ESCUELA DE INGENIERÍA DE FUENLABRADA

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE LA TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DISEÑO E IMPLEMENTACIÓN DE UN PROTOCOLO DE COMUNICACIONES PARA EL DESCUBRIMIENTO DE ENLACES EN LA TOPOLOGÍA DEL PLANO DE CONTROL IN-BAND PARA REDES SDN PERIPLUS

Autor: Jesús Fernández Funcia
Tutora: Eva María Castro Barbero
Contutor: Pedro de las Heras Quirós

Curso académico 2023-2024

Trabajo Fin de Grado

Diseño e Implementación de un Protocolo de Comunicaciones para el Descubrimiento de Enlaces en la Topología del Plano de Control In-Band para Redes SDN Periplus

Autor : Jesús Fernández Funcia
Tutora : Eva María Castro Barbero
Contutor : Pedro de las Heras Quirós

La defensa del presente Proyecto Fin de Carrera se realizó el día de de 2024, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2024

A mis abuelos
que siempre han estado y estarán.

Agradecimientos

En primer lugar quiero agradecer el apoyo recibido por mis padres, María Isabel y Jesús, y por mi hermano Jaime. Ellos han sido testigos de todo este proceso académico y personal que empezó hace tantos años, y que sin su apoyo y compresión no hubiera sido posible completar. Gracias por acompañarme todas esas tardes y noches de estudio, por compartir mis alegrías, pero sobre todo mis frustraciones y adversidades. Solo ellos saben el largo camino que he recorrido para llegar hasta aquí.

Quiero dedicar un agradecimiento especial a mis abuelos Jesús, Olga, Isabel y Matías. Aunque desgraciadamente no han podido ver este trabajo completado, han sido una parte fundamental en mi desarrollo tanto académico como personal y sé que me van a acompañar siempre. Este trabajo va por ellos.

También quiero dedicarle este trabajo a mis amigos, por acompañarme durante todos estos años, compartir tantos momentos juntos que nunca olvidaré y en especial por darme ánimos y ayudarme a evadirme cuando las cosas no me salían bien. Es imposible no hacer una mención especial a mis compañeros y amigos de carrera, que han hecho que las horas de estudio fueran más amenas y que las pequeñas adversidades que iban surgiendo fueran mucho más pequeñas de lo que parecían.

No me puedo olvidar de mi novia Alba y de todas las personas importantes que me han apoyado y acompañado. Gracias por la paciencia y la compresión durante todos estos años, habéis sido un pilar muy importante y un pedacito de este trabajo también es vuestro sin ninguna duda.

Por último, y no menos importante, quiero agradecer a Eva y a Pedro su ayuda en el desarrollo de todo este proyecto, por su tutorización y por su continua disponibilidad para ayudarme ante cualquier duda que surgía y por su paciencia ante mis continuas preguntas. También quiero mencionar a todos los profesores y tutores de la escuela que me han ayudado a crecer académicamente y me han permitido llegar hasta aquí.

A todos, gracias.

Resumen

Las Redes Definidas por Software (SDN) representan un nuevo enfoque en las telecomunicaciones, rompiendo con la estructura de las redes convencionales. En lugar de configurar individualmente cada dispositivo de red, las SDN centralizan esta tarea en un programa llamado controlador, que toma decisiones sobre la función de encaminamiento y otras. Este método ofrece una solución más ágil para adaptarse a las necesidades cambiantes de las redes actuales.

El objetivo de este Trabajo de Fin de Grado es desarrollar e implementar un algoritmo que se muestre como alternativa al protocolo LLDP para el aprendizaje de rutas alternativas dentro de las SDN. Este nuevo algoritmo crea un único paquete en el controlador de una subred, que es distribuido por inundación a través de todos los switches del escenario. De esta forma, también se reduce la cantidad de paquetes.

Las SDN empleadas en este proyecto están sustentadas en 3 tecnologías principales: Mininet, OpenFlow y Open vSwitch. Estas tecnologías, junto con el lenguaje de programación Python, permiten probar esta nueva implementación en dos escenarios diferentes: uno compuesto por un solo controlador y otro con dos controladores. De esta forma se puede mostrar que el algoritmo se implementa con éxito tanto dentro de una misma zona gestionada por un único controlador como entre dos zonas que pertenecen a controladores diferentes.

Summary

Software-Defined Networking (SDN) represents a new approach in telecommunications, breaking away from the structure of conventional networks. Instead of individually configuring each network device, SDNs centralize this task in a program called the controller, which makes routing decisions. This method provides a more agile solution to adapt to the changing needs of current networks.

The aim of this Final Degree Project is to develop and implement an algorithm that serves as an alternative to the Link Layer Discovery Protocol (LLDP) for learning alternative paths within SDNs. This new algorithm creates a single packet in the controller of a subnet, which is distributed by flooding through all the switches in the scenario. This approach also reduces the number of packets present in the scenario.

The SDNs used in this project are based on three main technologies: Mininet, OpenFlow, and Open vSwitch. These technologies, along with the Python programming language, enable testing this new implementation in two different scenarios: one composed of a single controller and another with two controllers. This demonstrates that the algorithm is successfully implemented both within a single zone managed by a single controller and between two zones belonging to different controllers.

Índice general

Lista de Acrónimos

SDN Software Defined Networking

LLDP Link Layer Discovery Protocol

API Interfaz de Programación de Aplicaciones

OSI Open Systems Interconnection

CLI Command Line Interface

NIC Adaptadores de Red Virtual

TCP Transmission Control Protocol

NSH Network Service Header

IP Internet Protocol

MV Máquina Virtual

MSS Tamaño Máximo de Segmento

MTU Unidad Máxima de Transferencia

BFD Bidirectional Forwarding Detection

C-Adv Controller Advertisement

Índice de figuras

Capítulo 1

Introducción

1.1. Introducción

Las redes de computadores desempeñan un papel clave en muchos campos de la vida actual, como por ejemplo en las comunicaciones o en la prestación de servicios. La gestión y el control de estos sistemas de una manera eficiente es esencial para poder asegurar su rendimiento y su adaptación a las necesidades de una sociedad que está en constante evolución, y que cada vez demanda más el uso de este tipo de tecnologías.

Las redes de computadores tradicionales están organizadas en dos grandes planos: el plano de control y el plano de datos. El plano de control se encarga de definir cómo se ha de procesar el tráfico de la red en el plano de datos. Por ejemplo los algoritmos de encaminamiento son un ejemplo de funciones definidas por el plano de control. El plano de datos se ocupa de cursar el tráfico de acuerdo a las reglas definidas por el plano de control. El encaminamiento de los mensajes según la política de encaminamiento definida por el plano de control es una de las funciones principales del plano de datos.

A pesar de presentarse como dos conceptos independientes, en la práctica ambos planos suelen estar integrados en los equipos. Las Redes Definidas por Software (o Software Defined Networks—SDN) proponen separar estos dos planos e instalarlos en dispositivos independientes. Esto ofrece varias ventajas, como la escalabilidad, la flexibilidad y la innovación.

Para poder comunicar los dispositivos del plano de control con los dispositivos del plano de datos se utilizan protocolos como OpenFlow. Los controladores decidirán el comportamiento de los dispositivos de red y lo instalarán en dichos dispositivos. Por tanto, el funcionamiento de los dispositivos de la red queda determinado por la configuración que establezca el controlador. Los controladores del plano de control necesitan disponer de una red de control que les permita comunicarse tanto con los dispositivos del plano de datos como con los demás controladores del plano de control.

Esta red de control puede ser definida a través de dos estrategias diferentes:

- **Out-of-band**: la red de control y la red del plano de datos se encuentran separadas físicamente. En esta estrategia, se utilizan protocolos distribuidos para el enrutamiento y la conmutación por los dispositivos de red, que se encuentran directamente conectados con la red de control a través de un puerto específico en los dispositivos del plano de datos. De esta forma, los protocolos de encaminamiento permiten que el controlador pueda comunicarse con los dispositivos de la red e instalar el comportamiento que deben implementar. Esta estrategia se utiliza en centros de procesamiento de datos.
- **In-band**: en ocasiones no se puede justificar el uso de una red diferente a la usada en el plano de datos sólo para cursar el tráfico de control. En esta estrategia, el tráfico de control se reenvía a través de los mismos dispositivos de red que se utilizan para reenviar el tráfico del plano de datos. Los dispositivos de red no establecen una comunicación directa con los controladores, ya que requieren implementar protocolos y mecanismos adicionales. Esta configuración se emplea en escenarios que presentan redes muy extensas, como pueden ser una red de un campus o redes de entornos industriales.

Aunque se muestren como dos configuraciones opuestas, también se pueden utilizar estrategias híbridas. Este Trabajo de Fin de Grado se centra en el uso de planos de control in-band para redes SDN.

Es necesario tener en cuenta que la implementación de una red de control in-band presenta varios retos, entre los que destaca la falta de estandarización de los protocolos y los mecanismos necesarios. Para poder superar estos desafíos, los controladores necesitan descubrir y establecer rutas bidireccionales hacia los dispositivos de red o switches que se inician en la topología de la red de control. Una vez establecidas las conexiones los controladores pueden instalar reglas que posibilitan a los switches reenviar tráfico a lo largo de las rutas calculadas.

La detección y recuperación ante caídas de enlaces es otro de los principales retos a los que se enfrentan los planos de control in-band en redes SDN. Para garantizar la disponibilidad del servicio, todos los equipos presentes en la red deben implementar mecanismos para detectar rápidamente enlaces o equipos defectuosos. De esta forma, se podrá redirigir el tráfico a través de rutas alternativas, minimizando el impacto del problema en la red.

Periplus es un plano de control in-band para SDN que aborda dos de los principales desafíos de este tipo de redes: la necesidad de configuración manual de nuevos switches y la necesidad de protocolos auxiliares para descubrir enlaces en la red.

En Periplus, para solventar el problema de la configuración el controlador utiliza un mecanismo que emplea Proxy ARP, Anycast y mensajes Packet-In/Packet-Out de OpenFlow para facilitar el proceso de incorporación de switches a la red de control. Una vez que el controlador Periplus recibe el primer mensaje enviado por un switch

en el proceso de arranque, determina su ubicación en la topología de la red e instala en él reglas de flujo a través de la red de control, que permitirán al nuevo dispositivo comunicarse con el controlador en ambas direcciones. Periplus utiliza el mecanismo de encaminamiento en origen para que los dispositivos puedan comunicarse entre ellos.

Periplus utiliza el Protocolo de Exploración de Capa de Enlace (Link Layer Discovery Protocol—LLDP) para descubrir los enlaces redundantes en la red, que no han sido descubiertos durante el proceso de arranque. LLDP es un protocolo estándar que permite a los controladores descubrir información sobre los enlaces físicos que hay entre los switches.

A lo largo de este Trabajo de Fin de Grado, se emplean las tecnologías descritas anteriormente, y el lenguaje de programación Python, para poder desarrollar un nuevo algoritmo que sirva como reemplazo del protocolo LLDP. Dicho algoritmo tiene como objetivo enviar un único paquete desde el controlador principal a todos los equipos de la subred por inundación, permitiendo así actualizar el grafo de la topología del escenario en cada switch, sin perder la capacidad de establecer rutas alternativas y comunicarse con otras subredes vecinas.

De esta forma se consigue reducir el número de paquetes presentes en el escenario en comparación con LLDP, simplificando el proceso de reenvío de paquetes, permitiendo que se puedan aprender en el controlador rutas alternativas sin utilizar LLDP. También se implementan mecanismos para prevenir la aparición de bucles que puedan provocar explosiones de tráfico en los reenvíos.

Para poder probar este nuevo algoritmo se han utilizado dos escenarios de red en los que se han analizado las características previamente descritas: el primer escenario presenta un único controlador en una sola subred, y el segundo escenario está compuesto por 2 controladores diferentes que forman 2 subredes vecinas. Ambos escenarios son descritos a lo largo del proyecto.

1.2. Tecnologías y herramientas de red

En esta sección se introduce la arquitectura de red SDN, los protocolos que se van a utilizar en este Trabajo Fin de Grado y el entorno de emulación necesario para desarrollar el trabajo.

1.2.1. SDN

Las SDN son redes que nacen con la idea de generar una mayor flexibilidad y manejo en las redes de comunicaciones. Hasta finales del siglo XX, estas redes estaban controladas por organismos privados que entorpecían su progreso y su actualización, por lo tanto se podría decir que las SDN estaban delimitadas dentro de un sistema rígido.

Es a partir de comienzo del siglo XXI, cuando se produce un cambio en la situación de estas redes, ya que comienza a implementarse el protocolo OpenFlow, que permite separar el plano de control de datos en switches y routers. Con el desarrollo de este nuevo protocolo, un único controlador podía centralizar las decisiones del tratamiento de los paquetes presentes en una red.

Esta transformación atrae a las grandes empresas tecnológicas como Google, que adoptan esta nueva tecnología como método de gestión de sus redes, con su correspondiente beneficio económico. Las SDN transforman el campo digital al desarrollar un trabajo más eficiente y flexible en la configuración de redes, adaptándose a las necesidades crecientes del sector [9].

En la actualidad, las SDN introducen un controlador centralizado que se ocupa de separar la lógica de control de la infraestructura de red, consiguiendo así, supervisar y gestionar el tráfico de red [12]. Este controlador recurre a Interfaces de Programación de Aplicaciones (APIs) que permiten el desarrollo de aplicaciones de gestión de red que implementan una configuración más dinámica.

Estas infraestructuras constan de varios componentes clave que permiten esta evolución tan significativa:

- **Controlador SDN**: El controlador es el núcleo de la SDN, es un programa responsable de tomar decisiones de enrutamiento y configuración de la red. Algunos controladores de uso frecuente incluyen OpenDaylight y ONOS [15].
- **Dispositivos de red**: enrutadores y switches físicos que encaminan el tráfico de red. En una SDN, estos dispositivos son más simples y se centran en el reenvío de paquetes, ya que la administración de las decisiones de red se lleva a cabo en el controlador. El comportamiento de los dispositivos de red queda determinado por la configuración que reciben del controlador.
- **APIs de enlace hacia abajo** (Southbound APIs): Estas interfaces permiten que el controlador se comunique con los dispositivos de red y les instruya sobre cómo deben gestionar el tráfico. El protocolo más extendido para estas interfaces es OpenFlow [20].
- **APIs de Enlace Hacia Arriba** (Northbound APIs): Estas interfaces permiten a las aplicaciones y servicios de nivel superior comunicarse con el controlador SDN. Esto es esencial para implementar políticas de red específicas y para crear servicios personalizados.
- **Aplicaciones SDN**: Estas aplicaciones se ejecutan en la parte superior de la infraestructura SDN y aprovechan la flexibilidad que ofrece para implementar servicios y políticas de red específicos [5].

En la figura 1.1 se muestra a nivel conceptual cual es la diferencia entre las redes tradicionales y las nuevas SDN que externalizan el plano de control a un solo equipo

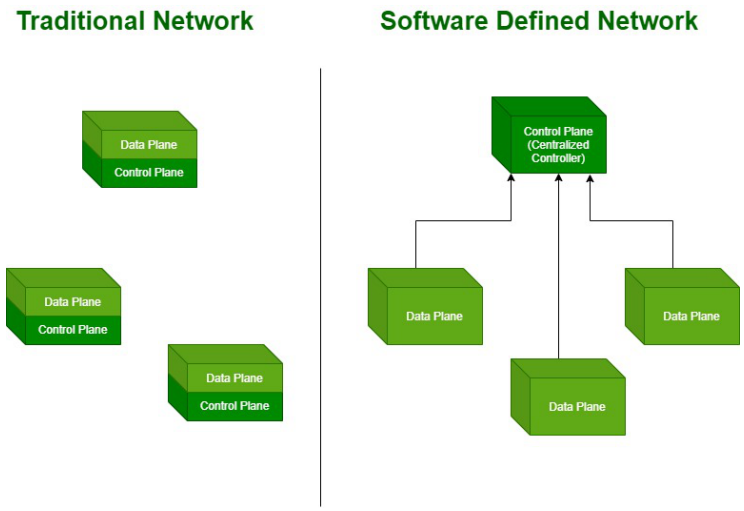

Figura 1.1: Red tradicional y una Red SDN [4].

centralizado.

El desarrollo de este tipo de redes centralizadas redefine las características proporcionadas por la red, entre las que destacan:

- **Presentar una mayor flexibilidad**: Las redes SDN son altamente adaptables a las necesidades cambiantes de las organizaciones, lo que permite una configuración dinámica y personalizada.
- **Gestionar mejor los recursos**: SDN permite una asignación más eficiente de recursos de red, lo que puede llevar a un uso más eficiente de la capacidad y reducir costos.
- **Simplificar la gestión**: La centralización del control simplifica la gestión de la red y facilita la implementación de políticas de seguridad y calidad de servicio.
- **Permite un desarrollo más acelerado**: SDN fomenta la innovación al permitir el desarrollo de nuevas aplicaciones y servicios de red de manera más rápida y sencilla.

Aunque las SDN están ampliamente implementadas, no están exentas de desafíos que le obligan a evolucionar. Algunos de éstos incluyen la seguridad de la red y la necesidad de garantizar la interoperabilidad con sistemas y dispositivos existentes. No obstante, la comunidad sigue abordando estos problemas de forma activa para poder mejorar su desarrollo [2].

1.2.2. Mininet

Mininet es un entorno de emulación de redes de código abierto que es ampliamente utilizado tanto en el campo de la investigación como en las redes de computadoras [13]. Esta tecnología nace a principios del siglo XXI con el objetivo de desarrollar un entorno de prueba para poder realizar experimentos con redes de manera eficiente, realista y rápida.

Mininet se basa en el principio de la virtualización de redes, donde se crean múltiples máquinas virtuales (MV) ligeras, llamadas ”mininetes”, dentro de un único sistema operativo. Estos mininetes representan dispositivos de red, como hosts y switches, que están interconectados para formar una topología de red deseada. Un controlador de red, como OpenDaylight o Floodlight, se utiliza para gestionar el comportamiento de los switches virtuales [11].

Esta plataforma permite a los usuarios crear topologías de red personalizadas utilizando máquinas virtuales ligeras (contenedores). Este nuevo entorno, junto con su naturaleza de código abierto, facilitó la creación de redes escalables y realistas en entornos controlados y fomentó su adopción en instituciones académicas e industriales. La combinación de Mininet y SDN ha permitido diversificar implementación y desarrollar escenarios tecnológicos diversos.

Los componentes esenciales de Mininet incluyen:

- **Hosts y switches virtuales**: Estos elementos forman la infraestructura de red virtual y representan los dispositivos que forman la topología emulada.
- **Controlador de red**: El controlador de red es un programa que se encarga de gestionar y controlar el comportamiento de los switches virtuales, lo que permite establecer políticas de red y controlar el tráfico.
- **Interfaz de línea de comandos**: Mininet proporciona una Command Line Interface (CLI) que permite a los usuarios interactuar con la topología de red emulada, configurar dispositivos y supervisar el tráfico.

En la figura 1.2 se muestra una representación de la emulación de una red física en un entorno virtual donde se puede observar un controlador que gestiona 3 switches para dar conectividad a 3 máquinas.

Mininet sustenta su desarrollo en una serie de características que facilitan el manejo y desarrollo de las SDN, como por ejemplo:

- **Desarrollo de un entorno controlado**: Permite a los investigadores crear entornos de red controlados y reproducibles para probar y validar protocolos y aplicaciones de red.
- **Implementaciones asequibles**: Mininet utiliza máquinas virtuales ligeras, lo que permite la emulación de redes complejas en una única máquina física.

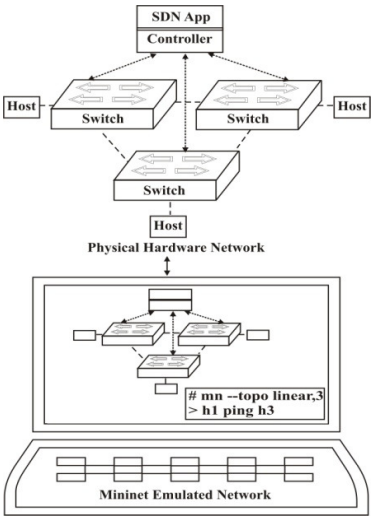

Figura 1.2: Emulación de una red física frente a una red Mininet [19].

- **Redes y estructuras flexibles**: Los usuarios pueden crear topologías de red personalizadas y experimentar con diferentes configuraciones de red.
- **Utilización en diferentes entornos**: Mininet se utiliza ampliamente en entornos académicos para enseñar conceptos de redes de computadoras y experimentar con escenarios de red prácticos [14].

Mininet ofrece una interfaz de programación de alto nivel en Python para definir la topología de la red que se desea arrancar: máquinas, switches, enlaces entre los diferentes dispositivos, arranque del controlador SDN dentro de la red, etc. Esta interfaz de programación ofrece la posibilidad de programar la topología de la red de forma sencilla y establecer diferentes escenarios para realizar pruebas en el diseño de protocolos de red.

Actualmente Mininet es una de las plataformas clave en el desarrollo de redes de computadoras, ya que nos permite experimentar de una manera eficiente y flexible con topologías de red personalizadas. Su capacidad para crear entornos controlados y reproducibles lo convierte en una herramienta esencial para la investigación y el desarrollo en el ámbito de las redes [10].

1.2.3. OpenFlow

OpenFlow es un protocolo de comunicación utilizado en las SDN que establece un estándar para la separación entre el plano de control y el plano de datos de una red, lo que brinda una mayor flexibilidad y adaptabilidad en la gestión de redes. Este protocolo permite la programación centralizada y dinámica de dispositivos de red, facilitando

así la implementación de políticas de red en el controlador y su instalación en los dispositivos de red.[12].

A medida que las SDN ganaron atención en la comunidad de redes y la industria, OpenFlow se consolidó como uno de los protocolos clave para implementar SDN. Grandes empresas tecnológicas y organizaciones adoptaron OpenFlow para mejorar la gestión de sus redes y permitir una mayor flexibilidad y programabilidad.

El concepto central de OpenFlow se basa en la separación de los planos de control y datos en una red. Mientras que los dispositivos de red tradicionales como routers y switches gestionan tanto el control como el reenvío de paquetes, en una arquitectura donde el plano de datos (reenvío de paquetes) y el plano de control (toma de decisiones) están separados, OpenFlow regula la comunicación entre el controlador y los dispositivos de red para que sea posible el funcionamiento de ambos planos. Los dispositivos de red, conocidos como switches OpenFlow, reenvían paquetes de acuerdo con las reglas y decisiones definidas por un controlador centralizado [12].

El protocolo OpenFlow se sustenta sobre 3 elementos clave:

- **Switches OpenFlow**: Estos dispositivos son responsables de reenviar paquetes de acuerdo con las instrucciones del controlador OpenFlow que se instalan en las tablas de flujo del switch. Los switches OpenFlow se caracterizan por ser dispositivos simples en términos de lógica de control.
- **Controlador OpenFlow**: El controlador OpenFlow es el componente central que toma decisiones de enrutamiento y configura las tablas de flujo en los switches OpenFlow. Ejemplos populares de controladores OpenFlow incluyen OpenDaylight y Ryu.
- **Tablas de Flujo**: Las tablas de flujo en los switches OpenFlow determinan cómo se deben manejar los paquetes entrantes. El controlador OpenFlow programa estas tablas para establecer políticas de red y reglas de enrutamiento a través del protocolo OpenFlow.

En la figura 1.3 se muestra cómo existe una comunicación entre el controlador y cada dispositivo de la red. Esta comunicación se establece a través del protocolo OpenFlow. De esta forma, el controlador instala el comportamiento del dispositivo de red en tablas de flujos que gobiernan el funcionamiento del dispositivo.

El desarrollo de un protocolo con estas características, ha generado un impacto significativo en la forma en que se gestionan las redes modernas. En primer lugar, proporciona flexibilidad al permitir una mayor programabilidad y adaptabilidad de la red. Esto facilita la implementación de políticas y servicios personalizados, lo que resulta en una red que se puede adaptar con precisión a las necesidades específicas de una organización.

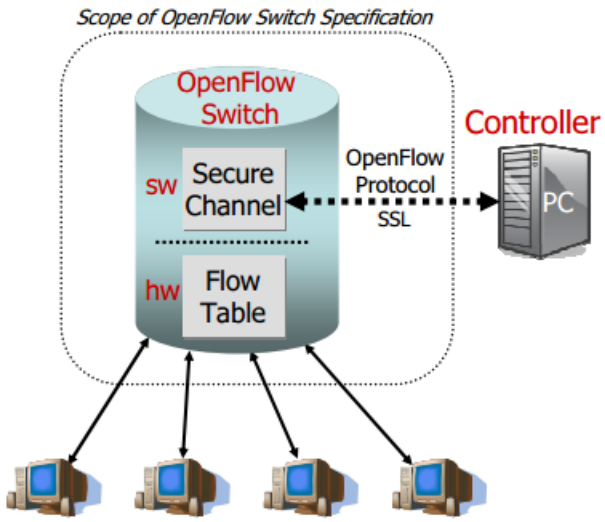

Figura 1.3: Partes de un Switch OpenFlow [7].

Además, la centralización en un controlador de OpenFlow ofrece una gestión de políticas de red eficiente y la implementación de cambios de manera coherente en toda la red. Esta característica garantiza que las políticas y reglas se apliquen de manera uniforme, lo que facilita la consistencia y la seguridad de los escenarios.

Otra ventaja clave radica en la optimización de recursos que OpenFlow permite. Esta tecnología habilita una asignación más eficiente de los recursos de red disponibles, lo que conduce a un uso más efectivo de la capacidad de la red y, en última instancia, a la reducción de los costos operativos.

A pesar de reunir todas las características previamente enunciadas, OpenFlow muestra desafíos aún por resolver, como la seguridad de la red y la escalabilidad. Sin embargo, la comunidad sigue abordando estos problemas, en paralelo con las SDN, para poder mejorar su desarrollo.

1.2.4. Open vSwitch

Durante todo el siglo XX, las redes podían ser gestionadas principalmente en entornos físicos. Debido al desarrollo de la virtualización a principios del siglo XXI, y la creación de las MVs, surgió la necesidad de desarrollar un sistema o switch virtual que gestionara todas estas nuevas tecnologías, permitiendo conectar y ejecutar todas estas MVs en una sola máquina física, dando origen a los vSwitch.Esto permitió la segmentación, la eficiencia y el aislamiento de redes virtuales, lo que resultó fundamental para garantizar la seguridad y la eficiencia en entornos de virtualización [18].

El concepto central de vSwitch se basa en la creación de un switch de red virtual dentro de un hipervisor o sistema de virtualización. Este switch virtual se encarga de enrutar paquetes de datos entre MVs y la red física, además de permitir la propia comunicación entre las diferentes máquinas. Open vSwitch es una implementación de switch virtual de código abierto desarrollado por The Linux Fundation.

Los componentes fundamentales de Open vSwitch incluyen:

- **Adaptadores de Red Virtual (vNICs)**: Cada MV tiene una o más vNICs que se conectan al Open vSwitch. Estos actúan como interfaces de red virtuales para las MVs y se comunican con el Open vSwitch para el enrutamiento de paquetes.
- **Tabla de Conmutación**: El Open vSwitch mantiene una tabla de conmutación que registra las direcciones MAC de las MVs y sus respectivos puertos en el switch. Esto permite que el Open vSwitch pueda operar con el comportamiento de un switch Ethernet físico.
- **Controlador de Open vSwitch**: Algunos hipervisores, como VMware vSphere, incluyen un controlador de OpenFlow que permite la configuración y gestión del Open vSwitch, incluyendo la definición de políticas de red y la configuración de reglas de seguridad.

En la figura 1.4 de puede observar un esquema de como se conectan todos los componentes que forman parte de la estructura de un Open vSwitchpara permitir la conectividad de las MV.

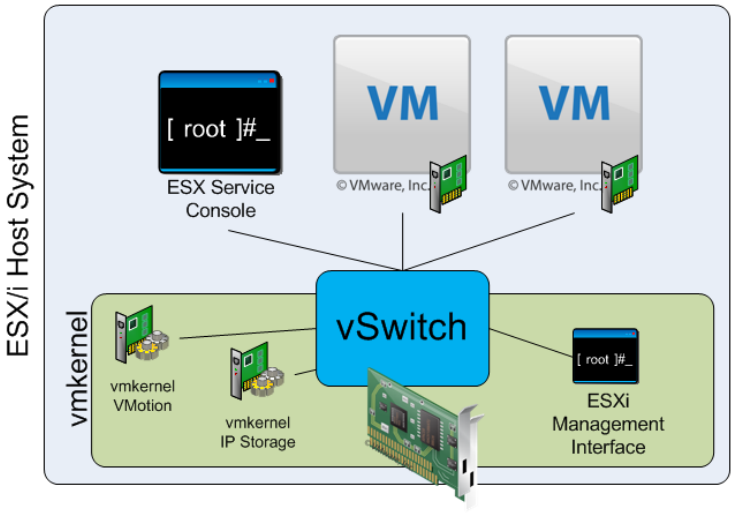

Figura 1.4: Conexión de MVs a través de Open vSwitch [17].

Open vSwitch aporta una amplia gama de nuevas características, siendo uno de los componentes que optimizan la gestión de redes en entornos virtuales.

En primer lugar, Open vSwitch permite la segmentación de red, lo que significa que se pueden crear múltiples redes virtuales en una infraestructura física compartida. Esto facilita la segmentación y el aislamiento de una manera eficiente del tráfico entre las distintas MVs, lo que contribuye al desarrollo de un entorno virtualizado organizado y seguro.

Además, Open vSwitch destaca por su flexibilidad. Los administradores de red tienen la capacidad de configurar y ajustar la conectividad de red de las MVs de manera dinámica, sin requerir modificaciones en la infraestructura física subyacente. Esto facilita la adaptación de la red a las cambiantes necesidades de las aplicaciones y servicios, lo que permite desarrollar escenarios en constante evolución y adaptarlos a diferentes campos.

Por último, Open vSwitch ofrece un alto nivel de aislamiento y seguridad. Este atributo es fundamental en entornos de virtualización para garantizar la protección de las MVs y la red en su conjunto. La capacidad de aislar y salvaguardar el tráfico de las MVs contribuye a mantener la integridad y la confidencialidad de los datos, lo que resulta esencial en escenarios donde la seguridad de la información es prioritaria [16].

Open vSwitch es una tecnología esencial en entornos de virtualización que permite la creación y gestión de redes virtuales dentro de MVs. Al ofrecer segmentación, aislamiento y flexibilidad, Open vSwitch desempeña un papel crucial en la virtualización de redes. Sin embargo no está exenta de desafíos, como por ejemplo la gestión eficiente de los recursos de red, la escalabilidad o la integración con nuevas tecnologías de SDN.

1.2.5. Protocolo LLDP

El Protocolo de Descubrimiento de Dispositivos de Enlace (Link Layer Discovery Protocol—LLDP) es un protocolo estándar, definido en la capa 2 del modelo Open Systems Interconnection (OSI), que permite a los dispositivos intercambiar información sobre sus enlaces y sobre sí mismos a través de tramas Ethernet [1], que contienen los datos más importantes del dispositivo emisor, como son su nombre, su dirección MAC, las capacidades y el tipo de puerto [6]. Al recibir estos paquetes, los equipos son capaces de extraer esta información, almacenarla, conocer cuales son sus máquinas vecinas y establecer caminos entre las máquinas del escenario [3].

Este protocolo se puede utilizar para una variedad de propósitos prácticos, entre los que se incluyen:

- **Descubrimiento de dispositivos**: LLDP puede utilizarse para descubrir los dispositivos conectados a una red. Esta información puede utilizarse para crear inventarios de dispositivos y para simplificar la configuración de la red.
- **Mantenimiento de la topología de red**: LLDP puede utilizarse para mantener una visión actualizada de la topología de red.

- **Resolución de problemas**: LLDP puede utilizarse para ayudar a resolver problemas de red, como por ejemplo identificar el camino entre dos dispositivos o para determinar el estado de un enlace [8].

Todos estos propósitos permiten establecer enlaces y rutas entre los equipos, ya que mantiene actualizado el mapa de rutas de todos los elementos presentes en el escenario. Sin embargo, este protocolo presenta algunas deficiencias, como por ejemplo la falta de seguridad, ya que envía la información sin cifrar o el uso excesivo de paquetes.

El uso de LLDP con OpenFlow obliga a enviar un mensaje diferente desde el controlador a cada uno de los dispositivos de la red de forma periódica. A medida que aumenta el número de dispositivos de red, el número de mensajes crece y los enlaces más cercanos al controlador se pueden ver saturados por la gran cantidad de mensajes LLDP.

1.3. Estructura de la memoria

La estructura de la memoria del trabajo se compone de los siguientes capítulos:

- **Introducción**: Se realiza una descripción básica del proyecto y se exponen las ideas principales del mismo. Aparte, también especifica la estructura de la memoria. Se introducen las tecnologías y protocolos que van a utilizar el desarrollo del trabajo.
- **Objetivos**: En este capítulo se abordan los objetivos tanto generales como específicos que se han seguido para el desarrollo del proyecto así como una planificación temporal el mismo.
- **Diseño e Implementación**: Se describe el trabajo experimental realizado para estudiar y analizar el funcionamiento del plano de control in-band Periplus. Se detallan los escenarios de red utilizados y los procesos que hay que llevar a cabo para arrancar cada uno de los escenarios de red. En este capítulo se describe el protocolo diseñado para sustituir a LLDP, así como su implementación.
- **Plan de Pruebas y Resultados**: Se muestran las pruebas que se han llevado a cabo en los dos escenarios empleados para demostrar que la modificación ha sido implementada con éxito.
- **Conclusiones**: Capítulo dedicado a repasar los objetivos cumplimentados durante el desarrollo del proyecto, los conocimientos aplicados y aprendidos durante el mismo y, por último, estudiar posibles líneas de trabajo futuras para el proyecto.

Capítulo 2

Objetivos

2.1. Objetivo General

El principal objetivo de este Trabajo de Fin de Grado es diseñar, implementar y probar un algoritmo que sustituya LLDP de forma que se envíe por inundación un único paquete desde el controlador de una red SDN y que llegue a todos los switches para que estos puedan responder al controlador con los enlaces que tiene cada dispositivo y el controlador pueda conocer la topología completa de la red.

2.2. Objetivos Específicos

El objetivo general se puede descomponer en una serie de objetivos más concretos, que van surgiendo según se desarrolla el trabajo y que abarcan el desarrollo completo del objetivo general. Estos objetivos específicos son:

- **Aprender el funcionamiento de los escenarios Periplus**: Realizar un trabajo de estudio experimental para aprender cómo funciona el plano de control in-band Periplus, diseñando para ello varios escenarios de red en los que probar el algoritmo que se desea desarrollar.
- **Aprender el funcionamiento del protocolo LLDP**: Este protocolo busca descubrir la topología de la red. En el contexto del controlador Periplus, LLDP permite que el controlador conozca la topología completa y calcule los caminos alternativos desde cada switch al controlador y viceversa.
- **Diseñar un algoritmo que reduzca el número de paquetes en la red**: Actualmente, el controlador genera un paquete diferente para cada switch del escenario para poder usar LLDP y realizar el descubrimiento de la red. Se desea diseñar un nuevo mecanismo de descubrimiento que reduzca estos envíos a un único paquete, que es propagado por todo el escenario a través del método de inundación

controlada. De esta forma, se reduce la cantidad de paquetes de control presentes en el escenario pudiendo usar ese ancho de banda para la transmisión en el plano de datos.

2.3. Planificación Temporal

El desarrollo de este proyecto dio comienzo en enero de 2023 y está dividido en varias etapas:

1) **Fase de estudio**: Consiste en repasar los conceptos teóricos y prácticos de redes de ordenadores que se han estudiado a lo largo de todo el grado y que permiten desarrollar el trabajo exitosamente, además de introducir el entorno de emulación práctico en el que se van a probar los escenarios. Esta etapa tiene una duración de unos 3 meses aproximadamente y abarca el periodo entre principios de febrero y mediados de mayo.

2) **Fase de desarrollo**: Durante este periodo se desarrolla e implementa el software necesario para completar el trabajo, siempre manteniendo una comunicación constante con los profesores para aclarar cualquier duda. En este periodo también se empieza con el desarrollo de la memoria del primer capítulo. Esta etapa abarca desde mayo hasta finales de octubre.

3) **Fase de pruebas y obtención de resultados**: En el mes de noviembre se realizaron las pruebas y se obtuvieron las conclusiones, a la vez que se realizaba y completaba la memoria.

4) **Fase de documentación**: Los posteriores meses de diciembre del año 2023 y enero del 2024 se dedicaron a realizar correcciones y modificaciones de la memoria y del plan de pruebas.

En la imagen inferior se puede observar un diagrama de Gantt que muestra el desarrollo del trabajo a nivel visual.

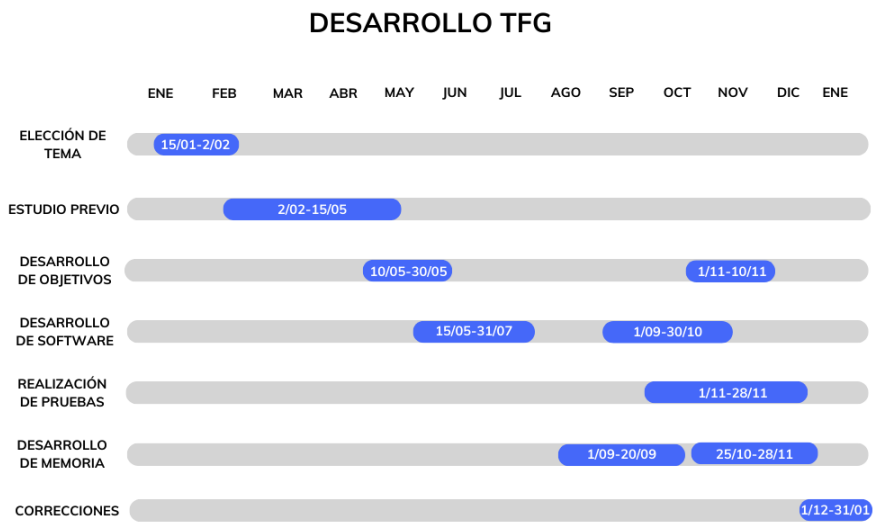

Figura 2.1: Diagrama de Gantt del desarrollo del TFG.
Fuente: elaboración propia.

Capítulo 3

Diseño e Implementación

En este capítulo se describe el entorno y herramientas utilizadas para el diseño y desarrollo de este Trabajo Fin de Grado. Sobre dicho entorno se realizarán las modificaciones del controlador Periplus[1] para implementar un mecanismo alternativo al descubrimiento que utiliza LLDP y se realizarán las pruebas que muestren el correcto funcionamiento.

3.1. Escenarios y diseño del algoritmo

Para desarrollar de forma completa el nuevo protocolo que se quiere implementar, es necesario definir 2 escenarios virtuales diferentes:

- **Escenario básico**: Es un escenario simple compuesto por un único controlador, encargado de gestionar una subred compuesta por 8 switches. Este escenario, el cual recibe el nombre de bucle4_bfd, se emplea para demostrar que el nuevo algoritmo es capaz de descubrir la topología completa de la red y establecer las rutas secundarias, reducir el número de paquetes de control y evitar la explosión de tráfico al realizar la inundación de mensajes.
- **Escenario con 2 controladores**: Es un escenario compuesto de 2 controladores de Periplus y 8 switches que forman dos subredes vecinas conectadas entre sí. En este escenario, llamado 2_controllers, se demuestra que el nuevo algoritmo es capaz de mantener las conexiones entre subredes vecinas gestionadas por controladores diferentes.

En ambos escenarios, la máquina virtual donde se está ejecutando el programa controlador también ejecuta un switch virtual al que denominaremos switch del controlador o switch raíz.

[1] El código fuente de Periplus modificado por el autor de este TFG está disponible contactando con los tutores.

Aunque parezcan dos escenarios totalmente diferentes, en realidad el segundo escenario es una extensión del primero, por lo tanto, la mayor parte del software utilizado es el mismo. Sin embargo, ambas topologías presentan ciertas diferencias a la hora de arrancarse o de monitorizar su actividad, lo cual hace que sea necesario dividir este capítulo en 3 apartados diferentes:

a) **Creación de un escenario en Mininet**: que explica como se crean los escenarios Mininet y como se definen los elementos de los escenarios, como un controlador o un switch (es el software común).

b) **Escenario básico**: explica como funciona un escenario con un único controlador.

c) **Escenario con 2 controladores Periplus.**

A través de la interfaz de programación en Python que ofrece Mininet podremos definir estos dos escenarios.

El desarrollo completo de todos los elementos citados se encuentra en el anexo de este trabajo. A lo largo de todas las secciones se muestran solamente ejemplos o pequeñas partes del código para hacer más fácil la compresión de los conceptos.

3.1.1. Creación de un escenario en Mininet

Antes de empezar el desarrollo del nuevo protocolo, es necesario conocer el funcionamiento de Periplus y de los escenarios de red con los elementos comunes a nivel de software y qué procesos los componen, como por ejemplo como se define un switch dentro de estos entornos virtualizados o las conexiones que se producen entre estos. Utilizaremos la interfaz de programación en Python que ofrece Mininet para la definición de los escenarios.

Para desarrollar un entorno virtual completo, primero es necesario definir los switches. La interfaz de programación que ofrece Mininet proporciona una forma de definir switches a partir de la función addSwitch. Esta función establece un switch dentro del espacio de nombres de red raíz de la máquina donde se ejecuta Mininet.

En el desarrollo de este proyecto no se utiliza dicha función ya que se desea emular un comportamiento de switch que se ejecute dentro de un entorno virtualizado diferente del espacio de nombres de red raíz. Por ello, se utilizará la función addHost para crear este espacio de nombres diferente, en el que posteriormente arrancaremos un Open vSwitch. A continuación se muestra cómo se crea un espacio de nombres para un switch al que denominaremos 's0'. Análogamente se crearán el resto de switches del escenario.

```
1 s0 = net.addHost('s0', ip='0.0.0.0.', mac='00:00:00:00:00:01')
```

Una vez definidos todos los switches del escenario, es necesario crear los enlaces que los conecten. Para crear dichas conexiones se especifica la máquina origen,

la máquina destino y sus correspondientes interfaces Ethernet. Es importante tener en cuenta que la asignación de las interfaces Ethernet, que se identifican como puertos en OpenFlow, comienza la enumeración en 1; sin embargo, a nivel de Ethernet se mantiene el valor inicial en 0, por tanto, s0-eth0 sería la primera interfaz Ethernet de s0 que se corresponde con el puerto 1 de s0, s0-eth1 sería el puerto 2 de s0 y así sucesivamente. En los diagramas de topología siempre se incluye el número de puerto para facilitar la comprensión de las configuraciones en OpenFlow.

En la siguiente línea se puede observar como se establece el enlace desde el puerto 3 de s1 hasta el puerto 1 de s5, es decir, se está definiendo la conexión entre s1-eth2 y s5-eth0.

```
1 net.addLink( s1, s5, 2, 0 )
```

Una vez definida la topología de conexión del escenario, es necesario arrancar los switches en los espacios de nombres en los que se han creado y establecer algunas configuraciones previas. Esta configuración se realiza a través del script start_controller _switch_bucle4_bfd.sh, que ejecuta la configuración inicial del entorno a través de otros dos scripts adicionales:

- **"Start-controler-switch-n-ifaces.sh"**: Se emplea para el switch raíz (el que se encuentra en el switch del controlador), es necesario pasarle como argumentos el nombre del switch, su número de interfaces y sus direcciones IP anycast e IP (10.0.0.1 y 11.0.0.1 respectivamente).

  La dirección IP anycast es la que usan todos los switches de una topología para comunicarse con el controlador (que se encuentra en el mismo espacio de nombres que el switch raíz). Todos los controladores tendrán configurada esta dirección IP anycast. La dirección IP adicional se utiliza para distinguir la máquina donde se ejecuta cada controlador y será diferente para cada controlador.

- **"Start-switch.sh"**: Se utiliza para el resto de switches, siendo necesario pasarle como argumentos el nombre del switch, su número de interfaces y sus dirección IP.

La ejecución de este archivo instala en todos los elementos presentes en el escenario una pequeña configuración inicial, permitiendo así que se puedan enviar los primeros paquetes OpenFlow que intercambian el controlador con cada uno de los switches que gestiona. Periplus configurará el funcionamiento de los switches a través de dichos mensajes, instalando el comportamiento de reenvío en sus tablas de flujos.

3.1.2. Escenario básico

En este primer escenario se muestra una topología con un único controlador y con caminos alternativos bidireccionales, desde los switches hasta el controlador.

En la figura 3.1 se muestra la topología creada para este escenario básico. Se crearán 8 switches (de s1 a s8), el switch raíz c0 que se encuentra en la máquina donde se ejecuta el controlador y 2 máquinas h2 y h3 (no se utilizan en el ámbito de este trabajo). Los números que aparecen en cada enlace se corresponden con los números de puerto de cada Open vSwitch. Por ejemplo, el switch s1 y s2 están unidos de la siguiente forma: desde el puerto 2 de s1 al puerto 1 de s2.

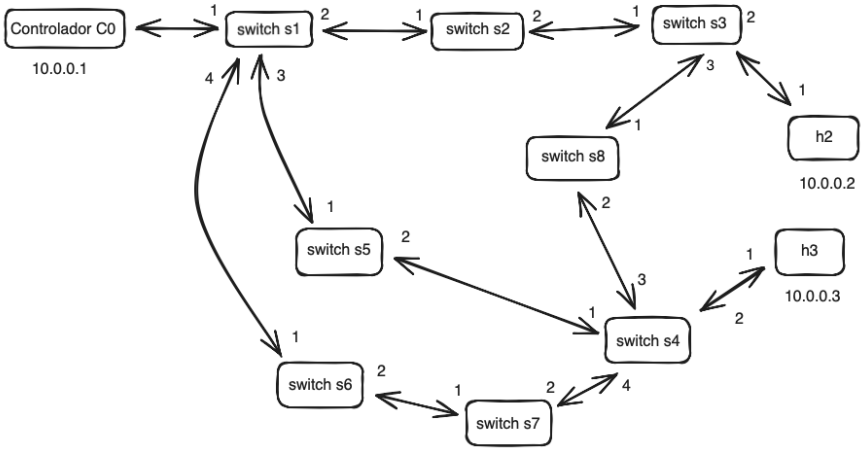

Figura 3.1: Topología escenario básico Periplus.
Fuente: elaboración propia.

El controlador Periplus instala reglas en los switches que implementan un mecanismo de encaminamiento basado en origen que añade una cabecera adicional situada entre las ya existentes Ethernet e IP, Network Service Header (NSH). Esta nueva cabecera, generada por el dispositivo emisor del mensaje, establece la ruta que sigue la conexión entre el origen y el destino, indicando los puertos de salida de cada uno de los switches, a través de los cuales el paquete llega a su destino. Esta nueva cabecera también es consultada por los switches intermedios, permitiéndoles reenviar la información a través del puerto de salida adecuado.

En esta cabecera NSH se incluye un camino principal y caminos alternativos por si hubiera algún fallo en un dispositivo intermedio o en algún enlace del camino principal.

El objetivo central de estudiar esta cabecera radica en observar los distintos cami-

nos alternativos en una ruta determinada ya que los caminos alternativos se calculan basándose en la información aprendida por LLDP. Demostraremos que, tanto el nuevo algoritmo como el protocolo LLDP realizan el descubrimiento de todos los enlaces del escenario y con esta información se pueden configurar tanto el camino principal como los alternativos.

Para poder completar la configuración del escenario y capturar los paquetes, es necesario arrancar el escenario[2]. Una vez arrancados el controlador y los switches, comienza el proceso de bootstrap de Periplus, que permite que los dispositivos puedan comunicarse automáticamente con el controlador. Este proceso consta de una serie de pasos:

- **Arranque del Switch**: Se inicia el software Open vSwitch con una configuración inicial básica que sólo permite una comunicación mínima, entre switches vecinos.
- **Mensajes solicitud de ARP**: El switch que está arrancando, necesita establecer una conexión TCP para usar OpenFlow. Por tanto, el switch necesita solicitar la dirección Ethernet del controlador. Si el switch está conectado directamente con el controlador, éste responderá y el controlador podrá configurar el comportamiento en dicho switch a través de OpenFlow. Si no está directamente conectado con el controlador, deberá esperar a que un switch vecino esté ya gestionado por el controlador para que le permita conectarse.
- **Conexión con el controlador**: el switch establece una sesión OpenFlow con el controlador. El controlador procesa estos paquetes para identificar el switch que está iniciando y su ubicación en la topología.
- **Instalación de Reglas**: Basándose en la información obtenida, el controlador instala reglas en dos switches de la red, en el switch raíz que permite la comunicación entre el controlador y el switch que acaba de arrancar y en el switch que acaba de arrancar que permite la comunicación en el sentido contrario. El switch pasará a estado gestionado.
- **Descubrimiento de Enlaces Restantes con LLDP**: Periplus utiliza el protocolo de descubrimiento de enlaces LLDP para identificar y completar la información sobre los enlaces secundarios y sobre rutas alternativas para alcanzar todos los switches de la topología.
- **Finalización del Bootstrap**: Con la instalación de reglas y la información completa sobre la topología de la red, el nuevo switch está ahora completamente gestionado por el controlador y puede participar en el reenvío de paquetes según las reglas OpenFlow establecidas por el controlador.

[2]El proceso para arrancar y apagar el escenario se encuentra en el apartado Anexo.

Este proceso automatizado facilita la incorporación de nuevos switches a la red, reduciendo la necesidad de intervención manual. Además, LLDP garantiza un descubrimiento completo de la topología de la red.

A lo largo de todo este capítulo se desarrolla una alternativa al uso de LLDP para el descubrimiento de la topología. El objetivo es enviar un único paquete que permita conocer todos los enlaces que hay en el escenario de red. Este paquete es creado por el controlador, que a través del proceso de inundación, lo distribuye a través de todos los switches de la subred, actualizando así los enlaces entre los equipos y las rutas secundarias hacia el controlador. Periódicamente el controlador generará este paquete para que se distribuya a través de todos los enlaces de la red y permita obtener información actualizada del estado de los nodos y enlaces. Este nuevo algoritmo también permite al controlador mantener y establecer nuevos los caminos a las subredes vecinas a través de sus switches frontera en escenarios con varios controladores.

En la figura 3.2 se puede ver el diseño del nuevo algoritmo en el que el controlador envía un sólo paquete C-Adv (Controller Advertisement) que se propaga a través de todos los enlaces usando un mecanismo de inundación controlada. Cada switch envía como respuesta al controlador un paquete OpenFlow de tipo Packet-In informando de la interfaz por la que ha recibido C-Adv. De esta forma el controlador puede descubrir todos los enlaces que hay en la topología.

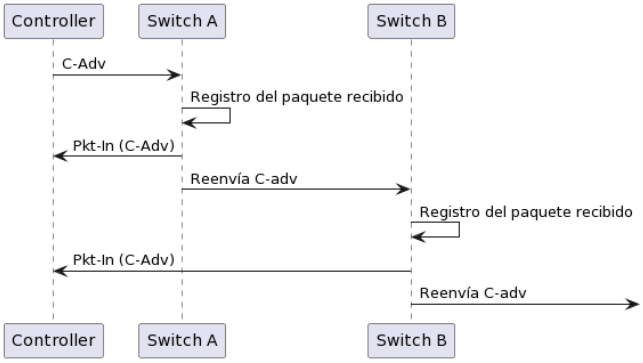

Figura 3.2: Diseño de protocolo alternativo a LLDP para descubrimiento de enlaces. Fuente: elaboración propia.

Esta nueva implementación también evita el reenvío del paquete a switches que ya lo hayan recibido, a través de un sistema de registro de paquetes, que se realiza en el switch que recibe el C-Adv y se actualiza cada cierto tiempo. De esta forma, si el switch recibe el paquete varias veces dentro del mismo ciclo, este se descarta y no se reenvía a los equipos vecinos, evitando así cargar la red con paquetes innecesarios. Por este motivo, lo llamamos inundación controlada.

En la figura 3.3 se puede observar gráficamente sobre la topología del escenario el camino principal y el alternativo desde s5 al controlador. Las flechas de color rojo corresponden a los saltos que realiza el camino principal hacia el controlador y las flechas de color azul corresponden a los saltos del camino alternativo.

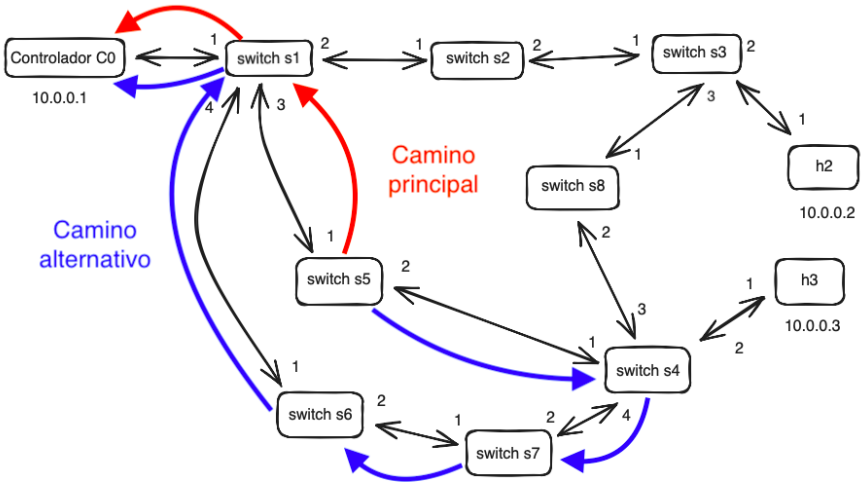

Figura 3.3: Representación gráfica del camino principal y el alternativo desde s5 al controlador.

Fuente: elaboración propia

Para poder comprobar que esta nueva funcionalidad se ha implementado con éxito, desactivamos LLDP y comprobamos que se muestran los caminos alternativos en las cabeceras NSH. Los caminos alternativos se habrán calculado partiendo de la información recibida de los Packet-In(C-Adv) procesados por el controlador. La primera cabecera NSH contendrá la ruta principal, y para cada uno de los saltos, en caso de que exista una ruta alternativa, se añadirá una cabecera NSH adicional que describirá la ruta secundaria correspondiente. Una ruta dentro de una cabecera NSH queda descrita por la lista de puertos de salida por los que hay que reenviar un paquete para llegar a su destino.

La información sobre los puertos de salida se codifica en la cabecera NSH empleando 4 bits: ABCD, donde:

- Los 3 bits de menor significancia, en este caso BCD, codifican el número de puerto de salida en un switch, siendo este un valor comprendido entre 0 y 7.
- El bit más significativo, el bit A, indica si el puerto tiene una ruta alternativa. Si A=0, el puerto no muestra ruta alternativa, pero si A=1, dicho puerto tendrá una ruta alternativa que será incluida en una cabecera NSH adicional.

- El último puerto del camino siempre está codificado como f' y número de puerto, para indicar que es el último salto antes de alcanzar el destino. Por ejemplo, 0xf9 indica que es el último salto y que el puerto por donde debe reenviarse el paquete es el puerto 1. Si este puerto no estuviera activo, existe un camino alternativo descrito en otra cabecera NSH adicional.

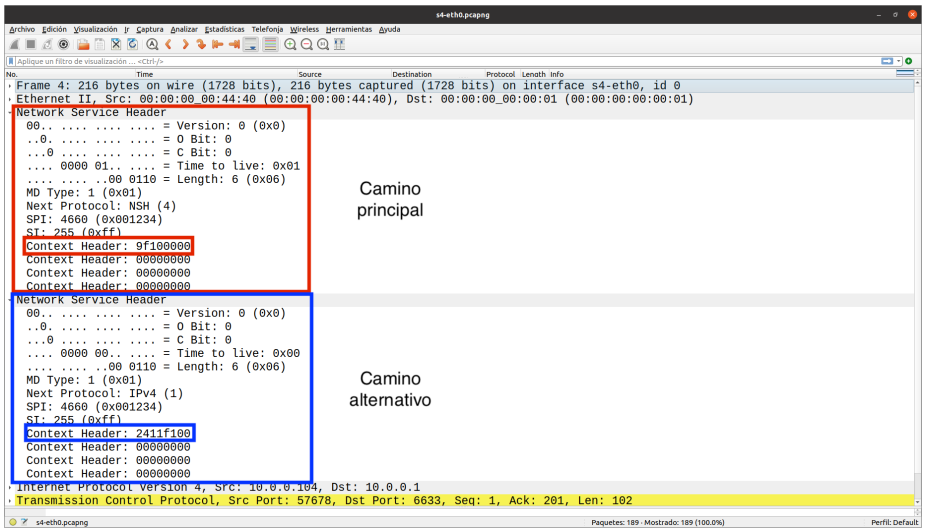

Figura 3.4: Ejemplo de cabeceras NSH con un camino principal y otro alternativo. Fuente: elaboración propia

En la figura 3.4 podemos observar un ejemplo de dos cabeceras NSH que contienen el camino principal y el alternativo. El paquete está generado por s4 y se dirige al controlador. Se ha capturado en el puerto 1 de s4 (véase la figura 3.3). En ese enlace el paquete lleva un camino principal s5-s1-s0 que se corresponde con la codificación 9f1 (véase la cabecera NSH marcada en rojo en la figura 3.4). Se debe tener en cuenta, que el puerto de salida de s4 no está incluido en la ruta, ya que las capturas se realizan en los enlaces entre nodos, y en consecuencia el puerto saliente (el de s4 en este caso) ya ha sido utilizado y eliminado de la cabecera NSH del camino principal.

Por tanto, el camino principal codificado como 0x9f1 representa: 9 (1001) que es el puerto 1 de s5 y tiene camino alternativo por estar activado el bit más significativo. Si el puerto 1 de s5 no está activado, se utilizará el camino alternativo. Si el puerto 1 está funcionando el mensaje se reenviará a través de dicho puerto llegando al switch s1. En este caso, s1 utilizará su puerto 1 para llegar hasta el controlador. Véase como el último salto de un camino siempre se codifica como 'f' y número de puerto. Si el puerto 1 de s5 no está activado, se utilizará el camino alternativo, en este caso 0x2411f1 (véase cabecera NSH marcada en azul en la figura 3.4). Es decir, los paquetes seguirán el camino: puerto 2 de s5, puerto 4 de s4, puerto 1 de s7, puerto 1 de s5 y puerto 1 de

s1 siendo éste el último puerto.

Nótese que si el algoritmo de descubrimiento no funciona, no se descubrirían todos los enlaces de la topología y no existirían caminos alternativos.

3.1.3. Escenario en bucle con 2 controladores

Este segundo escenario muestra una topología compuesta por 2 controladores Periplus, que toman el nombre de c0 y c1 respectivamente. Este nuevo entorno amplía el escenario definido en el anterior apartado añadiendo un nuevo controlador a la red tal y como se muestra en la figura 3.5. Llamaremos 'subred' al conjunto de dispositivos controlados por un determinado controlador. En este caso, con 2 controladores, una vez que todos los switches hayan arrancado, quedarán definidas 2 subredes que contienen cada una de ellas los dispositivos gestionados por cada controlador.

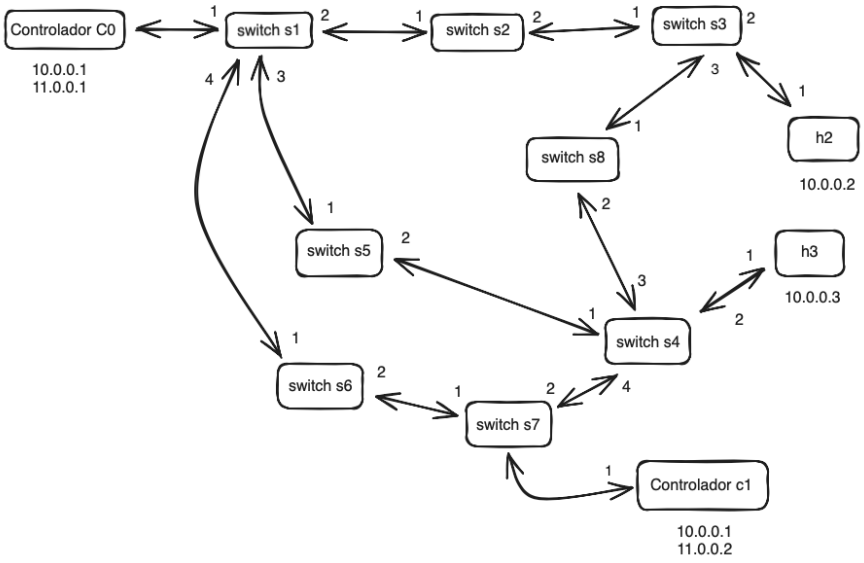

Figura 3.5: Topología escenario con 2 controladores.
Fuente: elaboración propia

La inclusión de este nuevo controlador, conectado a s7, redefine la distribución de las subredes, de tal forma que ambos controladores se disputan el control de los switches presentes en el escenario. Esto provoca la aparición dos redes o zonas independientes, que se ven obligadas a comunicarse entre ellas (a lo largo de este apartado se describe esta interacción y cuáles son las modificaciones que sufren con el nuevo algoritmo).

El proceso de arranque de este nuevo escenario es similar al escenario anterior y está descrito en el Anexo.

Para que los switches sepan a cuál de los controladores están conectados, y dado que ambos tienen la dirección 10.0.0.1 configurada, es esencial asignar una dirección adicional a cada controlador: 11.0.0.1 para c0 y 11.0.0.2 para c1. Estas direcciones también permiten la comunicación entre los controladores. Después del proceso de bootstrap, algunos switches quedarán gestionados por c0 y otros por c1.

En este tipo de escenario existirá un conjunto de switches llamados switches frontera, que son aquellos que están conectados a un controlador pero que tienen como vecino a un switch gestionado por otro controlador diferente.

Para que un controlador conozca la existencia de otros controladores, se utilizan también los mensajes UDP periódicos llamados C-Adv, que son enviados a todos los switches con el propósito de identificar la existencia de enlaces (como ya se había visto en el escenario básico) pero también la existencia de otros controladores. Cuando un controlador detecta a otro, establece un camino en su switch raíz que agrega la información necesaria hasta llegar al switch frontera correspondiente. Una vez que el paquete ha llegado al switch frontera, será reenviado a través del enlace que le conecta al switch que pertenece al otro controlador. Este switch conoce el camino hacia su controlador y añadirá las cabeceras NSH necesarias para que el paquete pueda llegar al segundo controlador. Como resultado, los controladores pueden intercambiar información a través de los switches frontera.

Uno de los principales objetivos de esta nueva implementación es mantener esta comunicación a través de los switches frontera. Por eso, es necesario configurar el nuevo algoritmo para que tenga en cuenta las direcciones de los controladores vecinos.

En la figura 3.6 se puede observar un escenario con 2 subredes definidas, en las cuales los switches s1, s2, s3, s4, s5 y s6 actúan como switches frontera.

Para poder comprobar que ambas subredes interactúan a través de mensajes, es necesario consultar las cabeceras NSH, al igual que se hacía en el escenario con un solo controlador.

En la figura 3.7 se puede observar la captura de tráfico entre los switches c0 y s1, que actúan de switches frontera entre la subred 1 y la subred 2 respectivamente. Este tráfico es generado en el controlador c0 con dirección el controlador c1. El paquete en dicha captura ya habrá salido de c0 y por tanto la cabecera NSH mostrará qué es lo que tiene que hacer s1.

En las cabeceras NSH se puede ver el camino principal desde c0 a c1 es 0xfc, es decir, f por ser el último salto y c (1100) porque s1 debe reenviar el paquete por su puerto 4, en dirección a s6. Ahí termina el camino que c0 incluye en el paquete para llegar al router frontera de su zona. Adicionalmente si s1 no tiene su puerto 4 activo,

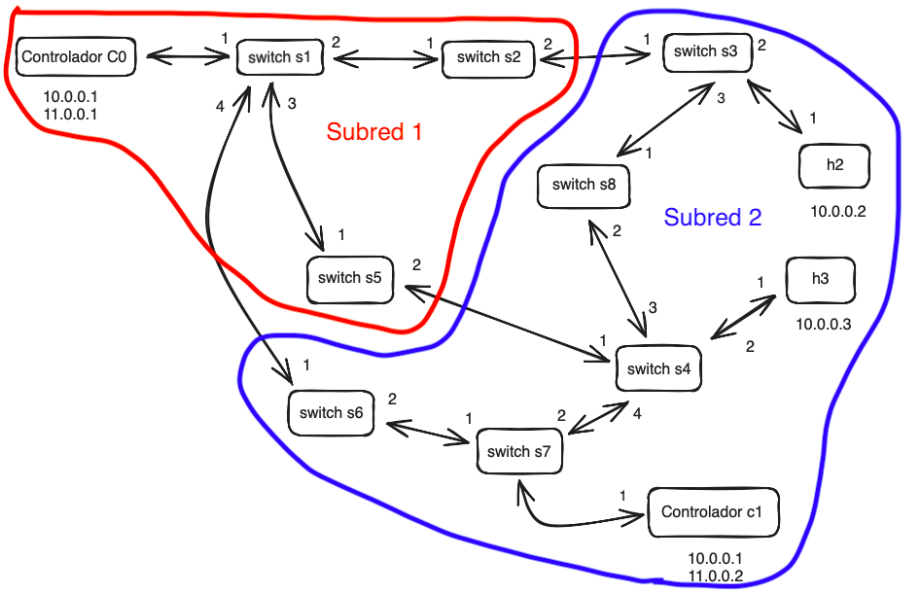

Figura 3.6: Representación de dos subredes o zonas gestionadas por 2 controladores. Fuente: elaboración propia

hay configurado un camino alternativo para llegar a c1: 0x3f2. Con este camino alternativo s1 lo enviaría a través de su puerto 3 hacia s5 y s5 lo reenviaría a través de su puerto 2, llegando a s4. En la figura 3.8 se puede observar de una manera gráfica el camino principal y el alternativo desde el controlador de la subred 1 hacia la subred 2.

Tanto si se utiliza el camino principal como el alternativo, el mensaje llegará a un switch que está gestionado por el controlador c1 y allí se compondrá una nueva cabecera NSH para alcanzar c1.

En la figura 3.9 se puede observar una captura realizada en el puerto 1 de s6. Estos mensajes no portan ya ninguna cabecera NSH, ya que han completado el camino que el controlador c0 había definido para llegar al switch frontera. Por el contrario, en la figura 3.10 se puede observar la captura realizada en el puerto 2 de s6, donde se muestra la cabecera incluida por el switch s6 para poder llegar al controlador c1. En este caso, los paquetes se encuentran viajando desde s6 hacia s7. Al llegar a s7 la cabecera NSH con valor 0xf3 indica que s7 es el último salto y que deberá reenviar el paquete a través del puerto 3 de s7, para así llegar a c1.

De esta forma se puede comprobar que las dos subredes vecinas pueden establecer comunicación entre ellas y son capaces de enviar información de un controlador a otro.

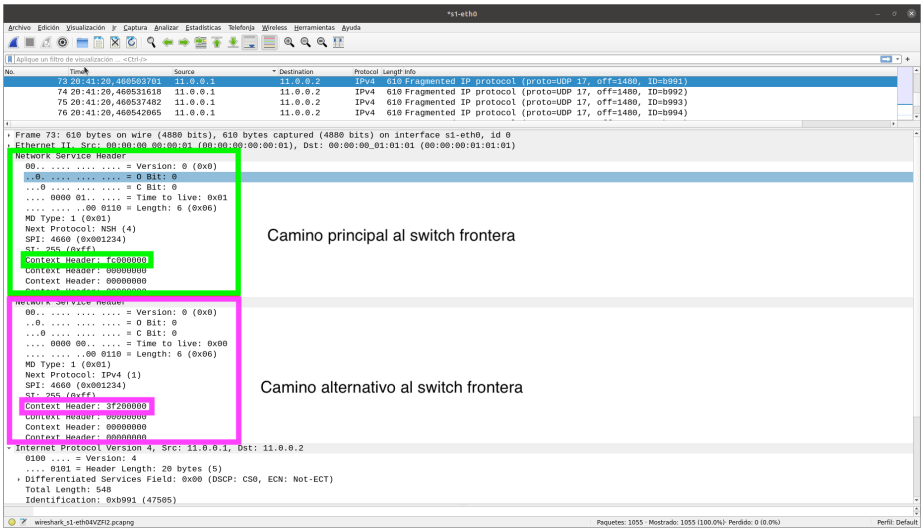

Figura 3.7: Ejemplo de cabeceras NSH en la comunicación entre dos controladores.
Fuente: elaboración propia

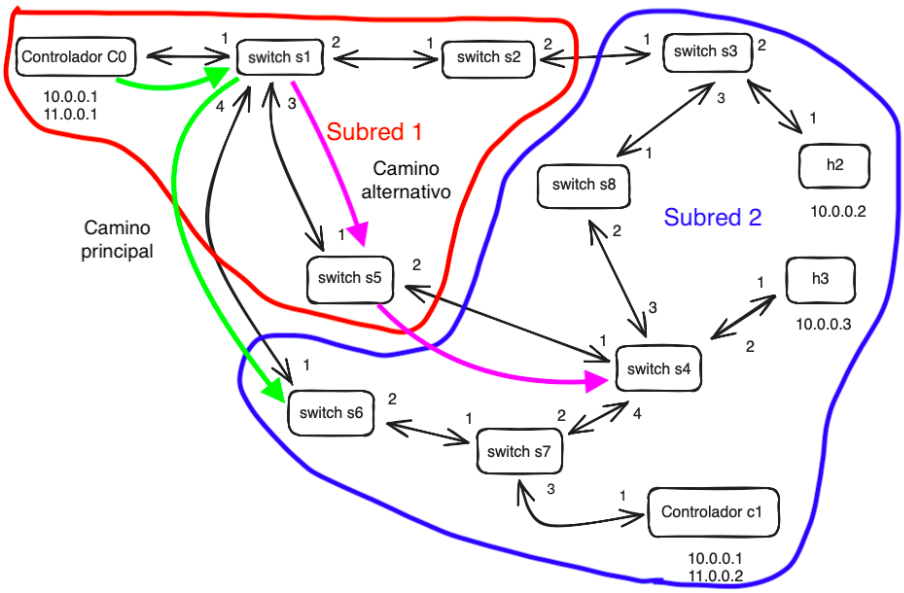

Figura 3.8: Rutas desde c0 al switch frontera en la comunicación desde c0 hacia c1.
Fuente: elaboración propia

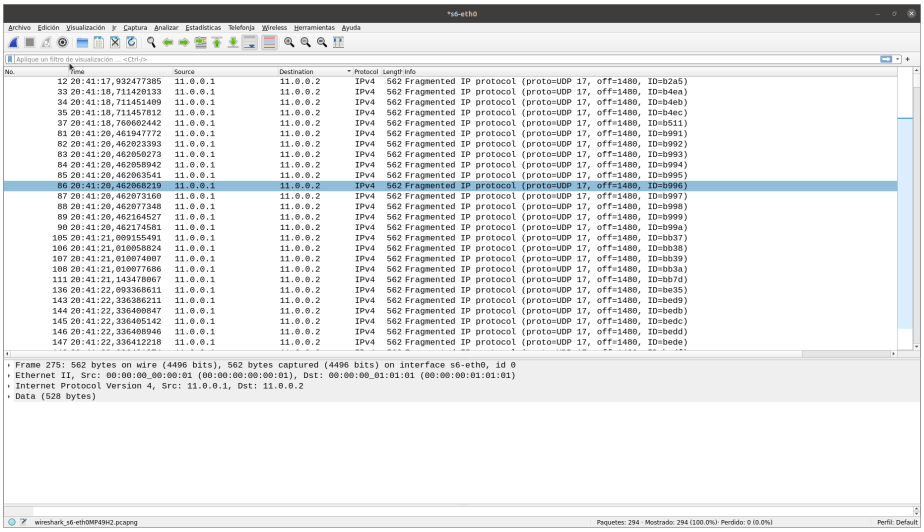

Figura 3.9: Captura de tráfico en el switch frontera s6 en su puerto 1 para la comunicación desde c0 a c1.

Fuente: elaboración propia

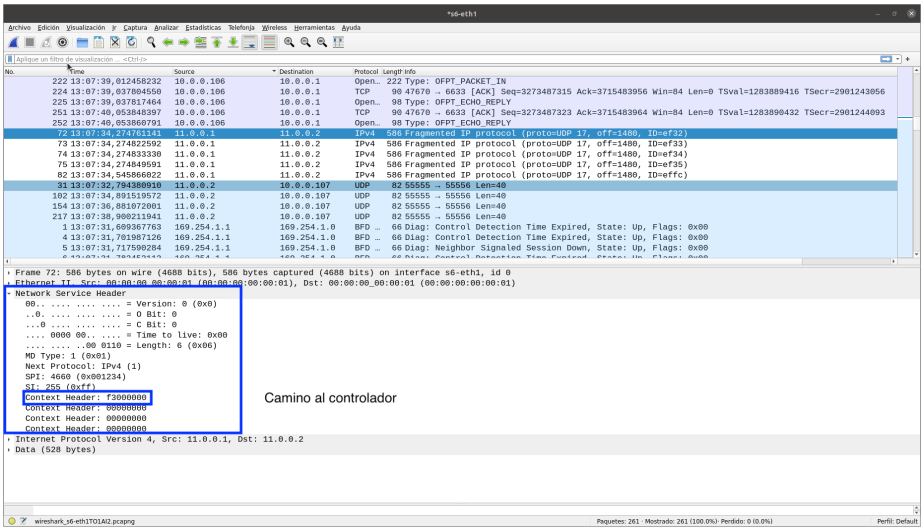

Figura 3.10: Captura de tráfico en el switch frontera s6 en su puerto 2 para la comunicación desde c0 a c1.

Fuente: elaboración propia

3.2. Implementación del algoritmo

Para afrontar el cambio de LLDP al nuevo mecanismo, es necesario modificar el fichero llamado "InBandController.py". Este archivo, desarrollado en Python, es la implementación del controlador Periplus que permite el boostrap de la red SDN y aprende las rutas alternativas a través de LLDP. Se desactivará el funcionamiento de LLDP y se incluirá el nuevo comportamiento.

Esta nueva implementación se basa en el envío de paquetes C-Adv que se envían por UDP desde el controlador a los switches, utilizando los puertos 55555 y 55556 como origen y destino respectivamente. Su función principal es el descubrimiento de nuevos enlaces entre los dispositivos y el descubrimiento de controladores.

Cuando un switch recibe un paquete C-Adv, genera un mensaje Packet_In que contiene el C-Adv recibido e información adicional sobre el switch, como su ID y los puertos conectados.

Al recibir un Packet_In con un C-Adv, el controlador extrae la dirección IP del switch que lo envió y la información del enlace incluida en dicho C-Adv. El controlador actualiza la topología de la red si ha recibido información nueva. De esta manera, el controlador puede mantener una imagen actualizada de la red sin necesidad de configuración manual.

En la figura 3.11 se muestra un esquema de como el controlador envía los paquetes C-Adv a los switches y estos le responden con un Packet_In para actualizar su topología.

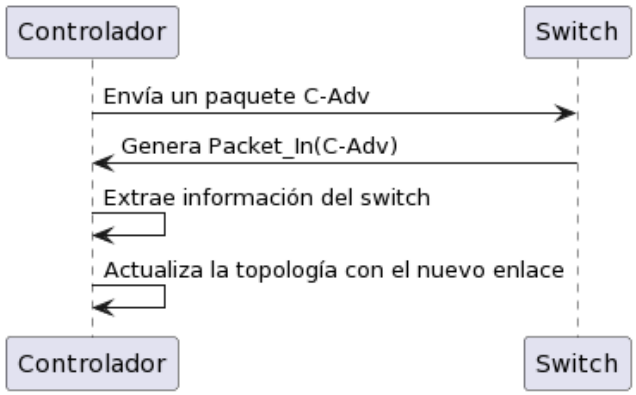

Figura 3.11: Esquema de detección de enlaces basado en paquetes C-Adv.
Fuente: elaboración propia.

Las modificaciones realizadas implementan el siguiente comportamiento:

1) Envío de un único mensaje desde el controlador al switch raíz periódicamente

(funciones send_C_Adv y send_periodic_C_Adv).

2) Instalación de flujos en los switches para definir cómo deben comportarse al recibir un paquete C-Adv (función install_flows_to_process_C_Adv). Estos flujos implementan la inundación controlada y el envío del Packet_In con el C-Adv recibido al controlador.

3) Acciones que debe realizar el controlador cuando reciba el Packet_In que contiene C-Adv (función packet_in_handler).

3.2.1. Funciones send_C_Adv y _send_periodic_C_Adv

En la función send_C_Adv el controlador manda un mensaje C-Adv al switch raíz del controlador.

La función inicia obteniendo una lista de nodos de la topología. Luego, itera sobre estos nodos, para extraer únicamente el switch del nodo raíz, y para ése genera el C-Adv. Un único mensaje.

En el cuadro inferior se puede observar como se realiza este envío.

```
1  nodes = self.config.nodes()
2
3  # Manda C-Adv
4  for n in nodes:
5      # Controller
6      if n==self.CONTROLLER_IP+"CONTROLLER":
7          continue
8
9      # Root switch
10     if n in self.config.DG().nodes.keys() and "datapath" in
           self.config.DG().nodes[n].keys() and n == self.
           CONTROLLER_IP:
11         # Send C-Adv to controller's switch to discover new
               controllers
12         self._send_probe_controller(self.config.getProperty(n,
               "datapath"))
```

Para poder realizar este envío de C-Adv de manera periódica se define la función «_send_periodic_C_Adv», que repite el proceso cada 2 segundos, este periodo se encuentra definido en la variable PERIOD_SEND_C_ADV. Es decir, cada 2 segundos el controlador envía un C-Adv que inunda de forma controlada la red de switches que este controlador gestiona. Este envío periódico es necesario para poder tener actualizada la información de los enlaces que hay activados en cada momento.

En el cuadro inferior se puede observar la función «_send_periodic_C_Adv» definida en Python.

```
def _send_periodic_C_Adv(self):
        while True:
            self.info_log("\n#### _send_periodic_C_Adv\n")

            self.send_C_Adv()

            time.sleep(self.PERIOD_SEND_C_ADV)
```

Como ya se menciona en la introducción de este capítulo, el controlador envía periódicamente estos paquetes UDP a los switches utilizando los puertos 55555 y 55556 como origen y destino respectivamente. Al recibir un C-Adv, el switch genera un mensaje Packet_In que encapsula el C-Adv y lo envía de vuelta al controlador, que analiza el Packet_In y extrae la información del switch, como su dirección IP, para actualizar su tabla de topología con la información del enlace.

3.2.2. Función install_flows_to_process_C_Adv

Esta función define los flujos que el controlador instalará en los switches para la gestión de los C-Adv. Cada vez que un switch reciba un C-Adv deberá ejecutar una serie de acciones en función de diferentes situaciones:

- **Un nuevo C-Adv de mi controlador**: Con este flujo aprende el puerto en el que se recibió el mensaje, y luego envía un mensaje Packet-In al controlador. Esto permite al controlador actualizar la información de dicho puerto si todavía no lo tenía aprendido. Además, como es la primera vez que procesa este mensaje el switch debe enviarlo por todos sus puertos salvo por donde lo ha recibido, realizando la inundación.
- **Un C-Adv repetido de mi controlador**: Es un C-Adv que ha sido procesado ya, es decir, que hace menos de 2 segundos que este switch ya procesó ese C-Adv de su controlador. Por tanto, la recepción de este mensaje es debido a un bucle en la red que hay que parar. Este flujo detecta que el switch ya ha inundado el escenario con este mensaje a través de todos los puertos, y no hay necesidad de enviarlo nuevamente, por lo tanto envía un mensaje Packet-In al controlador, pero no lo reenvía por todos sus puertos.
- **Un C-Adv de un controlador desconocido**: Este flujo aprende el puerto en el que se recibió el mensaje y envía un mensaje Packet-In al controlador. El aprendizaje de este puerto se utiliza en caso de desconexión entre el switch y su controlador, permitiendo tener un puerto de salida activo hacia otro controlador. Por otro lado, cuando el controlador reciba el Packet-In que contiene el C-Adv de otro controlador, descubrirá la existencia de dicho controlador.

Para implementar cada uno de estos flujos, la función se divide en 3 secciones principales.

La primera sección de la función se encarga de crear los criterios de coincidencia que se utilizarán para identificar los C-Adv. Los criterios de coincidencia son una serie de condiciones que deben cumplirse para que un paquete se ajuste a un flujo determinado, en este caso, se busca un paquete que sea de tipo IP, tenga una dirección IP de origen igual a la IP del controlador, sea de protocolo UDP, y tenga los puertos UDP de origen y destino 55555 y 55556 respectivamente. Estas características son las que definen un C-Adv. Además se comprueba que el reg8 es cero porque es el que indica que hace más de 2 segundos que se recibió el último C-Adv. Por tanto, si reg8 es cero, el C-Adv recibido pertenece a un nuevo envío periódico y hay que procesarlo.

Esta coincidencia de flujo se define utilizando la clase 'OFPMatch' de la biblioteca OpenFlow. La clase 'OFPMatch' permite especificar una serie de criterios que deben cumplirse para que un paquete coincida con el flujo.

En el cuadro inferior se puede observar como se transcriben estas condiciones a código Python.

```
flow_match = parser.OFPMatch(reg8=0,
                             eth_type=ether_types.ETH_TYPE_IP,
                             ipv4_src=self.config.controller_ip
                                 (),
                             ip_proto=inet.IPPROTO_UDP,
                             udp_src=self.
                                 CONTROLLER_SRC_PORT_PROBE,
                             udp_dst=self.
                                 CONTROLLER_DST_PORT_PROBE
                            )
```

Una vez que se ha definido la coincidencia de flujo, se desarrollan las 4 acciones que se realizan cuando se cumpla la coincidencia de flujo.

La primera acción instalará una nueva regla en el switch. Esta regla está compuesta por una condición y una nueva acción que se aplicará para los futuros paquetes que se reciban en el switch. Esta regla tendrá como condición comprobar que se recibe un paquete C-Adv y como acción cambiar el valor de reg8 a 1. Este valor en el registro 8 indicará que los C-Adv no deben inundarse.

Nótese que esta regla tiene tiempo de caducidad, es efímera (opción hard_timeout). Este parámetro es necesario porque si no, el switch no volvería a realizar la inundación controlada con ningún mensaje C-Adv. Durante el tiempo de vigencia de esta regla, el switch no inundará con C-Adv recibidos por la existencia de bucles en la topología y eso evitará explosión de tráfico. Como la periodicidad de los C-Adv es cada 2 segundos, se ha establecido como 1 segundo el tiempo de vigencia de esta regla que controla

la inundación, evitando el reenvío de mensajes C-Adv de forma descontrolada.

En este cuadro se muestra esta nueva regla en código Python:

```
flow_actions=[parser.NXActionLearn(table_id=self.
    ACTIVE_PORTS_TABLE,
              specs=[
                  parser.NXFlowSpecMatch(src=ether_types.
                      ETH_TYPE_IP,
                                         dst=("eth_type", 0),
                                         n_bits=16),
                  parser.NXFlowSpecMatch(src=controller_ip_int,
                                         dst=("ipv4_src", 0),
                                         n_bits=32),
                  parser.NXFlowSpecMatch(src=inet.IPPROTO_UDP,
                                         dst=("ip_proto", 0),
                                         n_bits=8),
                  parser.NXFlowSpecMatch(src=self.
                      CONTROLLER_SRC_PORT_PROBE,
                                         dst=("udp_src", 0),
                                         n_bits=16),
                  parser.NXFlowSpecMatch(src=self.
                      CONTROLLER_DST_PORT_PROBE,
                                         dst=("udp_dst", 0),
                                         n_bits=16),
                  parser.NXFlowSpecLoad(src=1,
                                        dst=("reg8", 0),
                                        n_bits=32)
              ],
              hard_timeout=self.TIMEOUT_C_ADV_REG8_LEARN_FLOW,
              priority=40000,
              cookie=self.COOKIE)
]
```

En la figura 3.12 se muestra un esquema de cómo se comportan el switch cuando recibe un C-Adv dependiendo del contenido del registro reg8.

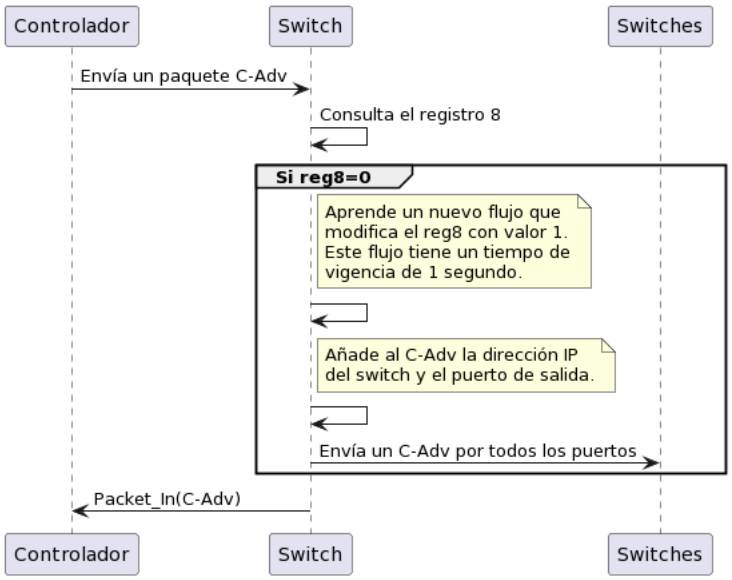

Figura 3.12: Comportamiento de un switch cuando recibe C-Adv.
Fuente: elaboración propia.

Junto a esta regla, el switch implementa otras 3 acciones para procesar los anuncios de control (C-Adv): envía un Packet-In con el contenido del C-Adv recibido al controlador local, posibilitando que dicho controlador actualice la información de sus enlaces. El switch además envía el C-Adv por todas sus interfaces salvo por donde lo ha recibido, pero añade información a ese mensaje: la dirección IP del switch que está reenviando el C-Adv y el puerto de salida por donde lo está enviando. De esta forma cuando el siguiente switch genere el Packet_In al controlador, con el C-Adv recibido, este C-Adv llevará esta información que permitirá identificar ese enlace. Por último el controlador instalará el nuevo flujo en los switches a través de la función add_flow.

En el cuadro inferior se puede observar estas acciones en el código Python:

```
flow_actions += [parser.OFPActionOutput(datapath.ofproto.
    OFPP_CONTROLLER)]
flow_actions += [parser.OFPActionSetField(ipv4_dst=ipv4_src)]
# Envio a todos usando grupo 20
flow_actions += [parser.OFPActionGroup(group_id=self.GROUP_ALL)
    ]

self.add_flow(datapath, 60005, flow_match, flow_actions,
              hard_timeout=self.
                  TIMEOUT_INSTALL_FLOWS_AS_MANAGED_SWITCH,
              table_id=self.INITIAL_TABLE)
```

El flujo que acabamos de describir se utiliza para gestionar la recepción de nuevos C-Adv en un switch. Además, el controlador deberá instalar un flujo especial para los C-Adv antiguos. Se consideran C-Adv antiguos aquellos que ya han sido procesados por el switch, y por lo tanto ya se ha establecido el valor del registro 8 a 1. Por eso la condición para la ejecución de este flujo incluye comprobar que el paquete es un C-Adv y tiene el registro 8 igual a 1.

En estos casos, la acción asociada al flujo es reenviar el paquete al controlador sin aprender un nuevo flujo y sin realizar la inundación, porque ese mensaje ya se ha enviado previamente.

```
flow_match = parser.OFPMatch(reg8=1,
                             eth_type=ether_types.ETH_TYPE_IP,
                             ipv4_src=self.config.controller_ip
                                 (),
                             ip_proto=inet.IPPROTO_UDP,
                             udp_src=self.
                                 CONTROLLER_SRC_PORT_PROBE,
                             udp_dst=self.
                                 CONTROLLER_DST_PORT_PROBE
                             )

flow_actions = [parser.OFPActionOutput(datapath.ofproto.
    OFPP_CONTROLLER)]

self.add_flow(datapath, 59999, flow_match, flow_actions,
              hard_timeout=self.
                  TIMEOUT_INSTALL_FLOWS_AS_MANAGED_SWITCH,
              table_id=self.INITIAL_TABLE)
```

Por último, el controlador debe instalar un flujo en los switches que gestione los paquetes C-Adv que provienen de controladores desconocidos. Los C-Adv de controladores desconocidos son aquellos generados por un controlador que el switch no conoce, uno situado en otra subred.

En estos casos, el controlador utiliza la acción 'NXActionLearn' para almacenar el puerto de entrada del paquete C-Adv en el registro 9[3]. Esta información se utiliza para identificar el controlador desconocido y el puerto por el que se recibió el C-Adv. Esta información permitirá al switch reconectar rápidamente al nuevo controlador si la conexión con el controlador actual falla. Este flujo también es efímero ya que si los mensajes C-Adv del controlador desconocido se dejan de recibir (porque se haya producido un fallo en este controlador) el switch debe borrar esta información.

En el cuadro inferior se puede observar como se transcriben estas condiciones a

[3]Este registro se utiliza por si el switch se desconecta con un determinado controlador para tener inmediatamente un puerto de salida a otro controlador activo

código Python.

```
flow_actions+=[parser.NXActionLearn(table_id=self.
    ACTIVE_PORTS_TABLE,
              specs=[
                  parser.NXFlowSpecLoad(src=("in_port", 0),
                                        dst=("reg9", 0),
                                        n_bits=32)
              ],
              hard_timeout=self.REG9_TIMEOUT,
              priority=24999,
              cookie=self.COOKIE)
          ]
```

Una vez establecida esta nueva regla, el switch también debe generar un Packet_In hacia su controlador que contenga el C-Adv recibido, permitiendo que este actualice la topología de la red y tome conocimiento de la presencia de este nuevo controlador.

En la figura 3.13 se muestra un flujograma del funcionamiento de la recepción de mensajes C-Adv provenientes de controladores desconocidos.

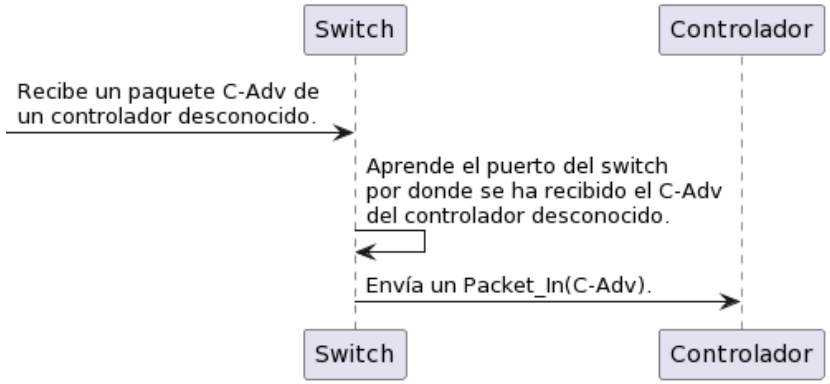

Figura 3.13: Flujograma sobre la gestión de controladores desconocidos.
Fuente: elaboración propia.

3.2.3. Función _packet_in_handler

Esta función se encarga de realizar ciertas acciones en el controlador en función del tipo de paquete que se reciba dentro de un paquete Packet_In.

Las modificaciones incluidas en este código se encargan de procesar los paquetes Packet_In que contienen C-Adv que se utilizan para refrescar los enlaces ya existentes

en el escenario o incluir nuevas rutas en caso de que se reciba información sobre nuevos enlaces.

El primer paso que se realiza es verificar si el paquete es de tipo IPv4 (pkt_ipv4), si la dirección IP de origen del paquete IPv4 es igual a la dirección IP del controlador (self.config.controller_ip()), y si el paquete UDP (pkt_udp) tiene el puerto de origen correcto (self.CONTROLLER_SRC_PORT_PROBE). Esto indica que se trata de un mensaje C-Adv proveniente del propio controlador.

Si el paquete es un C-Adv, el controlador extrae la siguiente información procesando el paquete Packet_In que contiene el C-Adv:

- La dirección IP del controlador que envió el paquete C-Adv.
- El número de puerto por el que el switch origen envió el C-Adv.
- La dirección IP del dispositivo switch de origen.
- El número de puerto en el que el switch receptor recibió el C-Adv.
- La dirección IP del switch receptor.

Una vez extraídos todos los datos del paquete C-Adv, el controlador procede a actualizar el grafo que representa la topología de la red. Si la conexión entre los dos switches origen y destino incluida en el C-Adv no se conoce, se aprenderá este nuevo enlace. Este descubrimiento permitirá calcular nuevas rutas entre los switches y el controlador.

Para poder gestionar los paquetes C-Adv provenientes de otros controladores se emplea el mismo código que en la versión anterior.

Capítulo 4

Plan de Pruebas y Resultados

Una vez se ha desarrollado todo el software, es necesario realizar una serie de pruebas para comprobar el éxito de esta nueva implementación. En este capítulo se describen cuales son las pruebas que se han realizado y cual es su resultado.

Para el desarrollo de estas pruebas, se emplean los 2 escenarios analizados en el capítulo 3:

- **Escenario básico**: donde se comprueba que los switches aprenden los caminos alternativos a través de la nueva implementación, y se demuestra que el número de paquetes que emplea la red es menor.
- **Escenario con dos controladores**: en el que se analiza como se detectan controladores vecinos a través de los switches frontera.

También es necesario realizar dos verificaciones: primero analizar los flujos instalados en las tablas de los switches, y segundo comprobar las cabeceras NSH. En el primer escenario se consultarán las tablas de los switches s2, s3, s4, s5 y s7. En el segundo escenario se realizan consultas en las tablas de c0 y c1 para monitorizar todo el flujo entre los dos controladores.

Para poder realizar la consulta de los flujos en los switches se emplea el siguiente comando en el terminal del dispositivo elegido[1]:

```
./dumpAllFlows.sh s3
```

Dicho comando ejecuta el script dumpAllFlows.sh, acompañado del argumento nombre del switch que se quiere consultar. En caso de que solamente se quiera visualizar los paquetes con destino al controlador, se puede filtrar la salida para que sólo

[1]En este caso se ha usado de ejemplo el switch s3, pero se puede cambiar el valor numérico para estudiar cualquier otro elemento.

muestre los flujos que contengan la dirección IP del controlador[2]:

```
./dumpAllFlows.sh s3 | grep 10.0.0.1
```

Para comprobar la cantidad de paquetes que se emplean al ejecutar el nuevo algoritmo se utiliza wireshark, que se abre ejecutando el comando que se muestra en el cuadro inferior en el terminal del switch:

```
sudo wireshark &
```

4.1. Pruebas escenario básico

En este primer escenario, el objetivo principal es demostrar la existencia de diferentes rutas hacia el controlador. Para poder localizar entre todos los datos mostrados cuales son los caminos hacia el controlador, es necesario localizar el parámetro ip.nw_dst=10.0.0.1 que indica que el flujo se utiliza para encaminar paquetes dirigidos a la dirección IP del controlador. En la figura 4.1 se muestra este flujo instalado en el switch s2, el recuadro de color rojo corresponde al camino principal y el recuadro azul corresponde al camino alternativo.

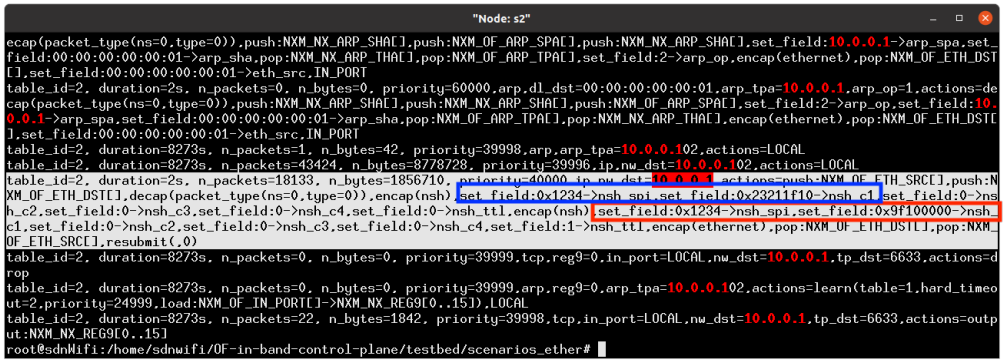

Figura 4.1: Flujo instalado en el switch s2 para encaminar paquetes dirigidos al controlador.

Fuente: elaboración propia.

En la figura 4.2 se puede observar en el escenario los saltos que forman parte del el camino principal y del alternativo desde s2 al controlador c0.

[2]La dirección IP puede cambiar en función del controlador que se quiere consultar.

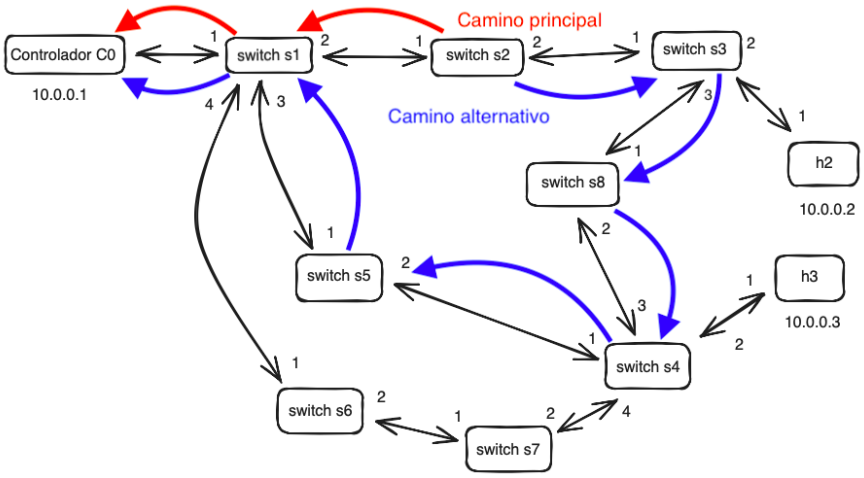

Figura 4.2: Camino principal y alternativo desde s2 al controlador.
Fuente: elaboración propia.

Al observar el flujo instalado en el switch de s2 correspondiente a la figura 4.1, se puede apreciar que el switch s2 tiene un camino principal para alcanzar el controlador y un camino alternativo. La ruta marcada en rojo presenta la ruta principal 0x9f100000, es decir $s2 \rightarrow s1 \rightarrow c0$. Cabe recordar que el valor del primer salto es 9 en lugar de 1 debido al bit más relevante a 1 para indicar que el puerto 1 del switch s2 tiene un camino alternativo. La ruta marcada en azul corresponde al camino alternativo, que es 0x23211f10, es decir $s2 \rightarrow s3 \rightarrow s8 \rightarrow s4 \rightarrow s5 \rightarrow s1 \rightarrow c0$. A partir de esta información, se sabe que si el puerto 1 de s2 está inactivo, s2 encaminará los paquetes al controlador a través del camino alternativo.

Si se analiza la figura 4.3 correspondiente a s3, se puede apreciar que s3 presenta 3 caminos diferentes para alcanzar el controlador. La ruta marcada en rojo presenta la ruta 0x99f100000, es decir $s3 \rightarrow s2 \rightarrow s1 \rightarrow s0$. Al igual que en s2, al ser la ruta principal, se modifica el valor del bit mas relevante, en el caso en el que alguno de esos puertos del camino principal tengan camino alternativo. En este caso hay dos caminos alternativos:

a) Desde s3 el puerto de salida 1 tiene alternativo y por eso está codificado como 9.

b) Desde s2 el puerto de salida 1 tiene alternativo y por eso está codificado como 9.

La primera ruta marcada en azul que se encuentra en la posición más cercana al camino principal, corresponde al camino alternativo 0x3211f100, es decir $s3 \rightarrow s8 \rightarrow s4 \rightarrow s5 \rightarrow s1 \rightarrow s0$. La segunda ruta marcada en azul corresponde al camino alternativo, que es 0x23211f100, es decir es $s2 \rightarrow s3 \rightarrow s8 \rightarrow s4 \rightarrow s5 \rightarrow s1 \rightarrow s0$.

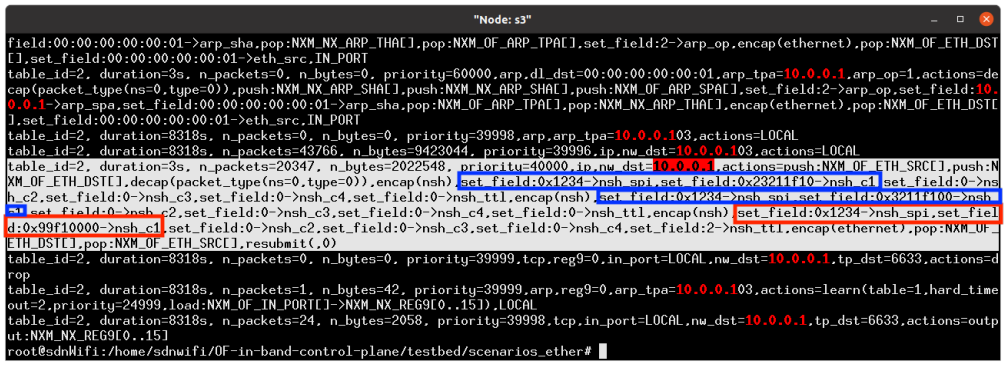

Figura 4.3: Flujo instalado en el switch s3 para encaminar paquetes dirigidos al controlador.

Fuente: elaboración propia.

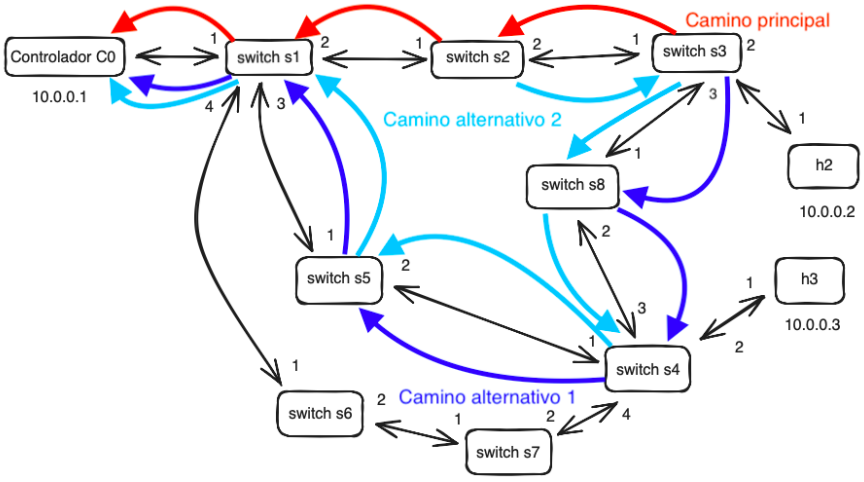

Figura 4.4: Camino principal y alternativos desde s3 al controlador.
Fuente: elaboración propia.

En la figura 4.4 se puede observar en el escenario los saltos que forman parte del camino principal y los alternativos desde s3 y s2 al controlador c0.

La segunda ruta alternativa parte desde el switch s2 porque los caminos alternativos se inician desde el salto en el que hay un alternativo. Cada vez que el paquete sale de un nodo se borra de la cabecera NSH el puerto de salida de ese nodo. Por tanto, los caminos alternativos sólo tienen sentido en el nodo que estamos considerando cuando se está encaminando, no se consideran desde el switch origen de ese paquete. Este fenómeno también sucede en los demás switches que también presentan más de un camino alternativo en alguno de los saltos intermedios del camino principal.

En el switch s4 correspondiente a la figura 4.5 se pueden apreciar 3 caminos para encaminar paquetes hacia el controlador, la ruta principal 0x99f10000 marcada en rojo, que transcurre por $s4 \rightarrow s5 \rightarrow s1 \rightarrow s0$, la primera ruta secundaria 0x411f1000 marcada en azul, es decir $s4 \rightarrow s7 \rightarrow s6 \rightarrow s1 \rightarrow s0$ y la segunda ruta secundaria 0x2411f1000 marcada en azul, es decir $s5 \rightarrow s4 \rightarrow s7 \rightarrow s6 \rightarrow s1 \rightarrow s0$.

Figura 4.5: Flujo instalado en el switch s4 para encaminar paquetes dirigidos al controlador.

Fuente: elaboración propia.

En la figura 4.6 se puede observar en el escenario los saltos que forman parte del camino principal y los alternativos desde s4 al controlador c0.

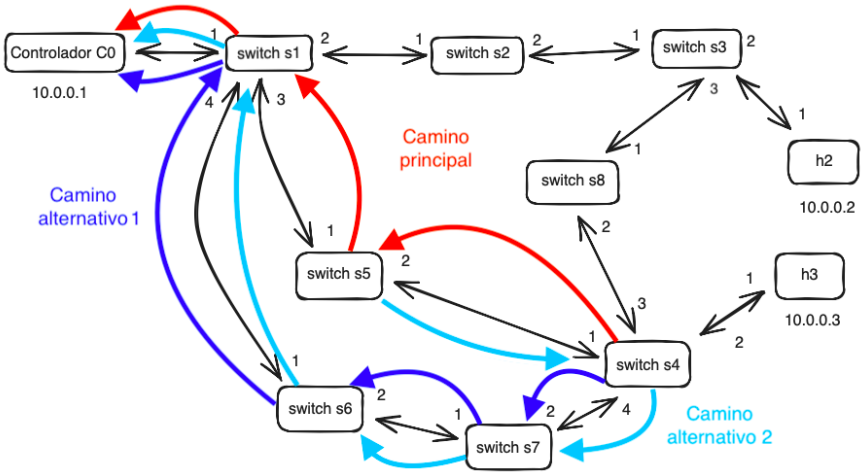

Figura 4.6: Camino principal y alternativos desde s4 al controlador.
Fuente: elaboración propia.

En el switch s5 correspondiente a la figura 4.7, al igual que sucede con los otros equipos, se muestran 2 caminos alternos, el camino principal 0x9f100000 marcado en rojo, que conecta directamente $s5 \rightarrow s1 \rightarrow s0$ y el secundario 0x2411f100 marcado en azul, que traducido recorre la ruta $s5 \rightarrow s4 \rightarrow s7 \rightarrow s6 \rightarrow s1 \rightarrow s0$.

En la figura 4.8 se puede observar en el escenario los saltos forman parte del camino principal y del alternativo desde s5 al controlador c0.

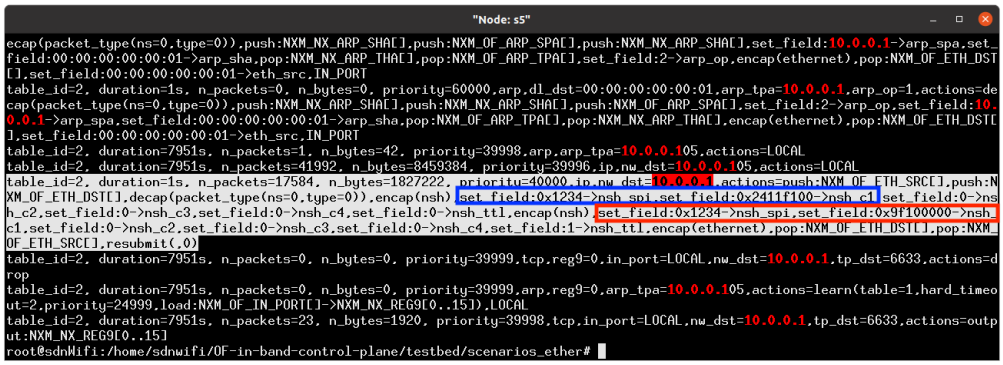

Figura 4.7: Flujo instalado en el switch s5 para encaminar paquetes dirigidos al controlador.

Fuente: elaboración propia.

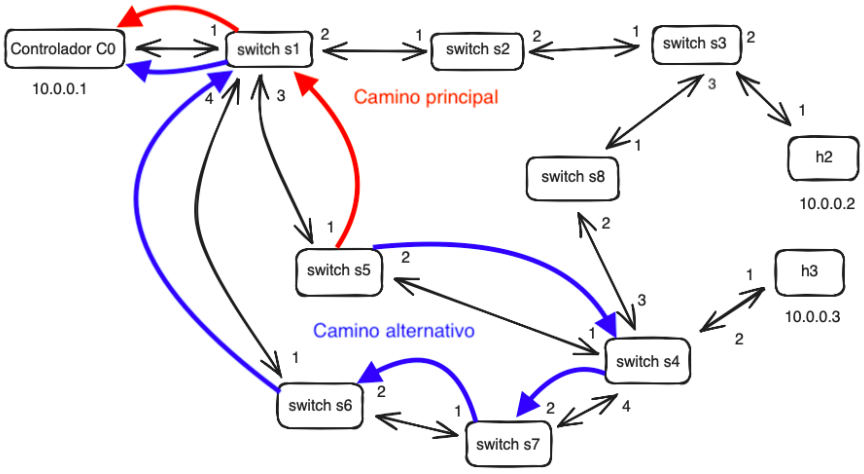

Figura 4.8: Camino principal y alternativo desde s5 al controlador.
Fuente: elaboración propia.

Por último, se analiza el flujo para encaminar paquetes hacia el controlador presente en el switch s7 correspondiente a la figura 4.9, que está compuesto por su ruta principal 0x99f10000 marcada en rojo, que transcurre por los switches $s7 \rightarrow s6 \rightarrow s1 \rightarrow s0$ y por dos rutas secundarias. Su primera ruta secundaria es 0x211f1000 que se encuentra marcada en azul, es decir $s7 \rightarrow s4 \rightarrow s5 \rightarrow s1 \rightarrow s0$ y su segunda ruta alternativa es 0x2211f1000 que se encuentra marcada en azul, es decir $s6 \rightarrow s7 \rightarrow s4 \rightarrow s5 \rightarrow s1 \rightarrow s0$.

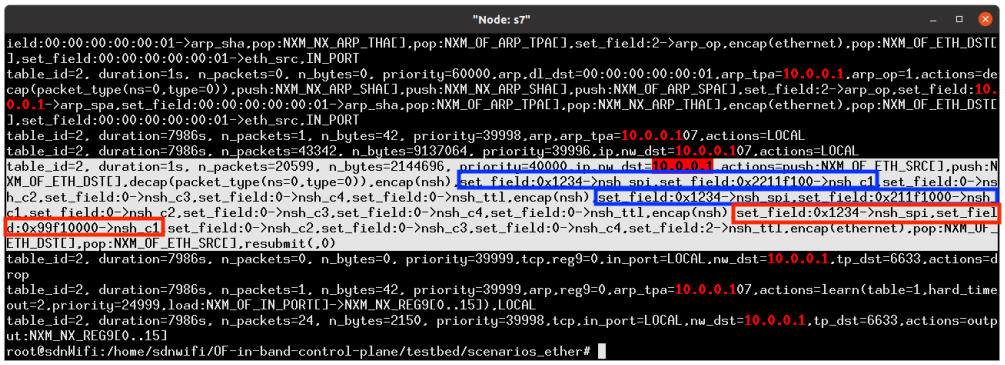

Figura 4.9: Flujo instalado en el switch s7 para encaminar paquetes dirigidos al controlador.

Fuente: elaboración propia.

En la figura 4.10 se puede observar en el escenario los saltos que forman parte del camino principal y los alternativos desde s7 al controlador c0.

Se puede apreciar que todos los switches del escenario muestran una ruta principal al controlador y una ruta alternativa que se podrá utilizar en caso de que la primera ruta falle, permitiendo así el funcionamiento ininterrumpido del escenario a pesar del fallo de alguna de las conexiones.

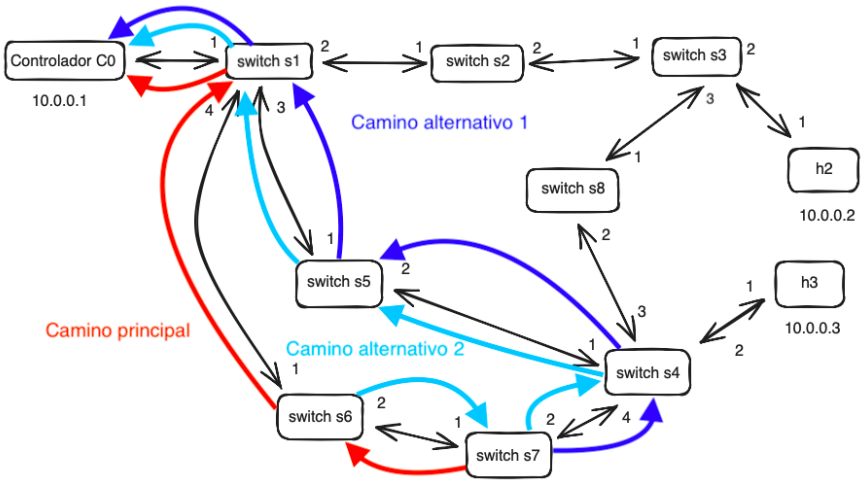

Figura 4.10: Camino principal y alternativo desde s7 al controlador.
Fuente: elaboración propia.

Uno de los objetivos de este nuevo algoritmo es reducir la cantidad de paquetes que se necesitan para poder actualizar la topología del escenario con el descubrimiento de nuevos enlaces. Para comprobar que esta reducción se ha conseguido, es necesario realizar capturas de Wireshark en dos escenarios diferentes: antes de la implementación y después. Para esta prueba, seleccionamos el enlace entre los el controlador c0 y el switch s1.

Como se puede apreciar en la figura 4.11, al realizar una captura de tráfico en el escenario antes de implementar el algoritmo, se pueden cuantificar una gran cantidad de paquetes (37 paquetes) que emplea el protocolo LLDP en un espacio muy corto de tiempo (menos de 2 segundos). El controlador genera por cada uno de los puertos de un switch un mensaje Packet_Out(LLDP) dirigido a dicho switch, para que éste copie el mensaje LLDP en el puerto indicado por el controlador. Por tanto si tuviéramos n switches con m puertos cada uno de ellos, el controlador generará nxm Packet_Out(LLDP). Además el LLDP recibido por el switch vecino, provocará el envío de un Packet_IN(LLDP) al controlador. Con la implementación de este nuevo algoritmo se busca sustituir todos los paquetes Packet_Out(LLDP) por un único mensaje C-Adv que inunde de forma controlada la topología y provoque los Packet_In(C-Adv) que permitan realizar el descubrimiento de la topología.

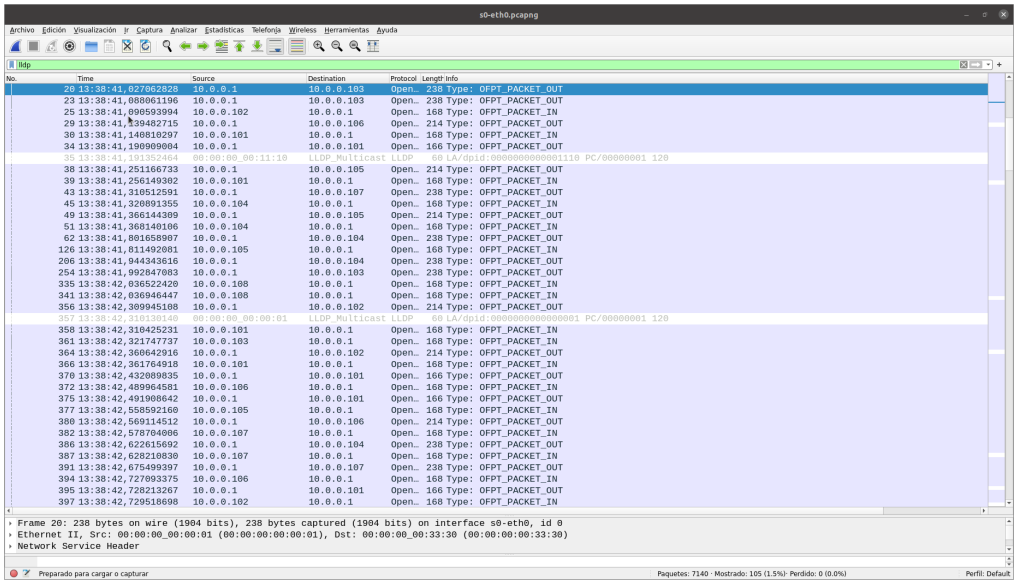

Figura 4.11: Captura del tráfico entre c0 y s1 con el protocolo LLDP.
Fuente: elaboración propia.

Si se compara la figura 4.12 con la figura analizada en el anterior párrafo, se puede observar que el número de paquetes capturados es bastante menor (13 paquetes) en un período similar al anterior. Un sólo mensaje C-Adv se propaga por toda la topología generando Packet-In(C-Adv) por cada enlace que hay entre 2 switches. Por tanto, desaparecen todos los mensajes Packet_Out. Esto implica que, con la nueva implementación, el número de paquetes que circula por el escenario es inferior y la posibilidad de que la red se congestione es mucho menor.

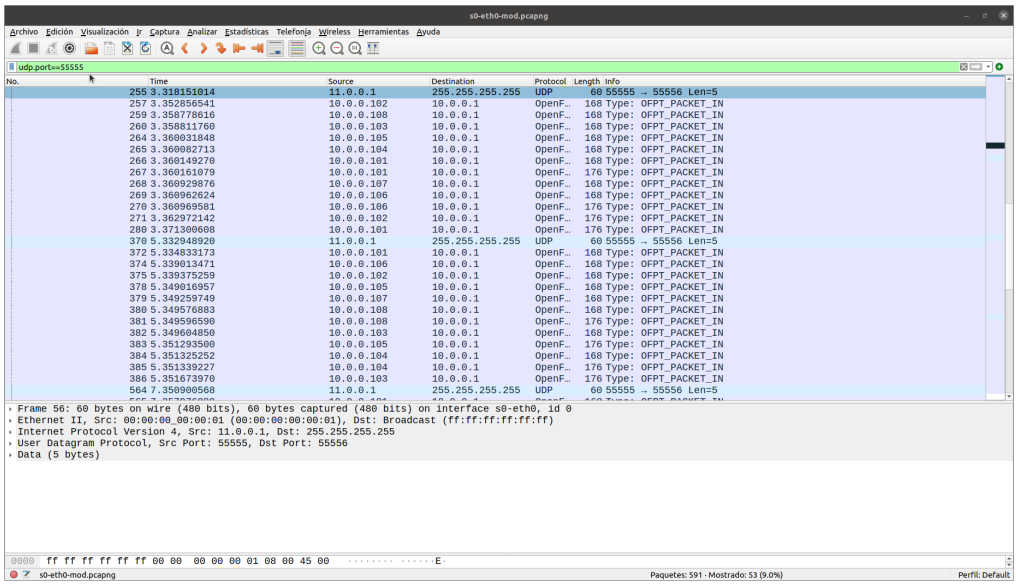

Figura 4.12: Captura del tráfico entre c0 y s1 con el nuevo algoritmo.
Fuente: elaboración propia.

4.2. Pruebas escenario con 2 controladores

En este segundo escenario, el objetivo principal es demostrar la detección de otros controladores vecinos y que el escenario permiten encaminar mensajes entre ellos. En este caso, es necesario fijarse en las direcciones destino 11.0.0.1 y 11.0.0.2 que identifican a los controladores c0 y c1.

Para ver cómo los controladores pueden intercambiar tráfico entre ellos, se van a consultar las tablas de cada uno de los switches raíz que residen en la misma máquina virtual que el controlador, de esta forma es posible apreciar las rutas que tienen instaladas para alcanzar al controlador vecino.

Si se observa la figura 4.13, se puede apreciar que el controlador c0 tiene una ruta para comunicarse con el controlador c1 (11.0.0.1), que está definida por el flujo 0x1fb que se encuentra marcado en rojo y una alternativa que es 0x4f2 que se muestra marcada en azul.

En la figura 4.14 se puede observar en el escenario los saltos que forman parte del camino principal y del alternativo desde c0 hacia c1.

El switch del controlador c0 sólo tiene configurado la parte del camino que corresponde a la zona del controlador c0, por tanto, sólo será capaz de llegar a los switches frontera con la zona del controlador c1. En este caso c0 sólo incluye el camino hasta

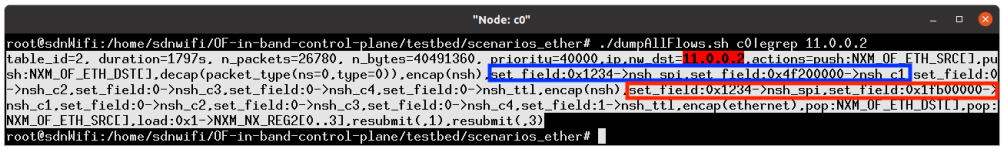

Figura 4.13: Flujo instalado en el switch c0 para encaminar paquetes dirigidos al controlador c1.

Fuente: elaboración propia.

que el paquete sale de s1 (camino principal) o hasta que el paquete sale de s6 (camino alternativo) ya que s5 y s7 pertenecen a la zona gestionada por c1.

Una vez llegan los mensajes a los switches pertenecientes a la zona gestionada por c1, estos incluyen nuevas cabeceras NSH que permiten completar los saltos desde los switches frontera hasta el controlador c1.

En las figuras 4.15 y 4.16, se puede observar como s5 y s7 respectivamente añaden estas nuevas cabeceras para poder llegar al controlador c1, que están definidas por los flujos 0x24f3, que se encuentra marcado en rojo, y 0xf3, que se muestra marcado en azul.

En la figura 4.17 se puede observar en el escenario los saltos que forman parte del camino principal y del alternativo completo desde c0 hacia c1.

Análogamente, el encaminamiento desde c1 hacia c0, se realiza de forma similar. El switch raíz de c1 incluye primeramente el camino desde c1 hasta los switches frontera de la zona gestionada por c1. A continuación, los switches frontera de la zona de c0 añadirán el camino para alcanzar c0, permitiendo establecer la ruta desde c1 a c0.

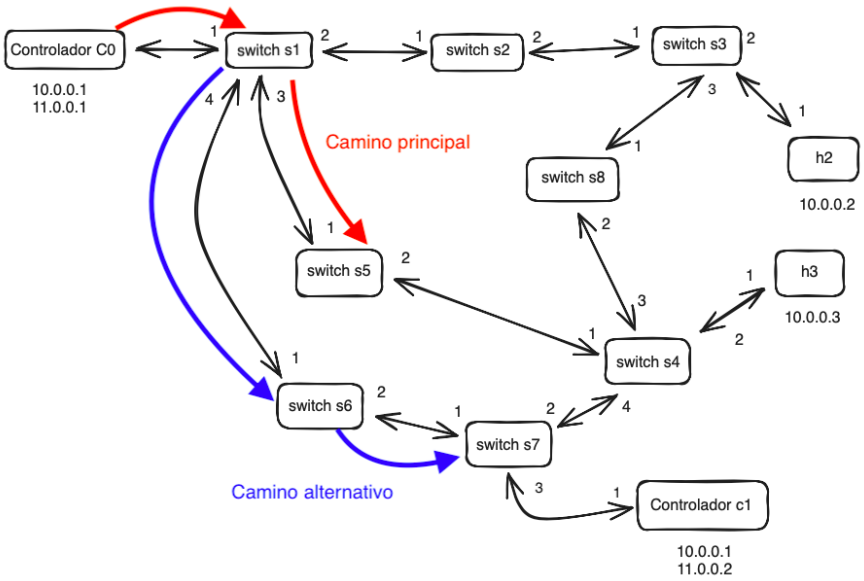

Figura 4.14: Camino principal y alternativo de desde c0 hacia c1 en la zona controlada por c0.

Fuente: elaboración propia.

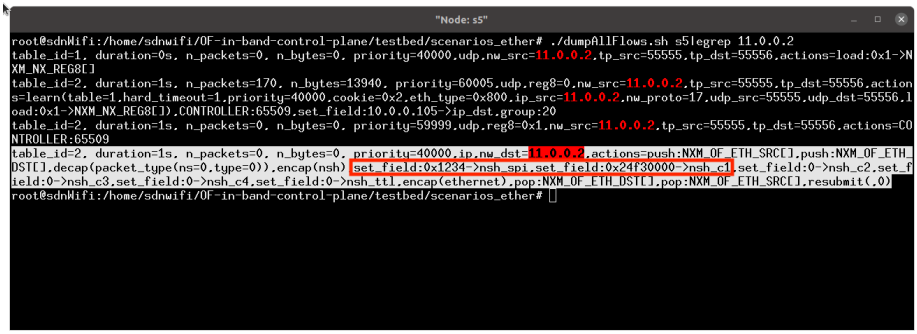

Figura 4.15: Flujo instalado en el switch s5 para encaminar paquetes dirigidos al controlador c1.

Fuente: elaboración propia.

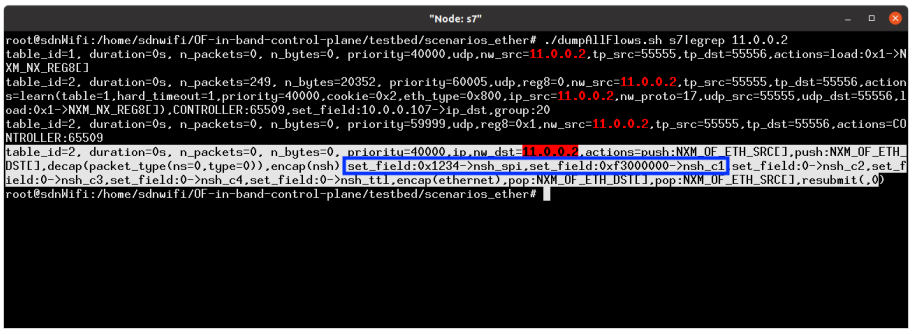

Figura 4.16: Flujo instalado en el switch s7 para encaminar paquetes dirigidos al controlador c1.

Fuente: elaboración propia.

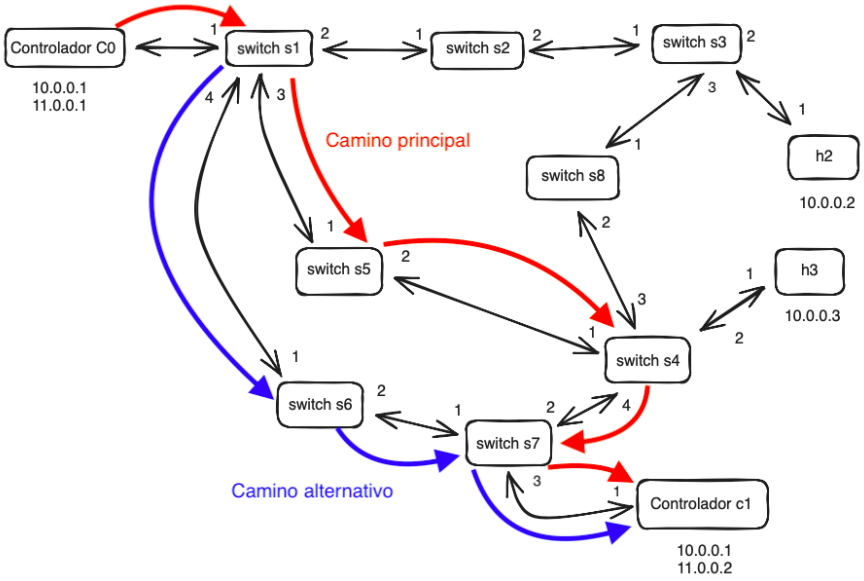

Figura 4.17: Camino principal y alternativo completo de desde c0 hacia c1.
Fuente: elaboración propia.

Capítulo 5

Conclusiones

Como ya se ha comentado en el capítulo 2 de este trabajo, el objetivo principal de este proyecto era desarrollar una alternativa al aprendizaje de rutas alternativas en un entorno virtualizado, buscando así desarrollar escenarios con una carga menor de paquetes, un aprendizaje más óptimo de las rutas y un envío de paquetes más simple, a la vez que se mantenía la comunicación entre los switches y entre subredes vecinas con distintos controladores.

Si se observan las capturas y las explicaciones desarrolladas en el capítulo 5 de este trabajo, se puede deducir que el objetivo principal se ha cumplido. La nueva implementación genera un único paquete que se distribuye por inundación a lo largo de todo el escenario y permite a los switches aprender tanto la ruta principal hacia el controlador de la subred, como posibles caminos alternativos.

Se ha sustituido el protocolo LLDP con una implementación más eficaz. Como se demuestra en el capítulo 4, la sustitución de los paquetes generados por este protocolo por uno solo originado en el controlador se ha completado de manera exitosa, consiguiendo así eliminar el protocolo a la vez que se reduce el número de paquetes en la red.

Los switches también son capaces de detectar si reciben un C-Adv por primera vez, lo que les permite descartar aquellos que ya han recibido previamente, evitando así la inundación de mensajes a través de bucles que pueden saturar la red.

Esta nueva implementación permite además mantener la capacidad de comunicar dos o más subredes, a través de la comunicación entre sus controladores.

En conclusión, se puede afirmar que tanto el objetivo principal, como los objetivos secundarios marcados al principio de este trabajo, se han cumplido satisfactoriamente.

5.1. Lecciones Aprendidas

Cada profesor y materia del grado ha desempeñado un papel crucial en el desarrollo de este trabajo. Sin embargo, algunas de estas asignaturas han ganado una importancia destacada debido a la naturaleza específica de su contenido:

- **Arquitectura de Redes de Ordenadores**: Proporciona una introducción a las arquitecturas de protocolos en redes de ordenadores, centrándose en los protocolos TCP/IP. Estos conocimientos son fundamentales a la hora de abordar este trabajo.
- **Sistemas Telemáticos**: Desarrolla y aporta los conocimientos básicos sobre el estudio de los dispositivos de interconexión de redes, los protocolos de encaminamiento. Estos conocimientos son la base de este proyecto.
- **Servicios y Aplicaciones Telemáticas**: Proporciona los conocimientos de Python necesarios para abordar las modificaciones realizadas durante la realización del trabajo.
- **Expresión Oral y Escrita y Búsqueda de Información**: Ayuda a desarrollar habilidades tan importantes como redactar, buscar información y desarrollar la memoria.

5.2. Trabajos Futuros

Se pueden incluir mejoras que permiten la continua evolución y desarrollo del controlador en función de sus necesidades futuras. Algunas de estas propuestas pueden ser:

- **Optimización de Algoritmos de Enrutamiento**: mejorar los algoritmos de enrutamiento con técnicas de inteligencia artificial para adaptar dinámicamente las rutas en función de las condiciones de la red.
- **Seguridad en Redes Virtualizadas**: Investigar y desarrollar mecanismos de seguridad adicionales para proteger las redes virtualizadas, como por ejemplo la implementación de firewalls, sistemas de detección de intrusiones o cifrado de tráfico en entornos virtualizados.
- **Escalabilidad del Sistema**: Analiza la escalabilidad del sistema en términos de manejo de un mayor número de nodos y conexiones sin perder la eficiencia que puede mostrar una red más pequeña.

Capítulo 6

Anexo

En este apartado, se muestra a nivel práctico como se crean los escenarios en entornos virtuales y cual es el software necesario para definir, y gestionar los dos escenarios principales.

Configuración de los elementos comunes

Como se menciona en el apartado de Escenario Mininet, el primer elemento que se define dentro de los entornos virtuales son los switches. En este primer script se puede observar la definición de todos los switches del escenario bucle4_bdf [1]. Cada switch se define por un nombre, una dirección IP que inicialmente no está configurada y una dirección Ethernet.

```
s0 = net.addHost('s0', ip='0.0.0.0.', mac='00:00:00:00:00:01')
s1 = net.addHost('s1', ip='0.0.0.0',  mac='00:00:00:00:11:10')
s2 = net.addHost('s2', ip='0.0.0.0',  mac='00:00:00:00:22:20')
s3 = net.addHost('s3', ip='0.0.0.0',  mac='00:00:00:00:33:30')
s4 = net.addHost('s4', ip='0.0.0.0',  mac='00:00:00:00:44:40')
s5 = net.addHost('s5', ip='0.0.0.0',  mac='00:00:00:00:55:50')
s6 = net.addHost('s6', ip='0.0.0.0',  mac='00:00:00:00:66:60')
s7 = net.addHost('s7', ip='0.0.0.0',  mac='00:00:00:00:77:70')
s8 = net.addHost('s8', ip='0.0.0.0',  mac='00:00:00:00:88:80')
```

Una vez definidos los switches, es necesario establecer los enlaces. Es importante recordar que los puertos comienzan su enumeración en 0 para definir las interfaces Ethernet, pero que dichos puertos comienzan su numeración en 1 para el protocolo OpenFlow.

```
net.addLink( s0, s1, 0, 0 )
```

[1] El software adicional de la topología con 2 controladores se detalla un poco más abajo.

```
net.addLink( s1, s2, 1, 0 )
net.addLink( s1, s5, 2, 0 )
net.addLink( s1, s6, 3, 0 )
net.addLink( s6, s7, 1, 0 )
net.addLink( s5, s4, 1, 0 )
net.addLink( s2, s3, 1, 0 )
net.addLink( s3, h2, 1, 0 )
net.addLink( s4, h3, 1, 0 )
net.addLink( s3, s8, 2, 0 )
net.addLink( s8, s4, 1, 2 )
net.addLink( s4, s7, 3, 1 )
```

A continuación se modifican las direcciones Ethernet de las interfaces creadas al generar los enlaces anteriores:

```
s1.cmd("ip link set s1-eth1 address 00:00:00:00:11:11")
s1.cmd("ip link set s1-eth2 address 00:00:00:00:11:12")
s1.cmd("ip link set s1-eth2 address 00:00:00:00:11:13")
s2.cmd("ip link set s2-eth1 address 00:00:00:00:22:21")
s3.cmd("ip link set s3-eth1 address 00:00:00:00:33:31")
s3.cmd("ip link set s3-eth2 address 00:00:00:00:33:32")
s4.cmd("ip link set s4-eth1 address 00:00:00:00:44:41")
s4.cmd("ip link set s4-eth2 address 00:00:00:00:44:42")
s4.cmd("ip link set s4-eth3 address 00:00:00:00:44:43")
s5.cmd("ip link set s5-eth1 address 00:00:00:00:55:51")
s6.cmd("ip link set s6-eth1 address 00:00:00:00:66:61")
s7.cmd("ip link set s7-eth1 address 00:00:00:00:77:71")
s8.cmd("ip link set s8-eth1 address 00:00:00:00:88:81")
```

Una vez completada la configuración del escenario, es necesario ejecutar dos scripts para arrancarlo: ‘start-controler-switch-n-ifaces.sh’ y ‘start-switch.sh’.

‘Start-controler-switch-n-ifaces.sh’ aparece definido en el siguiente script.

```
#!/bin/bash
if [ ! $# -eq 4 ]
then
    echo "Usage error: $0 <switchName> <N_IFACES> <
        CONTROLLER_IP> <CONTROLLER_EXT_IP>"
    exit -1
fi

SWITCH_NAME=$1
N_IFACES=$2
CONTROLLER_IP=$3
CONTROLLER_EXT_IP=$4

```

```
./scripts/startOvsDb.sh $SWITCH_NAME
./scripts/startOvs.sh $SWITCH_NAME
./scripts/createBrSdn.sh $SWITCH_NAME $N_IFACES $CONTROLLER_IP
ifconfig $SWITCH_NAME inet $CONTROLLER_IP
ifconfig $SWITCH_NAME:1 inet $CONTROLLER_EXT_IP

nsenter -t $PID_NAMESPACE -n ./scripts/reglas-controller-switch
    -n-ifaces.sh $SWITCH_NAME $SWITCH_IF_NUMBER
    $SWITCH_IP_ADDRESS $CONTROLLER_EXT_IP
nsenter -t $PID_NAMESPACE -n ip route change 10.0.0.0/8 dev
    $SWITCH_NAME advmss 1000


# BFD

if [ $BFD = "True" ]
then
   val=`expr $SWITCH_IF_NUMBER - 1`
   for i in $(seq 0 $val);
   do
      nsenter -t $PID_NAMESPACE -n ./scripts/startBFD-sw-iface.
          sh $SWITCH_NAME eth$i 100
   done
fi

echo "-----------------------------------------------------"
echo "Hora"
date +"%H:%M:%S,%N"
```

Al ejecutar este programa se producen las siguientes acciones:

- Emplea el script ‘startOvsDb.sh’ para crear la base de datos para Open vSwitch dentro de un espacio de nombres de red.
- Genera el switch Open vSwitch mediante ‘startOvs.sh’.
- Genera las interfaces requeridas para el Open vSwitch mencionado empleando ‘createBrSdn.sh’.
- Configura dos direcciones IP en el switch: 10.0.0.1 (dirección anycast) y 11.0.0.X (dirección IP específica).
- Se activa el protocolo Bidirectional Forwarding Detection (BFD) en las interfaces con un valor predeterminado de período de 100 ms para supervisar la conectividad de la red.

Estas acciones permiten configurar de una manera más precisa el funcionamiento de open vSwitch, además de habilitar la supervisión de la conectividad mediante BFD.

También se define el script 'start-switch.sh', que se encarga de configurar el resto de switches, siendo necesario introducir el nombre del switch, su número de interfaces y sus dirección IP.

```
#!/bin/bash


if [[ $# -ne 4 && $# -ne 5 ]]
then
    echo "Usage error: $0 <switchName> <if_number> <ip_address>
        <ext_ip_address> [bfd]"
    exit -1
fi


BFD="True"

if [ $# -eq 4 ]
then
    echo "No BFD..."
    BFD="False"
fi

SWITCH_NAME=$1       # cX
SWITCH_IF_NUMBER=$2  # 1
SWITCH_IP_ADDRESS=$3 # 10.0.0.1
CONTROLLER_EXT_IP=$4 # 11.0.0.X

echo "Starting $SWITCH_NAME $SWITCH_IP_ADDRESS"

PID_NAMESPACE=`ps -feaww | grep "mininet:$SWITCH_NAME" | head
    -1 | awk '{print $2}'`
echo "PID_NAMESPACE=$PID_NAMESPACE"

nsenter -t $PID_NAMESPACE -n ./scripts/startOvsDb.sh
    $SWITCH_NAME
nsenter -t $PID_NAMESPACE -n ./scripts/startOvs.sh $SWITCH_NAME
nsenter -t $PID_NAMESPACE -n ./scripts/createBrSdn.sh
    $SWITCH_NAME $SWITCH_IF_NUMBER $SWITCH_IP_ADDRESS
nsenter -t $PID_NAMESPACE -n ifconfig $SWITCH_NAME inet
    $SWITCH_IP_ADDRESS
nsenter -t $PID_NAMESPACE -n ifconfig $SWITCH_NAME:1 inet
    $CONTROLLER_EXT_IP

nsenter -t $PID_NAMESPACE -n ./scripts/reglas-controller-switch
    -n-ifaces.sh $SWITCH_NAME $SWITCH_IF_NUMBER
    $SWITCH_IP_ADDRESS $CONTROLLER_EXT_IP
nsenter -t $PID_NAMESPACE -n ip route change 10.0.0.0/8 dev
    $SWITCH_NAME advmss 1000
```

```
37
38
39  # BFD
40
41  if [ $BFD = "True" ]
42  then
43     val=`expr $SWITCH_IF_NUMBER - 1`
44     for i in $(seq 0 $val);
45     do
46        echo "Activating BFD interface eth$i"
47        nsenter -t $PID_NAMESPACE -n ./scripts/startBFD-sw-iface.
              sh $SWITCH_NAME eth$i 100
48     done
49  fi
50
51  echo "------------------------------------------------------"
52  echo "Hora de creaci n del switch $SWITCH_NAME"
53  date +"%H:%M:%S,%N"
```

Todos estos pasos permiten, tanto al switch controlador como a los restantes, establecer una configuración mínima inicial para conectarse y habilitar el enlace de todos los equipos presentes en el escenario. También permiten establecer el tamaño anunciado de segmento (MSS) en 1000, lo cual garantiza la inclusión de las cabeceras NSH sin exceder la Unidad Máxima de Transferencia (MTU) de Ethernet (lo que impediría el envío del mensaje), ya que las cabeceras NSH se agregan después de que Protocolo de control de transmisión (TCP) haya calculado el valor de MSS.

Encendido y apagado del escenario Periplus

Para poder arrancar un escenario con un controlador Periplus, es necesario ejecutar los siguientes comandos en una terminal shell desde la carpeta "$HOME/OF-in-band-control-plane/testbed/scenarios_ether/":

```
1  cd $HOME/OF-in-band-control-plane/testbed/scenarios_ether/
2  sudo python3 ./scenarios/bucle4_bfd/scenario_bucle4_bfd.py
```

Al ejecutar estas órdenes activaremos el entorno Mininet, dentro del cual se accede a la terminal del controlador S0 para poder activar el entorno sdnenv (si se observa dicha terminal, se puede apreciar que el prompt ha cambiado, mostrando un sdnenv delante):

```
1  s0~:> source /home/sdnwifi/sdnenv/bin/activate
```

Una vez activado, se puede lanzar el controlador ejecutando el siguiente comando:

```
source /home/sdnwifi/sdnenv/bin/activate ryu-manager --verbose
    InBandController.py --config-file ./scenarios/bucle4_bfd/
    params-controller-s0.conf > /tmp/salida 2>&1
```

Para poder apreciar de una forma más clara los posibles errores y la información que está gestionando la máquina, se redirige la salida al fichero "/tmp/salida/".

De este comando hay que destacar la opción "*--observe-links*", que permite a los switches aprender los distintos caminos alternativos posibles, a través del protocolo LLDP. El algoritmo desarrollado en este trabajo muestra una alternativa en la que dichos caminos puedan descubrirse sin utilizar LLDP y en consecuencia, sin la opción "*--observe-links*".

Para poder comprobar que los switches se conectan de forma correcta, antes de continuar con el arranque del escenario, se debe ejecutar, en una terminal independiente, un comando que es el encargado de filtrar en el fichero de salida la cadena MANAGED. El resultado de este comando irá mostrando una línea por cada switch que haya configurado el controlador y por tanto haya pasado al estado gestionado.

```
sudo tail -f /tmp/salida | grep MANAGED
```

Es recomendable mantener visible esta ventana para poder observar los switches que han pasado al estado gestionado.

Desde un terminal Linux independiente a los dos anteriores, arrancan los switches:

```
cd $HOME/OF-in-band-control-plane/testbed/scenarios_ether/
sudo ./scenarios/bucle4_bfd/start_scenario_bucle4_bfd.sh
```

Si se observan los dos terminales definidos previamente, se evidencia que de forma paralela van apareciendo con el mensaje MANAGED los switches en estado activo, lo cual les permite conocer que los switches ya tienen instalados los flujos que permiten el encaminamiento basado en las cabeceras NSH.

Una de las formas que tenemos de probar estos caminos alternativos es desactivando uno de los switches. Esta desconexión la podemos hacer con el siguiente comando, que en este caso anula el switch s5 del escenario:

```
cd $HOME/OF-in-band-control-plane/testbed/scenarios_ether/
sudo ./scripts/kill-switch.sh s5 2 10.0.0.105
```

Una vez desactivado s5, el switch s4 detecta que ya no puede utilizar ese camino y comienza a enviar los paquetes por la ruta alternativa. Sin embargo esta situación no

dura mucho tiempo, ya que el controlador identifica este fallo y reorganiza los grafos para volver a tener un camino principal con sus alternativas, tomando una de las rutas alternativas como vía principal.

Para apagar el entorno y finalizar las pruebas, simplemente hay que introducir el siguiente comando en el mismo terminal en el que se inician los switches:

```
sudo ./scenarios/bucle4_bfd/stop_scenario_bucle4_bfd.sh
```

Una vez apagados todos los switches, ejecutamos el comando `quit` o `exit` en el terminal en el que se arranca el escenario. De esta forma también se cierra Mininet.

Encendido y apagado del escenario con 2 controladores

Para poner en marcha una topología compuesta por 2 controladores, es necesario ejecutar los siguientes comandos, en una terminal shell desde la carpeta "$HOME/OF-in-band-control-plane/testbed/scenarios_ether/":

```
cd $HOME/OF-in-band-control-plane/testbed/scenarios_ether/
sudo python3  ./scenarios/2controllers/scenario_2controllers.py
```

Ahora, al igual que en el escenario con un solo controlador, es necesario arrancar los terminales[2] independientes para c0 y c1 respectivamente. Para conseguirlo se ejecutar los siguientes comandos:

```
#Comandos para el controlador c0
source /home/sdnwifi/sdnenv/bin/activate
ryu-manager --verbose InBandController.py --config-file ./
    scenarios/2controllers/params-controller-c0.conf > /tmp/
    salida-c0 2>&1
```

```
#Comandos para el controlador c1
source /home/sdnwifi/sdnenv/bin/activate
ryu-manager --verbose InBandController.py --config-file ./
    scenarios/2controllers/params-controller-c1.conf > /tmp/
    salida-c1 2>&1
```

Antes de arrancar los demás switches, es necesario ejecutar los siguientes comandos en dos terminales independientes, para ir comprobando que switches son supervisados por cada controlador, y por lo tanto, a que subred se incluyen:

[2]Siendo en este caso 2 en lugar de 1.

```
#Comando para el controlador c0
sudo tail -f /tmp/salida-c0 | grep MANAGED
```

```
#Comando para el controlador c1
sudo tail -f /tmp/salida-c1 | grep MANAGED
```

Es recomendable mantener visibles estas ventanas para poder observar qué switches se conectan a cada controlador.

Y para poder arrancar el resto de switches, ejecutamos en un terminal independiente el siguiente comando:

```
cd $HOME/OF-in-band-control-plane/testbed/scenarios_ether/
sudo ./scenarios/2controllers/start_scenario_2controllers.sh
```

Si se observan los dos terminales donde se han ejecutado los comandos 'tail', se puede apreciar como los switches se enlazan a uno de los dos controladores y comienzan a utilizar la estructura del grafo de rutas contenido en las cabeceras NSH de los paquetes.

Para apagar el escenario y finalizar las pruebas, simplemente hay que introducir el siguiente comando en el mismo terminal en el que se inician los switches:

```
sudo ./scenarios/2controllers/stop_scenario_2controllers.sh
```

Una vez apagados todos los switches, al igual que sucede en el escenario con un solo controlador, ejecutamos el comando `quit` o `exit` en el terminal en el que se arranca el escenario. De esta forma también se cierra Mininet.

Es este escenario también es posible demostrar que existen rutas alternativas, con desactivar uno de los switches ya es posible comprobar como la topología de cualquier subred se actualiza y empieza a emplear alguno de los caminos alternativos.

Bibliografía

[1] J. Doe. «Understanding LLDP: A Comprehensive Guide». En: *Networking Journal* 22.3 (2018), págs. 45-58.

[2] N. Feamster, J. Rexford y E. Zegura. «The road to SDN: An intellectual history of programmable networks». En: *ACM SIGCOMM Computer Communication Review* 44.2 (2013), págs. 87-98.

[3] M. García, A. López y P. Martínez. «LLDP and Network Automation: An Integrated Approach». En: *International Journal of Network Management* 15.2 (2019), págs. 112-129.

[4] *GeeksforGeeks: Architecture of Software Defined Networks (SDN)*. 2023. URL: https://www.geeksforgeeks.org/architecture-of-software-defined-networks-sdn/.

[5] Y. Hu, S. Qu y C. Lee. «SDN: Software Defined Networks». En: *CRC Press* (2016).

[6] IEEE 802.1AB-2005. *IEEE Standard for Local and Metropolitan Area Networks: Link Layer Discovery Protocol*. Inf. téc. 2005.

[7] *Image Source*. Consultado el 03/10/23. 2023. URL: https://gregsowell.com/?p=4442.

[8] S. Jones. «Challenges in Implementing LLDP: A Manufacturer's Perspective». En: *Network World* 34.7 (2017), págs. 89-101.

[9] D. Kreutz et al. «Software-defined networking: A comprehensive survey». En: *Proceedings of the IEEE* 103.1 (2015), págs. 14-76. DOI: 10.1109/jproc.2014.2371999.

[10] B. Lantz, B. Heller y N. McKeown. «A network in a laptop: rapid prototyping for software-defined networks». En: *ACM SIGCOMM Computer Communication Review* (2010). Consultado el 26/09/23. URL: https://www.academia.edu/2615953/A_network_in_a_laptop_Rapid_prototyping_for_software_defined_networks.

[11] J. McCauley et al. «Extending SDN to the Data Plane with OpenFlow». En: *2012 Proceedings IEEE INFOCOM*. Consultado el 26/09/23. 2012, págs. 1-2. URL: https://www.researchgate.net/publication/262154111_No_silver_bullet_Extending_SDN_to_the_data_plane.

[12] N. McKeown et al. «OpenFlow: Enabling Innovation in Campus Networks». En: *ACM SIGCOMM Computer Communication Review* 38.2 (2008), págs. 69-74. DOI: 10.1145/1355734.1355746.

[13] *Mininet Optical Architecture*. Consultado el 28/09/23. 2023. URL: https://mininet-optical.org/architecture.html.

[14] A. M. A. Nascimento, R. R. Lopes y C. E. Rothenberg. «Mininet-WiFi: Emulating software-defined wireless networks». En: *2015 11th International Conference on Network and Service Management (CNSM)*. Consultado el 28/09/23. 2015, págs. 413-418. URL: https://ieeexplore.ieee.org/document/7367387.

[15] A. Soni y S. Sehgal. «SDN: Concepts and Challenges». En: *International Journal of Computer Applications* 135.11 (2016), págs. 1-5. DOI: 10.5120/ijca2016909853.

[16] K. Sood et al. «Virtualized cloud computing: A vision, survey, and research challenges». En: *IEEE Network* 27.5 (2013). Consultado el 16/09/2023, págs. 4-10.

[17] *The Great vSwitch Debate - Part 1*. Consultado el 03/10/23. 2023. URL: https://kensvirtualreality.wordpress.com/2009/03/29/the-great-vswitch-debate-part-1/.

[18] K. Thimmaraju et al. «Taking control of SDN-based cloud systems via the data plane». En: Consultado el 15/09/2023. 2018. DOI: 10.1145/3185467.3185468.

[19] *What is Mininet?* Consultado el 1/1/24. 2023. URL: https://www.researchgate.net/figure/Emulating-Real-Networks-in-Mininet_fig1_287216738.

[20] Y. Zhang, Y. Wen y X. Guan. «Network Function Virtualization: Concepts and Applications». En: *IEEE Communications Magazine* 55.11 (2017), págs. 28-34. DOI: 10.1109/MCOM.2017.1600435.