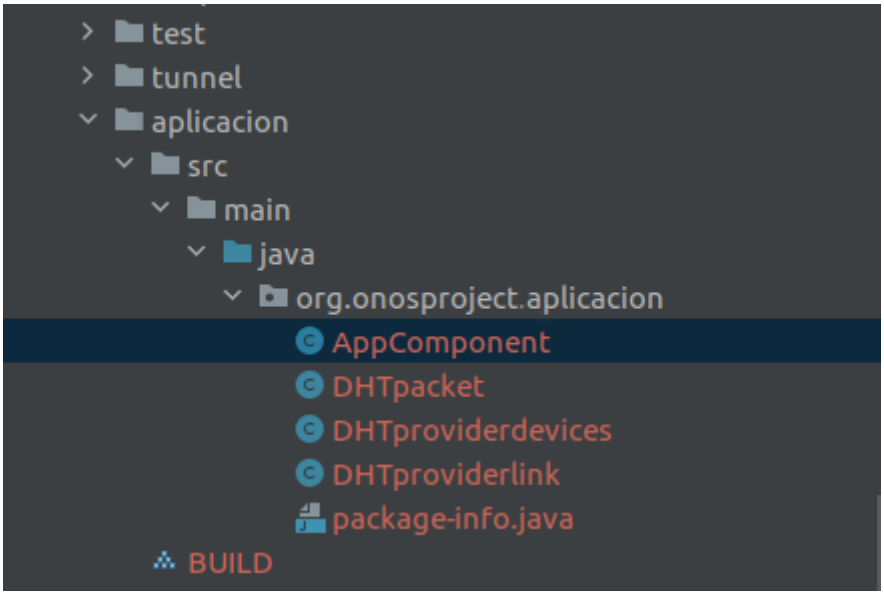


Ilustración 29: Estructura carpeta aplicacion

Una vez se han introducido todos los archivos y la aplicación tiene la estructura adecuada se procede a sincronizar el programa y migrar la aplicación de Maven a Bazel.

3.2.6 Migración de una aplicación basada en Maven a Bazel

Una vez que se ha sincronizado el programa, se pueden apreciar los errores que hay en el código y que son necesarios corregir para poder crear la aplicación basada en Bazel correctamente.

En primer lugar, se deben de modificar las librerías que venían en los módulos de la carpeta *aplicación,* puesto que la aplicación original era un proyecto Maven y ahora se trata de un

proyecto Bazel. Para saber que librerías eran las que se debían modificar, el programa Intellij las señalaba de color rojo y gris y aquellas que no se utilizaban, las señalaba únicamente de color gris, tal y como se muestra en la ilustración 30.

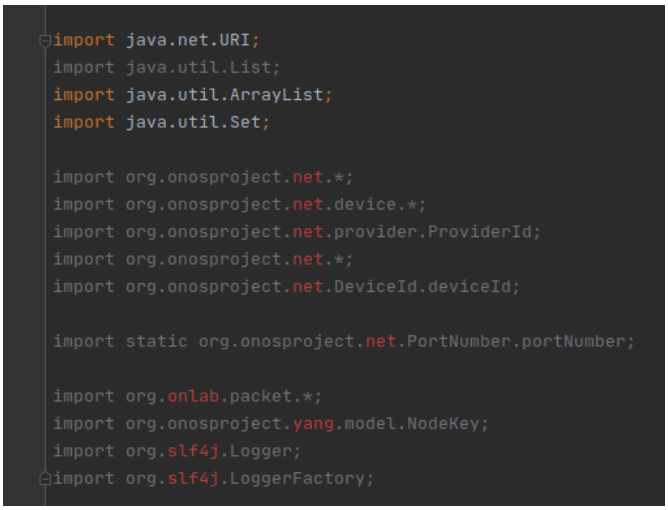


Ilustración 30: Ejemplo de librerías erróneas y no utilizadas.

A continuación, se procedió a eliminar aquellas librerías que estaban solamente en color gris, debido a que no se utilizaban y así dejar el programa más limpio y ordenado. En cuanto a las librerías de color rojo y gris, son aquellas que se debían corregir para que el código funcionara correctamente, en el caso de que no se corrigieran en el programa, aparecen demasiados errores y no se crea el proyecto. Para corregir dichas librerías, se buscaban todas las funciones de color rojo que aparecían en el programa, porque ese color indicaba que esa función no disponía de una librería. Cuando se localiza una función de color rojo, se coloca el ratón encima, clicas y afortunadamente Inteliij reconocía qué librería era la correcta para dicha función, se elegía la librería recomendada por Intellij y automáticamente el programa te importa la librería. En la ilustración 31 se puede observar un ejemplo de funciones que no disponen de librerías y en la ilustración 32, se puede observar cómo Intellij reconoce que librería es la necesaria para que la función correspondiente funcione.

```
if(hw.contains("sw")){
    uri = URI.create("sw:"+str_to_id(id));
    /** Creamos el dispositivo */
    desc = new DefaultDeviceDescription(uri, type, manufacturer, hw, sw, serial, cid,
            true, DefaultAnnotations.builder().set("Switch", "Legacy Switch")
            .set(AnnotationKeys.PROTOCOL, "No Have").set(AnnotationKeys.USERNAME, "GateWay")
            .set(AnnotationKeys.MANAGEMENT_ADDRESS, "1.0.0.1").set(AnnotationKeys.NAME, "GateWay")
            .build());
}
else if(hw.contains("sdn")){
    uri = URI.create("of:"+str_to_id(id));
    // Creamos el dispositivo
    desc = new DefaultDeviceDescription(uri, type, "Stanford University, Ericsson Research and CPqD Research",
            "OpenFlow 1.3 Reference Userspace Switch","eHDDP", serial, cid,
            true, DefaultAnnotations.builder().set("Switch", "SDN/ehddp Switch")
            .set(AnnotationKeys.PROTOCOL, "OF_13").set(AnnotationKeys.USERNAME, uri.toString())
            .set(AnnotationKeys.MANAGEMENT_ADDRESS, "127.0.0.1").set(AnnotationKeys.NAME, uri.toString())
            .build());
}
```

Ilustración 31: Funciones sin librerías.

```
/** Atributos */
org.slf4j.Logger? Alt+Intro
private final Logger log = LoggerFactory.getLogger(getClass());
private static final long DEF_BANDWIDTH = -1L;
private static final double DEF_METRIC = -1;
private static final Duration DEF_DURATION = Duration.ofNanos(-1L);
```

Ilustración 32: Intellij reconociendo las librerías.

Las nuevas librerías introducidas para que el proyecto funcionara son las siguientes:

→ org.onosproject.net.link.LinkProvider;
→ org.slf4j.Logger;
→ org.slf4j.LoggerFactory;
→ org.onosproject.net.link.LinkProviderService;
→ org.onosproject.net.link.LinkDescription;
→ java.util.Set;
→ org.onosproject.net.PortNumber.*portNumber*;
→ org.onosproject.net.*;
→ org.onosproject.net.link.DefaultLinkDescription;
→ org.onosproject.net.DeviceId.*deviceId*;

Además, hay ciertos comandos que se han tenido que modificar, debido a que no se llaman igual cuando la aplicación está montada sobre un proyecto Maven que cuando está montada sobre un proyecto Bazel. La mayoría de los comandos se corrigieron cuando se modificaron las librerías, pero el comando MANDATORY_UNARY siguió dando error, para corregirlo se buscó a que clase pertenecía, en este caso a la clase ReferenceCardinality.class y se observó que el nombre que utiliza Bazel para referirse a dicho comando es MANDATORY.

Otro de los cambios que se tuvo que realizar en el código, para que la aplicación funcionara, es cuando se definía la clase de los enlaces y los dispositivos. Cuando la aplicación estaba basada en un proyecto Maven, los PID utilizados para cada clase se llamaban igual, no era necesario hacer una distinción entre ellos, pero en el caso del proyecto Bazel, sí que fue necesario hacer esa diferencia entre el PID de los enlaces y de los dispositivos, porque en el caso de no hacer esta distinción no se activaba la aplicación, se mostraba un error que indicaba que la aplicación no había podido ser registrada.

En la ilustración 33, se muestra cómo están definidas las clases en Maven y en la imagen 34, se muestra cómo se han definido las clases en Bazel:

```
/**Clase para manejar los enlaces */
private DHTproviderlink DHTlink = new DHTproviderlink(PID);
/**Clase para manejar los devices */
private DHTproviderdevices DHTdevices = new DHTproviderdevices(PID);
```

Ilustración 33: Definición de clases en Maven

```
/**Clase para manejar los enlaces */
private DHTproviderlink DHTlink = new DHTproviderlink(PID_LINK);
/**Clase para manejar los devices */
private DHTproviderdevices DHTdevices = new DHTproviderdevices(PID_DEV);
```

Ilustración 34: Definición de clases en Bazel

Además, la función que activaba la aplicación también tuvo que ser modificada debido a que en ella se utilizan los PID modificados anteriormente:

```
protected void activate() {
    try{

        appId = coreService.registerApplication("org.onosproject.aplicacion");
        PID_LINK = new ProviderId("cfg", "org.onosproject.aplicacion.link");
        PID_DEV = new ProviderId("cfg", "org.onosproject.aplicacion.device");
        packetService.addProcessor(processor, PacketProcessor.advisor(2)); //.director(2));
```

Ilustración 35: Función de activación de la aplicación.

A continuación, se procedió a completar el archivo BUILD de nuestra aplicación. Este tipo de archivos son explicados de manera más detallada en el apartado 3.3.1. Para elegir la estructura del archivo BUILD se cogió como referencia el BUILD de otras carpetas que tuvieran una estructura similar a la de nuestra aplicación, en este caso, se volvió a elegir como referencia la

carpeta fwd. Por lo tanto, el archivo BUILD tiene la estructura que se muestra en la ilustración 36.

```
COMPILE_DEPS = CORE_DEPS + KRYO + CLI + [
    "//core/store/serializers:onos-core-serializers",
    "//core/store/primitives:onos-core-primitives",
    "//apps/fwd:onos-apps-fwd-native",
    "//protocols/openflow/api:onos-protocols-openflow-api-native",
    "//protocols/openflow/api:onos-protocols-openflow-api-tests",
    "//protocols/openflow/ctl:onos-protocols-openflow-ctl-native",
    "//protocols/openflow/ctl:onos-protocols-openflow-ctl-tests",
]

osgi_jar_with_tests(
    karaf_command_packages = ["org.onosproject.aplicacion"],
    deps = COMPILE_DEPS,
)

onos_app(
    app_name = "org.onosproject.aplicacion",
    category = "Provider",
    description = "Prtocol HDDP",
    required_apps = [
        "org.onosproject.openflow-base",
        "org.onosproject.hostprovider",
        "org.onosproject.netcfglinksprovider",
    ],
    title = "Aplicacion",
    url = "http://onosproject.org",
)
```

Ilustración 36: Archivos BUILD.

En primer lugar, para que los distintos módulos que se encuentran en la carpeta *aplicacion* se puedan ejecutar, se tienen que indicar todas las dependencias que dichos archivos tienen sobre otros archivos que se encuentren en la aplicación, en este caso todas esas dependencias están introducidas en la regla COMPILE_DEPS. En segundo lugar, se encuentra la regla osgi_jar, donde aparece como ha sido definido el paquete de nuestra aplicación y por último, se define la última regla denominada onos_app, en ella se define el nombre de la aplicación, la categoría a la que pertenecen, una descripción en donde se explique que función realiza la aplicación y el título de la aplicación.

Posteriormente, se debe de modificar el archivo .bazelproject que fue necesario para crear el proyecto Bazel. En dicho archivo, se almacenan los targets, es decir, la dirección de todas las carpetas que forman la aplicación, como ahora se ha introducido la carpeta *aplicacion*, es necesario añadir su dirección en dicho archivo. Para saber cómo se debía de escribir la dirección de la carpeta introducida se volvió a tomar como referencia la carpeta fwd.

La carpeta principal donde se encuentra *aplicacion* es apps de ahí que el principio de la línea sea //apps/aplicacion y a continuación aparece toda la dirección desde la carpeta donde se recopilan todos los archivos necesarios para ejecutar la aplicación: onos-apps-aplicacion-native. “Native” indica que hay una carpeta main dentro de *aplicacion*.

```
//apps/fwd:onos-apps-fwd-native
//apps/aplicacion:onos-apps-aplicacion-native
```

Ilustración 37: Target de la carpeta fwd y aplicacion.

Una vez que se han realizado todos los pasos mencionados, se sincronizó el programa, se compiló y no se produjeron más errores. A continuación, se procedió a crea el archivo .oar, necesario para introducir la carpeta *aplicacion* en el servidor local de ONOS. Para ello, se tuvo que modificar el archivo modules.bzl y se introdujo la línea "//apps/aplicacion:onos-apps-aplicacion-oar": [], dentro de la regla APP_MAP. Dicho archivo se encuentra dentro de onos > tools >build > bazel > modules.bzl.

Ya que se dispone del archivo .oar creado y que se tienen sincronizados los archivos y compilado el proyecto, se abre un terminal y se introducen los comandos necesarios para abrir de manera local ONOS, dichos comandos están especificados en el anexo número 3. Cuando ONOS esté abierto, se puede observar que hay algunas aplicaciones que están activadas y otras que no, como por ejemplo, nuestra aplicación, que se llama Aplicación. Para poder activar la aplicación, en primer lugar, se debe de localizar, para encontrarla de manera rápida y efectiva se debe de usar el buscador, dicho buscador te permite filtrar por Título, ID de aplicación, versión, categoría u origen. Una vez que se encuentra la aplicación, se selecciona y se abre una pequeña pestaña con los datos de la aplicación y encima de dicha pestaña hay un botón con el símbolo de play que es el que hay que seleccionar en caso de querer activar la aplicación. Cuando la aplicación haya sido activada, en el terminal, pueden ocurrir dos sucesos, el primero de ellos, es que en el terminal aparezcan errores, lo que indica que hay algún error en el proceso de activación de la aplicación y el segundo suceso, es que todo haya funcionado como debe y que el terminal indique, que la aplicación está preparada para ser usada.

En caso de querer desactivar la aplicación se debe de proceder de la misma manera, pero en vez de seleccionar play se selecciona stop, el cual se corresponde con un símbolo cuadrado. Cuando la aplicación haya sido desactivada, en el terminal, pueden aparecer dos líneas diferentes, una de ellas te indica si ha habido errores a la hora de desactivar la aplicación o te indica si esta ha sido desactivada correctamente.

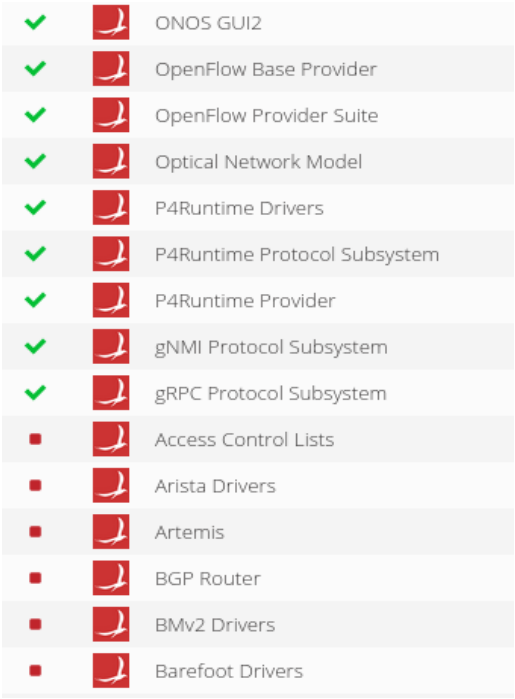


Ilustración 38: Ejemplo de aplicaciones activadas y desactivadas.

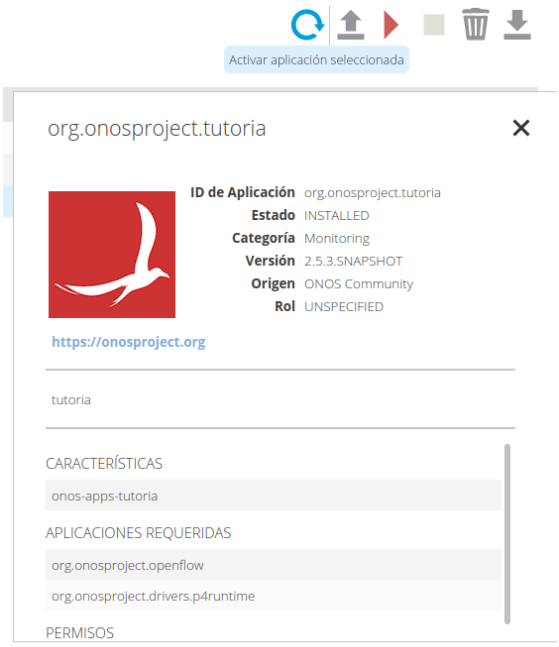


Ilustración 39: Activación de aplicación

3.2.7 Depuración de la aplicación

En el caso en el que la aplicación no esté realizando su trabajo correctamente, se debe de depurar el código, es decir, identificar y corregir los posibles errores de programación que haya en el código, para que la aplicación funcione correctamente.

En primer lugar, para poder configurar ONOS con Intellij, en necesario elegir la opción Run y después la opción Edit configurations.

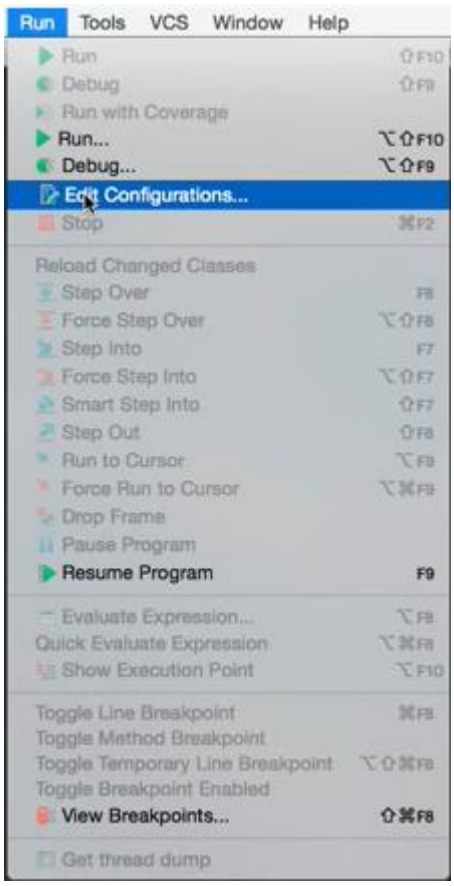


Ilustración 40: Edit configurations

En segundo lugar, aparecerá la pestaña Run/Debug Configurations, en ella se debe de seleccionar el botón Add New Configurations, una vez seleccionado, aparecerá una lista de configuraciones a elegir, en este caso se seleccionará la configuración Remote. A continuación, aparecerá la ventana que se muestra en la ilustración 41, en ella, se puede establecer el nombre de la configuración que se está realizando y en la pestaña Settings, se dejará el host y el puerto que vengan por defecto. Una vez que se han realizado dichos pasos, se selecciona aceptar para

aplicar los cambios. Una vez terminado dicho proceso se puede comenzar a depurar la aplicación.

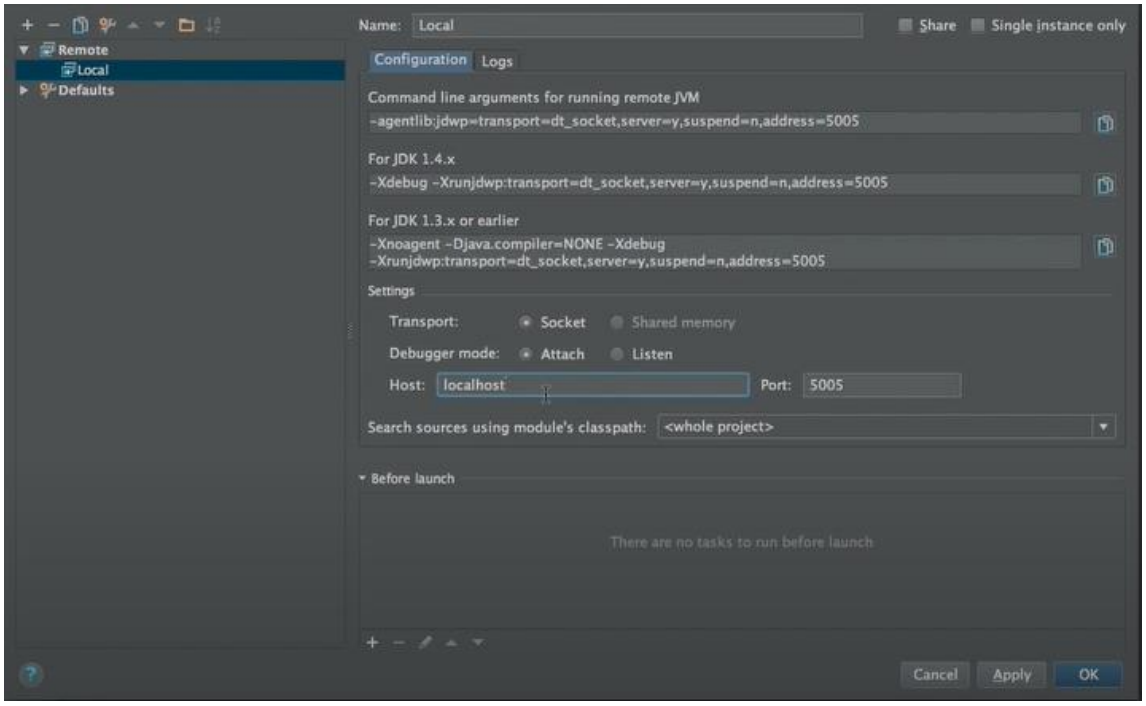


Ilustración 41: Run/Debug configurations

Para empezar a depurar, se deben de establecer puntos de ruptura, también conocidos como puntos de interrupción, que sirven para detener la ejecución del programa e ir comprobando línea a línea lo que está sucediendo en el código. Cuando ya se han establecido los puntos de ruptura que se crean convenientes, se procede a ejecutar ONOS mediante el terminal, dicho proceso está explicado en el anexo 3 de este proyecto. Posteriormente, se selecciona la opción de depuración y se debe proceder a activar la aplicación que corresponde al código que se está depurando, por último, se debe de lanzar una topología a través del terminal de Mininet, este proceso está explicado en el anexo 4. Una vez realizados dichos pasos, se podrá proceder a estudiar paso por paso la aplicación y a identificar y corregir los errores.

Capítulo 4: Resultados

Una vez que se han explicado todas las modificaciones y pasos llevados a cabo para poder implementar la aplicación, se procedió a realizar varias pruebas para comprobar si la implementación se había realizado correctamente. A continuación, se van a mostrar las distintas topologías, a las cuales se les implantó la aplicación para comprobar que esta funciona correctamente.

Topología lineal

En la ilustración 42, se puede observar que una de las pruebas se realizó sobre una topología lineal compuesta por 6 dispositivos.

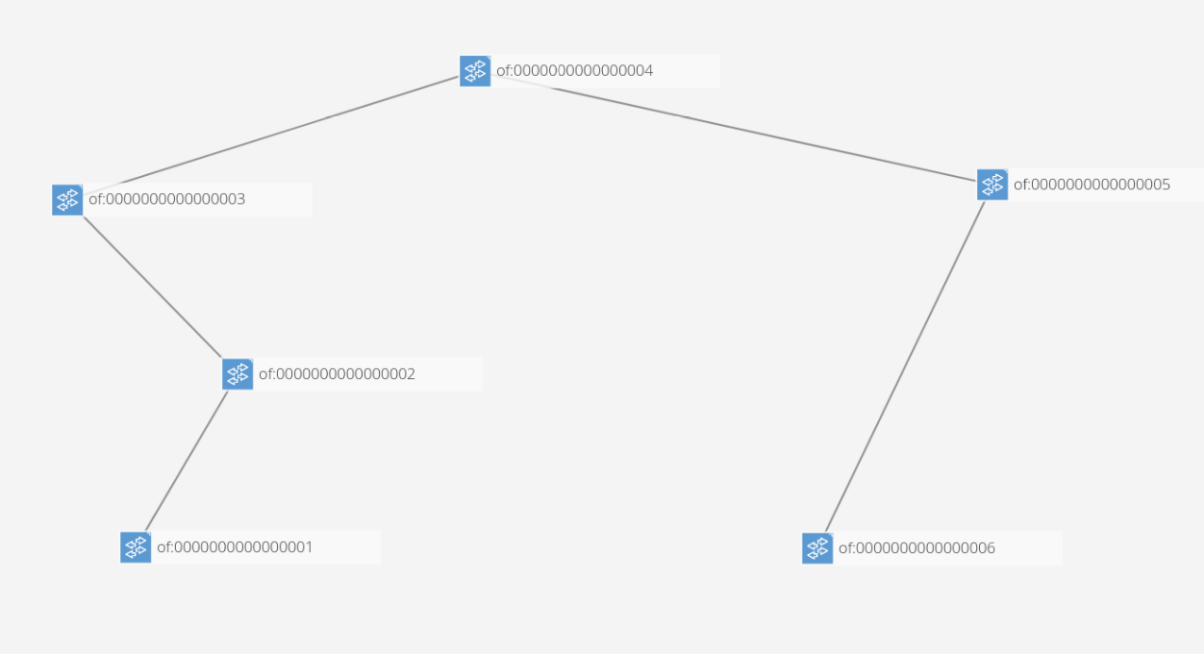


Ilustración 42: Topología lineal.

Para saber si el protocolo estaba funcionando correctamente se procedió a utilizar Wireshark, una herramienta usada para poder rastrear lo paquetes que están viajando en la red.

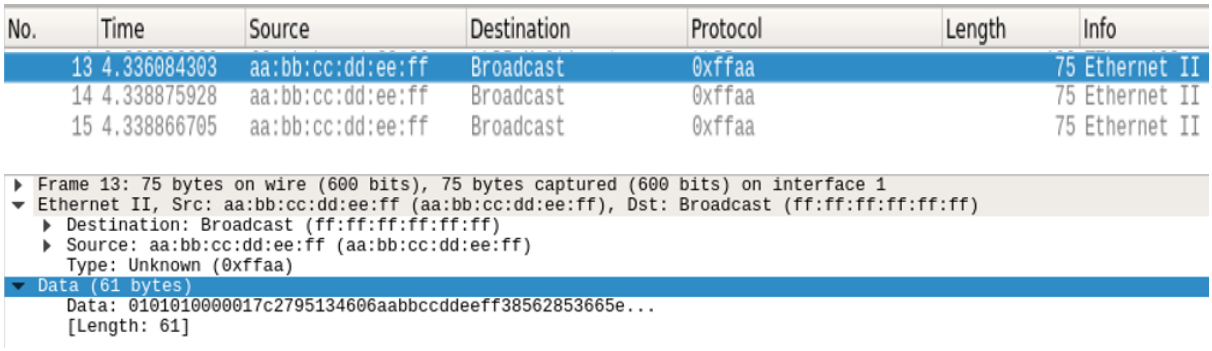


Ilustración 43: Wireshark eHDDP request.

El proceso de descubrimiento se inicia cuando el controlador ONOS, empieza a enviar paquetes de control a los nodos SDN y cuando dichos paquetes son difundidos por los nodos SDN que forman la red. En la imagen 43, se puede observar cómo un nodo ha recibido el paquete eHDDP request, a través de uno de sus puertos, una vez que el paquete ha recopilado la información necesaria, dicho nodo reenviará el paquete a otro nodo para poder continuar recopilando información de la topología.

Topología árbol

La siguiente topología que se decidió documentar fue la topología árbol, compuesta en este caso por 7 dispositivos. En la siguiente imagen se puede observar su representación.

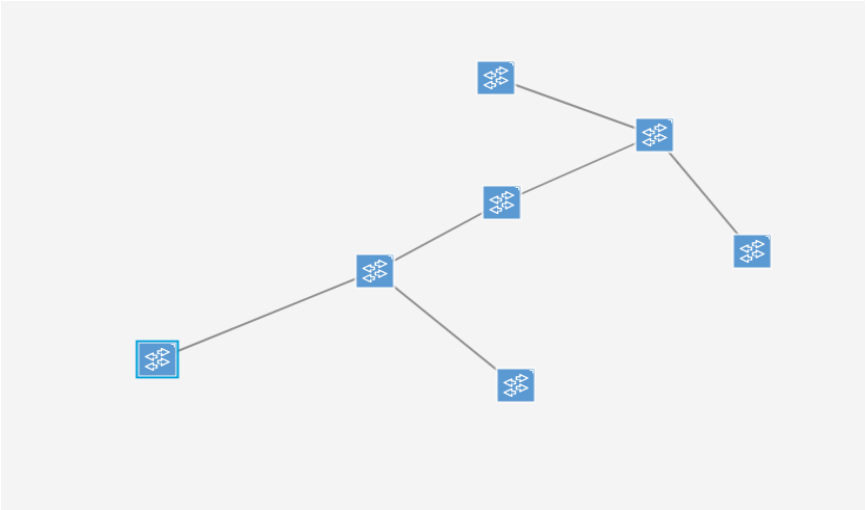


Ilustración 44: Topología árbol.

Para comprobar que la aplicación estaba funcionando como debía, se volvió a utilizar la misma herramienta que en el caso de la topología lineal, es decir la aplicación Wireshark. En este caso se obtuvieron los mismos resultados obtenidos en el caso de la topología anterior, es decir que el paquete eHDDP request se había creado adecuadamente y estaba realizando su función correctamente. A continuación, en la imagen 45, se puede observar el paquete eHDDP request de dicha topología.

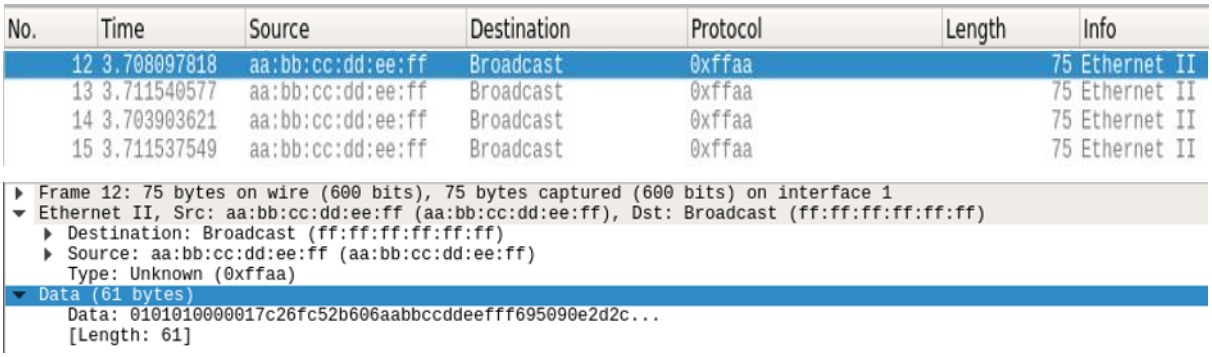


Ilustración 45: Wireshark eHDDP request

Topología triangular

La topología triangular que se muestra en la imagen 46, es una topología básica compuesta por tres switches y seis enlaces. Como se puede observar, ONOS representa de forma correcta dicha topología, lo que quiere decir que la implementación de la aplicación funciona correctamente.

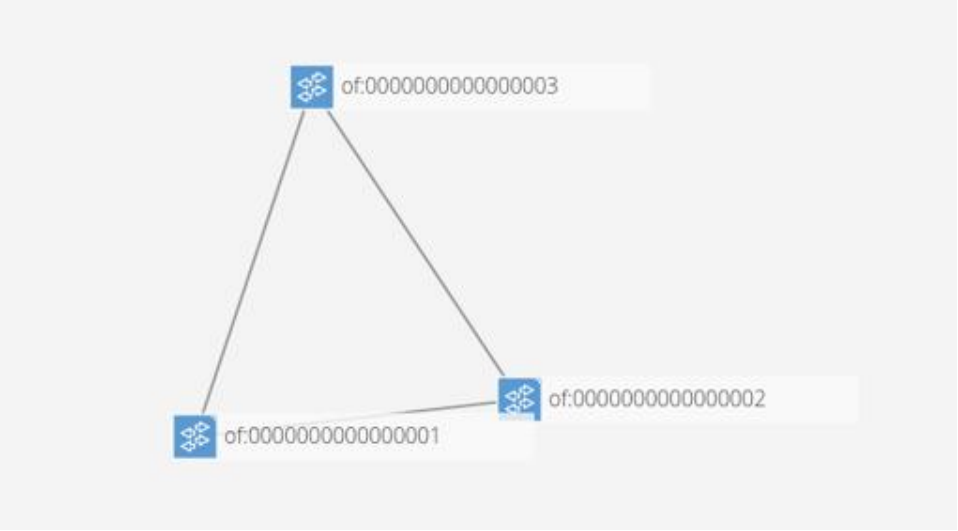


Ilustración 46: Topología triangular

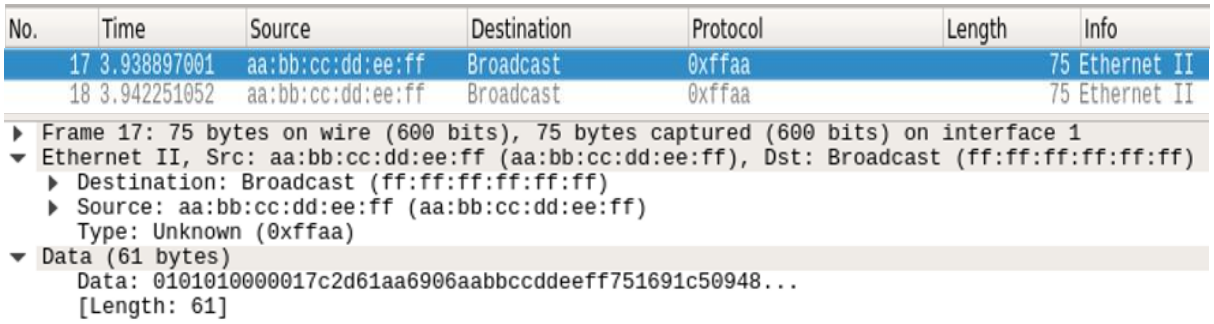


Ilustración 47:Wireshark eHDDP request.

Topología en anillo

Otra de las topologías que se utilizó para poner a prueba la aplicación fue la topología en anillo, también conocida como topología circular. En este tipo de estructura de red, cada dispositivo está conectado al siguiente dispositivo y el primero está conectado con el último. Para que esta topología funcione correctamente en necesario que todos los nodos que forman la red tengan, como mínimo, dos interfaces.

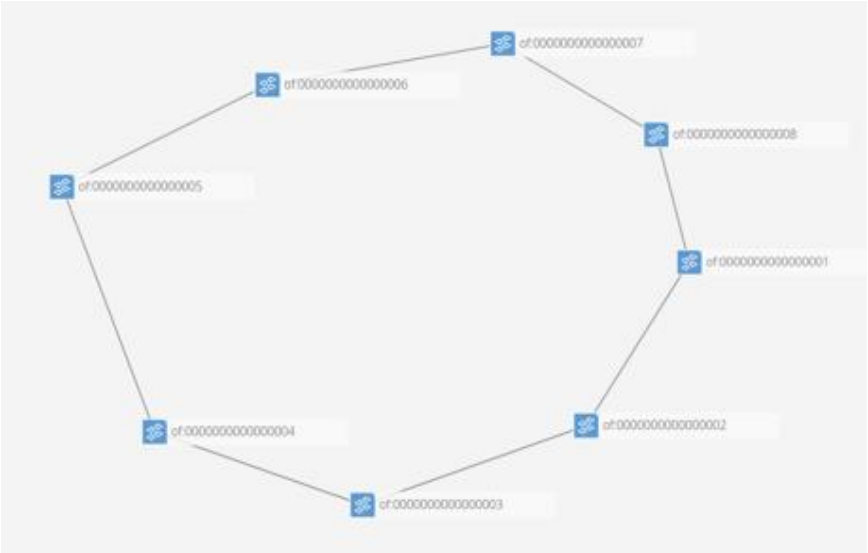


Ilustración 48: Topología en anillo.

Topología diamante

La siguiente topología que se presenta en la ilustración 49, está compuesta por cuatro nodos y es conocida como la topología diamante. En dicha topología se puede observar, al igual que en el caso anterior, que todos los nodos están conectados a sus nodos adyacentes y que el último nodo está unido al primero que forma la estructura de red.

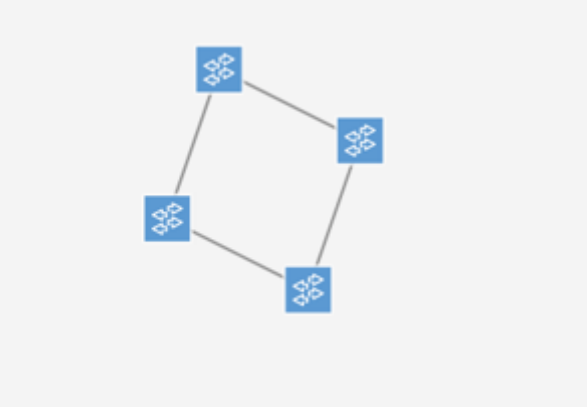


Ilustración 49: Topología diamante.

Topología torus

En este caso se puede observar una topología torus compuesta por 16 dispositivos y 64 enlaces. En la siguiente imagen, se puede comprobar cómo cada dispositivo dispone de 4 enlaces, para poder conectarse con otros dispositivos de la red, es decir un dispositivo está conectado a 4 elementos de la red.

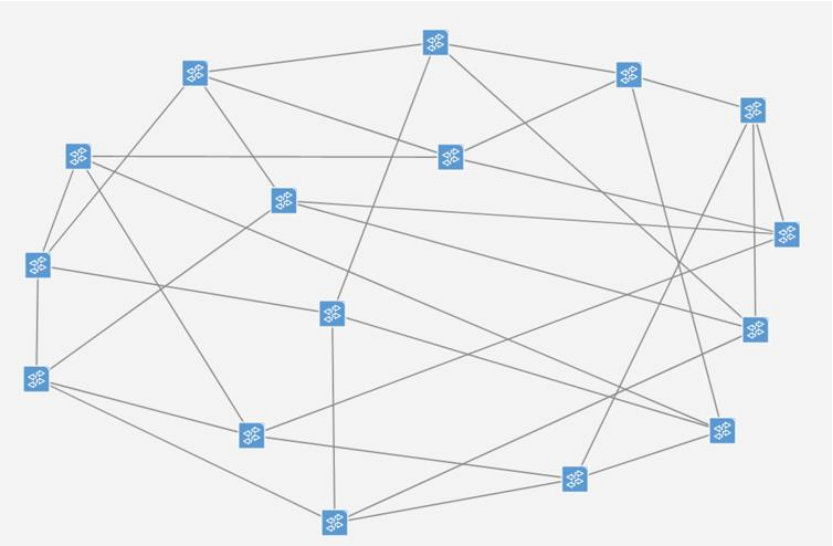


Ilustración 50: Topología Torus.

Topología híbrida

A continuación, se muestra la topología híbrida, en dicha estructura de red se realizó la última prueba que se llevó a cabo para comprobar si la aplicación funcionaba.

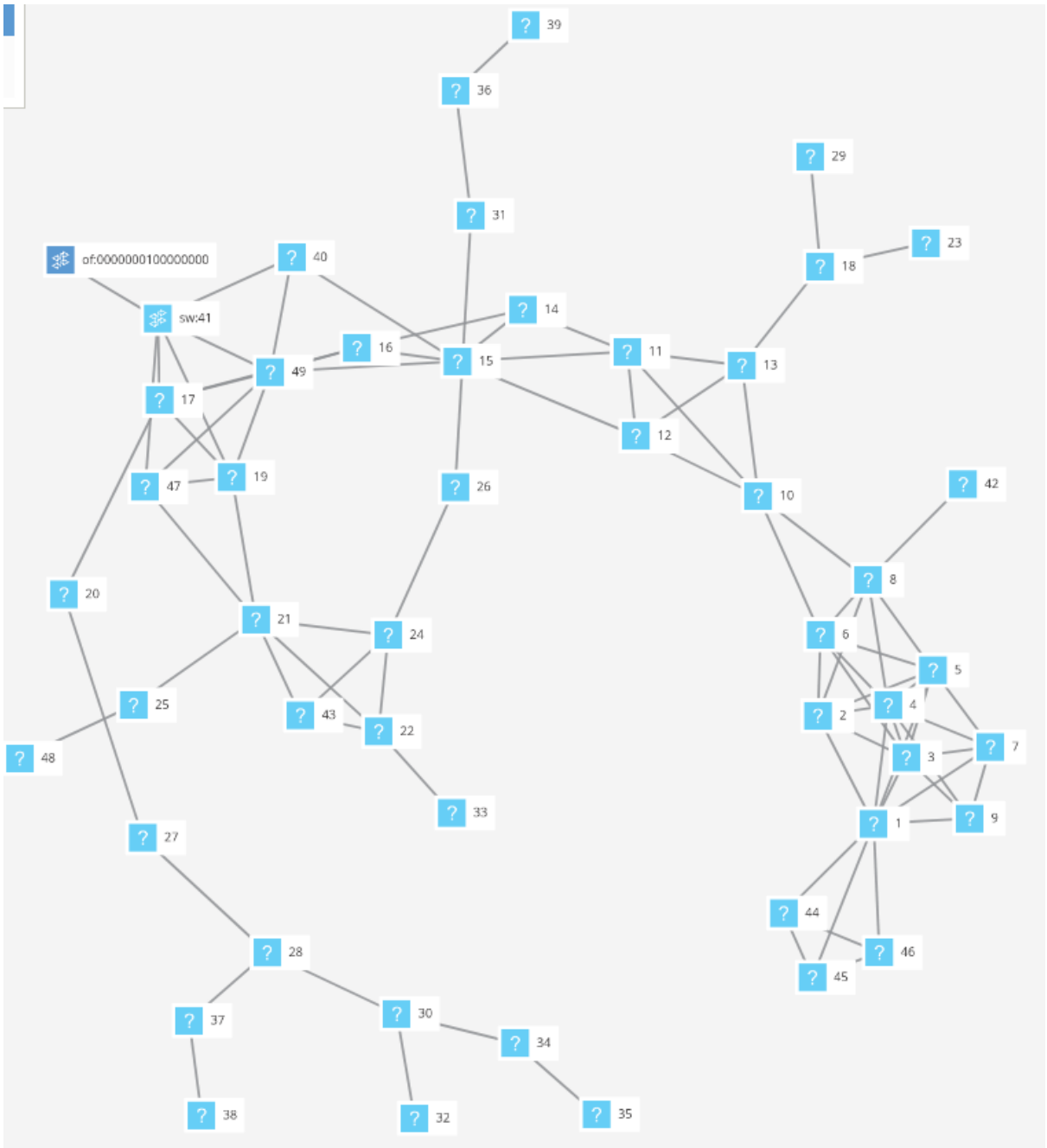


Ilustración 51:Topología híbrida

Capítulo 5: Conclusiones y trabajos futuros.

Conclusiones

En este proyecto se ha migrado la implementación de la aplicación de ONOS, para gestionar la información topológica, que el protocolo eHDDP recopila para el plano de control SDN. El objetivo principal era migrar dicha aplicación mediante el compilador Bazel, una herramienta que permite la construcción de softwares, pero de una forma más rápida, fiable y escalable. Para poder llevar a cabo este proyecto, se necesitó adquirir unos conocimientos teóricos sobre la estructura y el funcionamiento de las redes SDN, el lenguaje de programación Java, cómo funcionaba el protocolo Openflow y eHDDP. Una vez que se obtuvo el conocimiento teórico necesario, se realizaron una serie de cursos, para aprender a programar en Java orientado a objetos y se visualizaron una serie de tutoriales para saber cómo formar un proyecto Bazel. Una vez que se realizaron dichas tareas se comenzó a realizar el trabajo.

Se trabajó sobre un código que había sido proporcionado por los tutores de este proyecto, dicho código es la programación del protocolo eHDDP montado sobre un proyecto Maven. Durante el desarrollo, se han encontrado algunas dificultades, una de ellas fue la compatibilidad de las distintas herramientas utilizadas, por ejemplo, para formar el proyecto Bazel se necesitó del entorno de Intellij, pero no todas las versiones de Intellij disponían de la capacidad de realizar un proyecto Bazel, es por eso que en este trabajo se utiliza la versión 2020.2.4 de Intellij, además también fue necesario instalarse una versión de Ubuntu compatible con ONOS, en este caso, fue necesario la versión 18.04.

Otras de las dificultades encontradas, fue realizar el archivo BUILD para poder crear el proyecto Bazel, puesto que la información que hay disponible para crear dicho archivo es repetitiva y poco concreta. Además, hubo que modificar la mayoría de las librerías que utilizaba el código original, puesto que dicho código era un proyecto Maven y las librerías de Maven y Bazel se llaman de distinta manera. Afortunadamente, Intellij era capaz de reconocer que elementos eran necesarios modificar, lo cual ahorró mucho tiempo de investigación. Bazel es una herramienta con gran potencial puesto que es muy rápido a la hora de compilar y de formar proyecto, si este ya ha sido compilado con anterioridad, puesto que sólo comprueba si no hay errores en aquellos archivos que han sido modificados.

Una vez que se creó la aplicación, fue introducida en ONOS para poder comprobar si el código estaba realizando su función, para comprobar que la implementación se realizó correctamente se llevaron a cabo una serie de pruebas, que llevaron un tiempo considerable, pero que era necesario para realizar dicha comprobación. Finalmente, la implementación funcionó y se descubrió la topología de todas las redes propuestas. Es decir, recapitulando a los objetivos principales de la aplicación, la programación, la depuración y las pruebas de implementación se llevaron a cabo con éxito.

En conclusión, el proyecto Bazel ha demostrado lo rápido y eficaz que es para construir software y además, la aplicación también ha demostrado lo eficaz que es el protocolo eHDDP a la hora de descubrir la estructura de una red híbrida.

Trabajo futuro

Para proyectos que se puedan desarrollar en el futuro y que no se han desarrollado ahora debido a la falta de tiempo, es demostrar que la aplicación funciona en topologías más complejas, además también se puede intentar adaptar el código del protocolo eHDDP para poder recoger el rendimiento de distintas redes. También se podría realizar un estudio de los distintos protocolos de descubrimiento de topologías y estudiar cuáles son los más eficientes y cuáles son los que menos recursos necesitan para poder llevarse a cabo. Otro de los proyectos a poder desarrollar es la implementación de la aplicación desarrollada, pero en redes inalámbricas.

Anexos

Anexo 1: Presupuesto

El presupuesto se va a dividir en 4 apartados:

→ Coste de las horas de ingeniería.
→ Coste del material.
→ Coste de la conexión a Internet.
→ Coste total.

Costes	Cantidad	Euro/Unidad	Total (Euros)
Horas de ingeniería	350	40	14000
Ordenador	1	650	650
Conexión a Internet	4 meses	60	240
Total			14890

Tabla 4: Presupuesto

Anexo 2: Planificación temporal

2.1 Esquema temporal

Proyecto	Fecha de inicio	Duración	Fecha final
Investigación y estudio inicial.	01/07/2021	28	01/08/2021
Estudiar y comprender la arquitectura SDN.	01/07/2021	5	05/07/2021
Estudiar el protocolo HDDP.	06/07/2021	5	10/07/2021
Entender y configurar Mininet.	15/07/2021	5	19/07/2021
Familiarización con Java.	20/07/2021	10	29/07/2021
Familiarización con ONOS.	30/07/2021	3	01/08/2021
Implementación del procesamiento de la información topológica obtenida por el protocolo eHDDP en ONOS.	02/08/2021	22	15/09/2021
Implementar la aplicación de gestión de topologías.	02/08/2021	15	16/08/2021
Comprobar que el protocolo HDDP funciona en distintas configuraciones de red.	17/08/2021	7	23/08/2021
Valoración y verificación de la nueva implementación del protocolo HDDP.	24/08/2021	14	07/09/2021
Redacción del Trabajo Fin de Grado.	08/09/2021	16	23/09/2021

Tabla 5: Planificación temporal.

2.2 Diagrama de Gantt

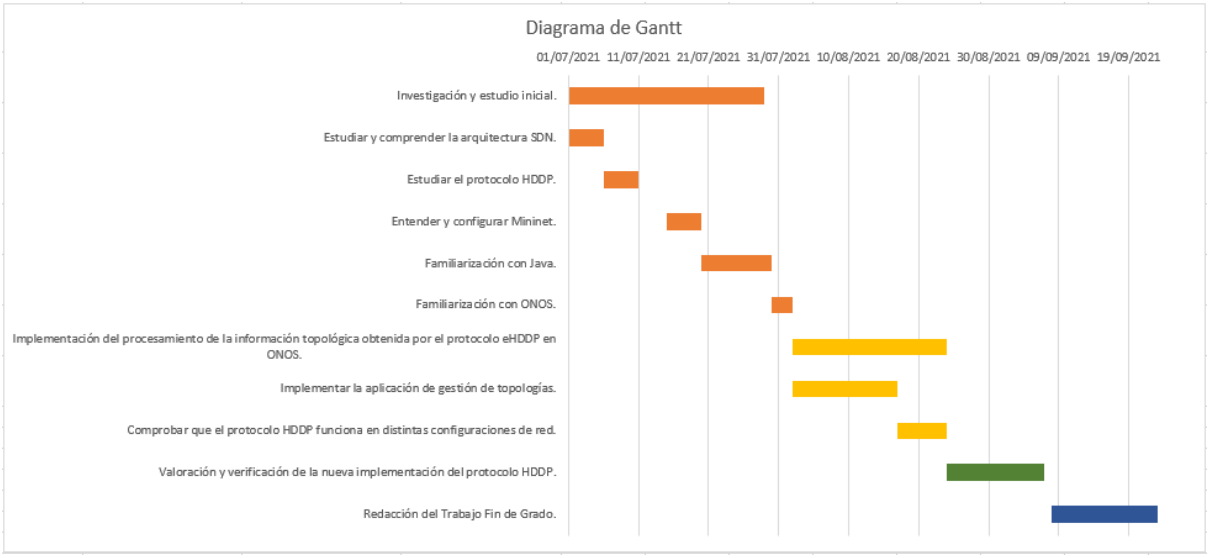


Ilustración 52: Diagrama de Gantt

Anexo 3: Instalación ONOS

En el siguiente apartado, se va a explicar qué versión de ONOS ha sido necesaria instalar y cómo se ha realizado la instalación para poder llevar a cabo este proyecto. Actualmente, la última versión disponible de ONOS es la 2.6.0, pero para este proyecto se ha elegido la última versión LTS disponible, es decir la versión 2.5.1, por si acaso la versión 2.6.0 no estaba perfeccionada.

En el caso de que alguien se quiera descargar la última versión, debe de introducir el comando en el terminal, como se muestra en la ilustración 53, con ese comando se construye ONOS y se consigue el código fuente.

```
irene@irene-HP-Pavilion-Laptop-15-ck0xx:~/Escritorio$ git clone https://gerrit.onosproject.org/onos
```

Ilustración 53: Descarga ONOS

En el caso de querer de cambiar de versión se debe de introducir el siguiente comando en el terminal:

```
irene@irene-HP-Pavilion-Laptop-15-ck0xx:~/Escritorio/onos$ git checkout 2.5.1
```

Ilustración 54: Cambiar de versión a ONOS 2.5.1

Con los siguientes comandos, el archivo onos.tar.gz, el cual se encuentra en la carpeta bazel-bin, se compilará y se instalará en el ordenador.

```
irene@irene-HP-Pavilion-Laptop-15-ck0xx:~/Escritorio/onos$ bazel build onos
```

Ilustración 55: Instalación ONOS.

A continuación, se muestra la línea de código que permite ejecutar ONOS de manera local en la máquina de desarrollo. Una vez se ejecute la siguiente línea de código, se observará el archivo de registro de ONOS mientras que en segundo plano comenzará ONOS a iniciarse.

```
irene@irene-HP-Pavilion-Laptop-15-ck0xx:~/Escritorio/onos$ bazel run onos-local -- clean debug
```

Ilustración 56: Ejecución ONOS.

En el comando de la ilustración 56, con la palabra clean, se consigue de ONOS una instalación limpia, con la palabra debug, se establece el puerto 5005 como puerto de depuración.

Si se quiere conectar a la consola de ONOS CLI, debe de introducirse la siguiente línea de código:

```
irene@irene-HP-Pavilion-Laptop-15-ck0xx:~/Escritorio/onos$ tools/test/bin/onos localhost
```

Ilustración 57: Establecimiento de conexión con la consola.

Si en vez de conectarse a la consola, lo que se desea es utilizar el navegador, se puede abrir utilizando el siguiente comando:

```
irene@irene-HP-Pavilion-Laptop-15-ck0xx:~/Escritorio/onos$ tools/test/bin/onos-gui localhost
```

Ilustración 58: Acceso navegador

Una vez que ya se han terminado de realizar los pasos anteriores, se puede acceder a ONOS, en el caso de este proyecto se accede de manera local, además para poder acceder se debe de introducir un usuario y una contraseña, onos y rocks respectivamente.

Ilustración 59: Pantalla de inicio ONOS

Anexo 4: Guía de comandos Mininet

Para poder emular una red, es necesario utilizar la herramienta Mininet. Los dispositivos necesarios para poder representar una red, se generan a través de líneas de comandos en el terminal. A continuación, se van a mostrar una serie de comandos básicos para poder manejar Mininet.

Para ejecutar Mininet se debe de insertar la siguiente línea de comando:

```
irene@irene-HP-Pavilion-Laptop-15-ck0xx:~$ sudo mn
```

Ilustración 60: Ejemplo comando sudo mn.

En este caso, se va a crear una topología por defecto, compuesta por dos hosts, 1 switch y un controlador.

Si se desea crear una topología compuesta por un switch y un cierto número de host, se dispone de la topología single, en este caso se han utilizado 4 hosts, para realizar la topología single, se debe de introducir el siguiente comando:

```
irene@irene-HP-Pavilion-Laptop-15-ck0xx:~$ sudo mn --topo single,4
```

Ilustración 61: Ejemplo comando sudo mn -topo single X.

Para crear una topología lineal se necesitan X switches y X hosts, en este ejemplo se van a utilizar 3 switches y hosts. Para poder crear la topología lineal se debe de introducir la siguiente línea de código:

```
irene@irene-HP-Pavilion-Laptop-15-ck0xx:~$ sudo mn --topo linear,3
```

Ilustración 62: Ejemplo comando sudo mn-topo linear, X.

Para crear una topología de árbol, se inserta el siguiente comando:

```
irene@irene-HP-Pavilion-Laptop-15-ck0xx:~$ sudo mn --topo tree,3
```

Ilustración 63: Ejemplo comando sudo mn -topo tree, X

Cuando la topología ya está creada, se puede interaccionar con el sistema mediante las siguientes líneas de código:

Con el siguiente comando se puede solicitar ayuda:

```
mininet> help
```

Ilustración 64: Ejemplo comando mininet>help.

Para enseñar los nodos, controladores y dispositivos finales que forman la red:

```
mininet> nodes
```

Ilustración 65: Ejemplo comando mininet>nodes.

Con la siguiente línea de código, se puede obtener la información de los nodos que forman parte de la red:

```
mininet> dump
```

Ilustración 66: Ejemplo comando mininet>dump.

A continuación, con el siguiente comando, se muestra cómo están conectados entre sí los equipos de la red:

```
mininet> net
```

Ilustración 67: Ejemplo comando mininet>net.

En el caso en el que se quiera conocer la interfaz del host:

```
mininet> h1 ifconfig
```

Ilustración 68: Ejemplo comando mininet>h1 ifconfig.

Para conocer el ancho de banda entre dos elementos:

```
mininet> iperf
```

Ilustración 69: Ejemplo comando mininet>iperf.

Para saber si entre dos hosts hay conectividad, se puede lanzar un ping desde el equipo final h1 hasta el equipo final h2:

```
mininet> h1 ping h2
```

Ilustración 70: Ejemplo comando mininet>h1 ping h2

El siguiente comando también dispone de la capacidad de lanzar un ping desde un equipo final hacia otro:

```
mininet> pingall
```

Ilustración 71: Ejemplo comando mininet>pingall.

Si se desea abrir un terminal para un equipo final:

```
mininet> h1 xterm
```

Ilustración 72: Ejemplo comando mininet>h1 xterm.

Cuando se quiera salir de Mininet, se deberá el siguiente comando:

```
mininet> exit
```

Ilustración 73: Ejemplo comando mininet>exit.

Bibliografía

[1] “Tecnología 5G: ventajas y desventajas para las empresas ” Último acceso: 19-07-2021 Disponible: https://www.apd.es/tecnologia-5g-ventajas-y-desventajas/

[2] Rosselyn Barroyeta. “Sistema inalámbrico mundial: el sueño de un mundo interconectado de Nikola Tesla”. Último acceso: 19-07-2021. Disponible: https://www.tekcrispy.com/2020/05/17/sistema-inalambrico-mundial-el-sueno-de-un-mundo-interconectado-de-nikola-tesla/

[3] “SDN – Software Defined Networking”, Marzo 2018, Último acceso: 20-07-2021. Disponible: https://www.eduardocollado.com/2018/03/26/podcast-140-sdn-software-defined-networking/

[4] “Funcionamiento de OpenFlow,” Feb. 2015, Último acceso: 25-07-2021. Disponible: http://electivaredesavanzadas.blogspot.com.es/

[5] P.Á. de Alba “Implementation of the HDDP protocol for hybrid SDN networks with P4”, Proyecto Final de Grado, Universidad de Alcalá, 2020.

[6] Isaias Martinez-Yelmo, Joaquin Alvarez-Horcajo, Juan Antonio Carral, Diego Lopez-Pajares. “eHDDP: Enhanced Hybrid Domain Discovery Protocol for network topologies with both wired/wireless and SDN/non-SDN devices.” Computer Networks, 2021, 107983, ISSN 1389-1286 Available: https://doi.org/10.1016/j.comnet.2021.107983

[7] Katia Hernández. “¿Qué es la arquitectura de red y cuáles son sus funciones?” Último acceso: 30-07.2021 Disponible: https://www.servnet.mx/blog/la-arquitectura-de-red-y-sus-funciones-para-un-buen-desempe%C3%B1o

[8] Joaquín Álvarez Horcajo. “Contribución al diseño y evaluación de conmutadores avanzados basados en protocolos de exploración de caminos” Tesis Doctoral, Universidad de Alcalá, 2020.

[9] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, y S. Banerjee, «DevoFlow: Scaling Flow Management for High-performance Networks», en Proceedings of the ACM SIGCOMM 2011 Conference, New York, NY, USA, 2011, pp. 254–265.

[10] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, y A. Vahdat, «Hedera: Dynamic Flow Scheduling for Data Center Networks», en Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, Berkeley, CA, USA, 2010, pp. 19–19.

[11] B. Stephens, A. Cox, W. Felter, C. Dixon, y J. Carter, «PAST: Scalable Ethernet for Data Centers», en Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies, New York, NY, USA, 2012, pp. 49–60.

[12] 802.1AB: Link Layer Discovery Protocol (LLDP)

[13] J. Álvarez-Horcajo, "Desarrollo de un switch híbrido OpenFlow/All-Path," Master's thesis, Universidad de Alcalá, Alcalá de Henares, Madrid, Spain, 2017. Disponible: https://ebuah.uah.es/dspace/handle/10017/29680

[14] E. L. Fernandes, E. Rojas, J. Alvarez-Horcajo, Z. L. Kis, D. Sanvito, N. Bonelli, C. Cascone, and C. E. Rothenberg, "The road to BOFUSS: The basic OpenFlow userspace software switch," Journal of Network and Computer Applications, vol. 165, p. 102685, 2020. Disponible: http://www.sciencedirect.com/science/article/ pii/S1084804520301594

[15] "BEBA: BEhavioral Based Forwarding," Último acceso: 10-08-2021. Disponible: http://http://www.beba-project.eu/

[16] E. Rojas, G. Ibañez, J. M. Gimenez-Guzman, J. A. Carral, A. García-Martinez, I. Martinez-Yelmo, and J. M. Arco, "All-Path bridging: Path exploration protocols for data center and campus networks," Computer Networks, vol. 79, no. 0, pp. 120 – 132, 2015. Disponible: http://www.sciencedirect.com/science/article/pii/ S1389128615000055

[17] R. S. Montero, E. Rojas, A. A. Carrillo, and I. M. Llorente, "Extending the Cloud to the Network Edge," Computer, vol. 50, no. 4, pp. 91–95, 2017.

[18] L. Peterson, A. Al-Shabibi, T. Anshutz, S. Baker, A. Bavier, S. Das, J. Hart, G. Palukar, and W. Snow, "Central office re-architected as a data center," IEEE Communications Magazine, vol. 54, no. 10, pp. 96–101, October 2016.

[19] O. Salman, I. H. Elhajj, A. Kayssi and A. Chehab, "SDN controllers: A comparative study," 2016 18th Mediterranean Electrotechnical Conference (MELECON), Lemesos, 2016, pp. 1-6, doi: 10.1109/MELCON.2016.7495430.

[20] "OpenDaylight" Último acceso: 12-08-2021. Disponible: https://www.opendaylight.org/

[21] David Erickson. 2013. “The beacon openflow controller”n Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking (HotSDN '13). Association for Computing Machinery, New York, NY, USA, 13–18. DOI. Disponible: https://doi.org/10.1145/2491185.2491189

[22] A. Azzouni, R. Boutaba, N. T. M. Trang, and G. Pujolle, “softdp: Secure and efficient openflow topology discovery protocol,” in NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium, 2018, pp. 1–7.

[23] J. Alvarez-Horcajo, E. Rojas, I. Martinez-Yelmo, M. Savi and D. Lopez-Pajares, "HDDP: Hybrid Domain Discovery Protocol for Heterogeneous Devices in SDN," in IEEE Communications Letters, vol. 24, no. 8, pp. 1655-1659, Aug. 2020, doi: 10.1109/LCOMM.2020.2991347.

[24] Washington A. Velásquez Vargas, «Emulación de una red definida por software utilizando MiniNet», Escuela Técnica Superior de Ingenieros en Telecomunicaciones (ETSIT - UPM).

[25] «Introducción a los grupos de control (cgroups) de Linux», Introducción a los grupos de control (cgroups) de Linux. Último acceso: 15-08-2021. Disponible: https://elpuig.xeill.net/Members/vcarceler/articulos/introduccion-a-los-grupos-decontrol-cgroups-de-linux.

[26] lantz edited, «Cluster Edition Prototype». Último acceso: 15-08-2021. Disponible: https://github.com/mininet/mininet/wiki/Cluster-Edition-Prototype.

[27] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," in Proceedings of the IEEE, vol. 103, no. 1, pp. 14-76, Jan. 2015, doi: 10.1109/JPROC.2014.2371999.

[28] Internet Engineering Task Force (IETF). “Forwarding and Control Element Separation (ForCES) Protocol Specification” Último acceso: 15-08-2021 Disponible: https://tools.ietf.org/html/rfc5810.

[29] Jackson Emilio Martínez Copete. “Estudio del funcionamiento de la herramienta Mininet”. Proyecto Final de Grado, Universidad Católica de Pereira, 2015.

[30] Sendoe Moronta. “Introducción a Bazel”. Último acceso: 19-09-2021. Disponible: https://www.bbvanexttechnologies.com/blogs/introduccion-a-bazel/

[31] Jorge Cano. “Entendiendo Bazel” Último acceso: 19-09-2021. Disponible: https://medium.com/@jorgeucano/entendiendo-bazel-377ef4063fa4

[32] E.E. García, J.A. Solano “Guía práctica de estudio: Depuración de programas” Último acceso:29-09-2021.
Disponible:http://odin.fib.unam.mx/salac/practicasFP/Anexo%20DepuracionProgramas.pdf

[33] “Debugging ONOS with Intellij” Último acceso: 29-09-2021. Disponible: https://www.youtube.com/watch?v=UzWcI9KvP0g

[34] “Installing Bazel on Ubuntu ”Último acceso: 1-10-2021. Disponible: https://docs.bazel.build/versions/main/install-ubuntu.html

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR