

Universidad de Alcalá Escuela Politécnica Superior

Máster Universitario en Ingeniería de Telecomunicación

Trabajo Fin de Máster

Diseño e implementación de protocolo
de control escalable en redes IoT para
entornos 6G

ESCUELA POLITECNICA
SUPERIOR

Autor: David Carrascal Acebron

Tutor: Elisa Rojas Sánchez

2023



Máster Universitario en Ingeniería de Telecomunicación



Universidad
de Alcalá

Madrid, 7 de julio de 2023

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Máster Universitario en Ingeniería de Telecomunicación

Trabajo Fin de Máster

**Diseño e implementación de protocolo
de control escalable en redes IoT para entornos 6G**

Autor: David Carrascal Acebron

Tutor: Elisa Rojas Sánchez

Tribunal:

Presidente: Juan Manuel Arco Rodríguez

Vocal 1º: M^a Elena López Guillén

Vocal 2º: Isaias Martinez Yelmo

*A mis hermanas, Natalia y Violeta,
quienes día a día, por oscura que sea la noche,
arrojan luz y esperanza a mi vida.*

Agradecimientos

Quiero empezar agradeciendo y reconociendo a mi tutora, Elisa Rojas, sin la cual este trabajo no habría sido posible. Quien desde segundo de carrera creyó en mí y, aún a día de hoy, sigue apostando día a día en mis capacidades, incluso cuando ni yo mismo soy capaz de verlas. Su destreza y conocimiento, su apoyo incondicional y carisma, su maestría y pasión por lo que hace, y a lo que se dedica, han hecho que etapa, tras etapa académica siga aprendiendo y disfrutando como el primer día. Este trabajo ha sido financiado por subvenciones de la Comunidad de Madrid a través de los proyectos TAPIR-CM (S2018/TCS-4496) y MistLETOE-CM (CM/JIN/2021-006), y por el proyecto ONENESS (PID2020-116361RA-I00) del Ministerio de Ciencia e Innovación de España.

También me gustaría agradecer a mi familia, por su cariño, comprensión e inspiración en estos meses que han sido tan duros para mí. Y qué decir de mis amigos, a los *Caye de Calle*, a los *C de Chill*, a mis estimados *Pueblerinos*, como no, a Pablo y Olga, a mis queridas Noci, a mi señor abuelo de confianza, Boby, a la señorita Laura de Diego, mis compis de la uni y toda la gente nueva que ha llegado a mi vida durante estos meses, a todos vosotros, gracias por las risas y los buenos momentos que hemos compartido juntos. Gracias de verdad.

No puedo terminar sin agradecer a toda la gente del Laboratorio LE34, quienes me alentado a seguir por este arduo camino de la investigación y quienes, con sus consejos y experiencias, han ido conformando al ingeniero que soy a día de hoy.

Sinceramente, mil gracias a todos.

Resumen

En este Trabajo Fin de Máster (TFM) se presenta el diseño e implementación de un protocolo de control escalable de redes Internet of Things (IoT) para entornos Software-Defined Networking (SDN) en la nueva generación de redes móviles, la Sexta Generación de redes móviles (6G). Dicho protocolo de control seguirá un paradigma de control de tipo *in-band*, con el cual se dotará de conectividad a los nodos de la red con el ente de control, empleando el plano de datos para la transmisión de información de control.

En aras de completar el proyecto, se ha partido del análisis de las necesidades y características de las distintas tecnologías que se emplearán en ejecución de los objetivos predefinidos y así discernir aquellas herramientas necesarias para la implementación del protocolo control. Una vez seleccionadas las herramientas, se han estudiado a fondo para realizar una implementación optimizada en la medida de lo posible. Este proyecto ha concluido con la validación mediante emulación del protocolo desarrollado para comprobar el correcto funcionamiento del mismo en distintos casos de uso.

Palabras clave: [6G](#); [IoT](#); [SDN](#); [Control in-band](#); Plano de control

Abstract

In this Master's Thesis (TFM) we present the design and implementation of a scalable control protocol for Internet of Things (IoT) networks in Software-Defined Networking (SDN) environments in the new generation of mobile networks, the sixth generation of mobile technologies (6G). This control protocol will be based on an *in-band* control paradigm, which will provide connectivity between the network nodes and the control entity, using the data plane for the transmission of control information.

To accomplish the project, we first analyse the requirements and characteristics of the different technologies that will be used in the execution of the predefined objectives and, accordingly, we determine the tools necessary for the implementation of the control protocol. Once the tools have been selected, they will be studied in depth in order to carry out an optimised implementation as far as possible. This project concludes with the validation the developed protocol by means of emulation to check its correct operation in different use cases.

Keywords: 6G; IoT; SDN; In-band control; Control plane

“No hay ningún viento favorable para el que no sabe a qué puerto se dirige”

Arthur Schopenhauer.

Índice general

Resumen	v
Abstract	vii
1. Introducción	1
1.1. El Internet de las Cosas y la red 6G	1
1.2. Redes SDN	4
1.3. Objetivos	7
1.4. Estructura del TFM	8
1.5. Contribuciones	9
2. Estado del arte	11
2.1. Red de comunicación 6G	11
2.1.1. Tecnologías habilitadoras	12
2.1.1.1. Banda de los THz	12
2.1.1.2. Próxima generación de MIMO	13
2.1.1.3. AI federada y distribuida	15
2.1.1.4. Plano de datos inteligente	16
2.1.1.5. Infraestructuras flexibles y programables	18
2.1.2. La tecnología IoT	19
2.2.1. Arquitectura de los ecosistemas IoT	20
2.2.3. Redes SDN	21
2.3.1. Arquitectura del paradigma SDN	21
2.3.2. Protocolo OpenFlow	23
2.4. Controladores SDN	23
2.4.1. Ryu	26
2.4.2. ONOS	28
2.5. Software Switches SDN	30
2.5.1. OvS	31
2.5.2. BOFUSS	33
2.5.2.1. Ports	35
2.5.2.2. Packet Parser	36

2.5.2.3.	Flow Tables	38
2.5.2.4.	Group Table	39
2.5.2.5.	Meter Table	39
2.5.2.6.	oflib	41
2.5.2.7.	Communication Channel	42
2.6.	Tecnologías Linux	42
2.6.1.	Interfaz virtual - tun/tap	42
2.6.2.	Interfaz virtual - veth	44
2.6.3.	Herramienta TC	46
2.6.3.1.	Qdiscs	46
2.6.3.2.	Classes	47
2.6.3.3.	Filters	47
2.6.4.	Namespaces	48
2.6.4.1.	Persistencia de las Namespaces	49
2.6.4.2.	Concepto de las Network Namespaces	50
2.6.4.3.	Comunicación inter-Namespaces: Veth	51
2.6.5.	Subsistema inalámbrico de Linux	51
2.6.5.1.	Limitaciones del módulo mac80211_hwsim	53
2.7.	Mininet y Mininet-WiFi	54
2.7.1.	Mininet	54
2.7.2.	Funcionamiento de Mininet	56
2.7.3.	Mininet-WiFi	60
2.8.	Contiki-ng	63
2.8.1.	Simulador Cooja	64
2.9.	Contribuciones en GitHub	65
3.	Diseño del protocolo de control in-band	67
3.1.	Protocolo In-Band	67
3.1.1.	Protocolo IoTorii	68
3.1.1.1.	Operativa del protocolo IoTorii	68
3.1.1.2.	Configuración del protocolo IoTorii	71
3.2.	Plataforma de desarrollo y validación	72
3.3.	Agente SDN	74
3.4.	Agente de control SDN	76
3.5.	Ánalisis funcional de la interfaz del BOFUSS	77
3.5.1.	Binario ofprotocol	78
3.5.2.	Binario ofdatapath	79
3.6.	Ánalisis de la clase UserAP en Mininet-WiFi	82

3.7. Análisis del entorno de depuración del BOFUSS	89
3.7.1. Limpieza del escenario	92
3.7.2. Puesta en marcha del escenario	93
3.7.2.1. Resolución de problemas encontrados	96
3.7.3. Configuración de VS Code	97
4. Implementación y evaluación del protocolo	99
4.1. Entorno de desarrollo y validación	99
4.2. Control in-band en el BOFUSS	100
4.2.1. Puesta a punto de la interfaz de control del Datapath - dpctl	103
4.2.2. Parser para la herramienta dpctl	105
4.2.3. Rollback a la herramienta dpctl	106
4.3. Implementación del protocolo IoTorii	109
4.3.1. Despliegue de la implementación en una RPi	115
4.4. Validación	115
4.4.1. Comprobación funcional	117
4.4.1.1. Tablas de vecinos	119
4.4.1.2. Tablas HLMAC	126
4.4.1.3. Conexiones in-band	130
5. Conclusiones y trabajo futuro	135
5.1. Conclusiones del TFM	135
5.2. Líneas de trabajo futuro	137
Bibliografía	139
Lista de Acrónimos y Abreviaturas	147
A. Anexo I - Pliego de condiciones	149
A.1. Condiciones materiales y equipos	149
A.1.1. Especificaciones Máquina A	149
A.1.2. Especificaciones Máquina B	150
A.1.3. Especificaciones Máquina C	150
B. Anexo II - Presupuesto	151
B.1. Duración del proyecto	151
B.2. Costes del proyecto	152

C. Anexo III - Manuales de usuario e Instalación	155
C.1. Herramienta <code>iproute2</code>	155
C.1.1. ¿Qué es <code>iproute2</code> ?	156
C.1.2. ¿Por qué necesitamos <code>iproute2</code> ?	157
C.1.3. Compilación e instalación de <code>iproute2</code>	157
C.1.4. Comandos útiles con <code>iproute2</code>	159
C.2. Herramienta <code>tcpdump</code>	160
C.2.1. ¿Qué es <code>tcpdump</code> ?	160
C.2.2. ¿Por qué necesitamos <code>tcpdump</code> ?	160
C.2.3. Instalación de <code>tcpdump</code>	161
C.2.4. Comandos útiles con <code>tcpdump</code>	161
C.3. Herramienta Mininet	163
C.4. Herramienta Mininet-WiFi	164
C.5. Herramienta Ryu	165
C.6. Herramienta ONOS	166

Índice de figuras

1.1.	Estudio de las conexiones IoT máximas simultáneas a nivel global [3]	2
1.2.	<i>Roadmap</i> propuesto por el SNS-JU para el desarrollo del 6G [7]	3
1.3.	Paradigma en las redes SDN [18]	5
1.4.	Paradigma control en las redes SDN [19]	7
2.1.	Descomposición genérica de un canal MIMO en subcanales paralelos	15
2.2.	Esquema genérico de un sistema FL [33]	17
2.3.	<i>Slicing</i> de red en segmentos funcionales [37]	19
2.4.	Arquitectura básica IoT [18]	21
2.5.	Arquitectura típica SDN [18]	22
2.6.	Arquitectura básica de agente OpenFlow [43]	24
2.7.	Arquitectura generica de controlador SDN [44]	25
2.8.	Arquitectura del controlador SDN RYU [47]	27
2.9.	Arquitectura del controlador SDN onos [50]	30
2.10.	Arquitectura genérica de un <i>softswitch</i> SDN [51]	31
2.11.	Arquitectura del OvS [53]	33
2.12.	Evolución del BOFUSS	34
2.13.	Arquitectura del BOFUSS [43]	35
2.14.	Parseador de paquetes del BOFUSS [43]	37
2.15.	Estructura de las <i>Group tables</i> del BOFUSS [43]	40
2.16.	Estructura de las <i>Meter tables</i> del BOFUSS [43]	40
2.17.	Proceso de Marshaling y Unmarshaling	41
2.18.	Diagrama de funcionamiento de las interfaces virtuales TUN/TAP [56]	44
2.19.	Comprobación con <code>ethtool</code> de tipo de interfaz virtual.	45
2.20.	Mecanismo de calidad de servicio implementado con clases y sub-clases [60] . .	48
2.21.	Enlace entre interfaces Veth separadas en dos Network Namespaces [18]	52
2.22.	Pipeline de transmisión del módulo mac80211_hwsim [64]	53
2.23.	Pipeline de recepción del módulo mac80211_hwsim [64]	54
2.24.	Arquitectura de Mininet [66]	57
2.25.	Salida por pantalla de la ejecución de la topología por defecto	58
2.26.	Listado de Named Network Namespaces existentes en el sistema	58

2.27. Listado de procesos con referencias a Mininet	59
2.28. Información de contexto sobre el proceso del Host1	59
2.29. Información de contexto sobre el proceso del Host2	60
2.30. Arquitectura básica de Mininet-WiFi [67]	61
2.31. UML de clases de tipo Nodo.	62
2.32. UML de clases de tipo Interfaz.	62
2.33. Gestión de la memoria en un sistema con Contiki OS [68]	64
2.34. Interfaz gráfica del simulador Cooja [70]	65
 3.1. Operativa del protocolo de IoTorii [71]	69
3.2. Mensajes de control en IoTorii [71]	72
3.3. Entorno de emulación real de una cabina de avión a escala 1:1 [74]	73
3.4. Ejecución del binario <code>ofdatapath</code> en modo standalone	80
3.5. Comprobación del puerto de escucha del binario <code>ofdatapath</code>	81
3.6. Diagrama UML de la clase <code>UserAP</code>	83
3.7. Topología básica haciendo uso del <code>UserAP</code> (Basic OpenFlow User-space Software Switch (BOFUSS))	84
3.8. Proceso de debug al BOFUSS	91
3.9. Listado de interfaces inalámbricas en <code>/sys/kernel/debug/ieee80211</code>	93
3.10. Comprobación de si el módulo <code>mac80211_hwsim</code> está cargado	95
3.11. Listado de <code>phy</code> inalámbricas usando el comando <code>iw</code>	96
 4.1. Error en la interfaz de control del BOFUSS desde Mininet-WiFi	103
4.2. Error en la interfaz de control del BOFUSS	104
4.3. Información ofuscada de la nueva herramienta <code>dpctl</code>	105
4.4. Resultado del parser la nueva herramienta <code>dpctl</code>	106
4.5. <i>Rollback</i> a la nueva herramienta <code>dpctl</code>	108
4.6. Funcionamiento de la herramienta antigua sobre Mininet	109
4.7. Diagrama de flujo para la implementación del protocolo IoTorii en el software switch BOFUSS	111
4.8. Diagrama de flujo de la operativa del protocolo IoTorii en el software switch BOFUSS	112
4.9. Entorno de validación haciendo uso del módulo del kernel <code>mac80211_hwsim</code> .	116
4.10. Topología básica a evaluar el funcionamiento de IoTorii	118
4.11. Topología básica representada con el motor de Mininet-WiFi	119
4.12. Intercambio de mensajes de tipo <i>Hello</i> en la topología básica	120
4.13. Tabla de vecinos para el nodo A	121
4.14. Tabla de vecinos para el nodo B	121

4.15. Tabla de vecinos para el nodo C	122
4.16. Tabla de vecinos para el nodo D	122
4.17. Tabla de vecinos para el nodo E	123
4.18. Tabla de vecinos para el nodo F	123
4.19. Tabla de vecinos extraídas de los logs de los <i>software switches</i>	125
4.20. Proceso de difusión de mensajes de tipo <i>SetHLMAC</i> en la topología básica . .	127
4.21. Tabla de HLMAC extraídas de los logs de los <i>software switches</i>	129
4.22. Establecimiento de la topología lógica	130
4.23. Obtención de los PID de todos los nodos de la topología	130
4.24. Comprobación de identificadores de Network Namespaces de los nodos A y B de la topología	131
4.25. Descubrimiento topológico llevado a cabo por Ryu	132
4.26. Conexiones in-band con el controlador Ryu que corre en la interfaz de <i>loopback</i> de la Network Namespace por defecto	133
A.1. Especificaciones de la máquina A	149
A.2. Especificaciones de la máquina B	150
A.3. Especificaciones de la máquina C	150
B.1. Diagrama de Gantt del proyecto	154
C.1. Interfaz de tipo CLI de Tcpdump	162

Índice de tablas

2.1. Resumen de los tipos de Namespaces en el Kernel de Linux	49
2.2. Resumen de contribuciones realizadas durante todo el proyecto del TFM	66
3.1. Comparativa del OvS con el BOFUSS	75
3.2. Comparativa del controlador ONOS con el controlador Ryu	77
4.1. Resumen del entorno de desarrollo y validación	100
B.1. Promedio de horas de trabajo	151
B.2. Presupuesto desglosado del Hardware para el TFM	152
B.3. Presupuesto desglosado del Software para el TFM	152
B.4. Presupuesto total con IVA	153
C.1. Herramientas de Iproute2 frente a las de net-tools	156
C.2. Especificaciones máquina de instalación Mininet	163
C.3. Especificaciones máquina de instalación Mininet-WiFi	164
C.4. Especificaciones máquina de instalación de Ryu	165
C.5. Especificaciones máquina de instalación de ONOS	166

Índice de Códigos

2.1. Manejo de interfaces TUN - TAP	43
2.2. Uso de las interfaces Veths	45
2.3. Casos de uso de las Netns	51
2.4. Ejecución de Mininet con la topología por defecto	56
2.5. Listar Named Network Namespaces	57
3.1. Interfaz CLI del binario ofprotocol	78
3.2. Interfaz CLI del binario ofdatapath	79
3.3. Puesta en marcha del escenario básico	84
3.4. Traza de la puesta en marcha del escenario básico	85
3.5. Puesta en marcha del BOFUSS	89
3.6. Script de limpieza del escenario - clean.sh	92
3.7. Script de puesta en marcha del escenario - launch.sh	94
3.8. Operativa básica de la herramienta hwsim_mgmt	95
3.9. Bloqueo de la interfaz por RF-Kill	96
3.10. desbloqueo de la interfaz por RF-Kill	96
3.11. JSON de depuración con GDB del BOFUSS	97
4.1. Diferencias en las herramientas dpctl	104
4.2. Extracción de flujos con la nueva versión de dpctl	105
4.3. Instalación de las dependencias de la nueva versión de dpctl	107
4.4. Construcción de la nueva versión de dpctl	107
4.5. Puesta en marcha y limpieza del escenario	117
4.6. Lanzamos el controlador de Ryu con la App de topo.discovery	131
C.1. Instalación de las dependencias de Iproute2	158
C.2. Obtención del source de Iproute2	158
C.3. Compilación e instalación de Iproute2	158
C.4. Comandos útiles con iproute2	159
C.5. Instalación del paquete Tcpdump	161
C.6. Comandos útiles con Tcpdump	161
C.7. Instalación de la herramienta git	163

C.8. Instalación de la herramienta Mininet	163
C.9. Instalación de la herramienta Mininet-WiFi	164
C.10. Instalación de la herramienta Ryu	165
C.11. Instalación de la herramienta ONOS	166

1. Introducción

En este primer capítulo, se desea presentar de manera concisa los aspectos más relevantes del Trabajo Fin de Máster (TFM), como son, las redes de dispositivos Internet of Things (IoT), la llegada de la Sexta Generación de redes móviles (6G), y la tecnología habilitadora en dichos entornos, el Software-Defined Networking (SDN). Se explorarán las necesidades actuales de las redes de sensores IoT, se indagará la postulada nueva generación de redes móviles, 6G, y se verá donde entrará las redes SDN, y qué mejoras deberán hacerse para hacer frente a las necesidades imperantes de las próximas redes de sensores.

De igual manera, se establecerán objetivos claros para el TFM y se describirá detalladamente cómo se planea llevarlos a cabo cada uno de ellos. Estos objetivos ayudarán a al diseño y desarrollo de un nuevo protocolo de comunicación de control escalable para redes de sensores en un ámbito de red SDN. De forma adicional, se presentará la estructura general del TFM, describiendo de manera breve los temas que se abordarán en cada capítulo. Por último, se indicarán las contribuciones realizadas en revistas científicas de alto impacto de este proyecto.

1.1. El Internet de las Cosas y la red 6G

Los recientes avances en las comunicaciones móviles junto a la mejora de las capacidades tecnológicas de los elementos hardware han llevado al IoT a un punto álgido, donde, a día de hoy, interconecta billones de objetos entre sí con comunicaciones Machine to Machine (M2M) tanto en entornos particulares, como en entornos industriales [1]. Se puede afirmar que sin lugar a dudas el IoT es parte del hoy y el mañana de Internet, ha revolucionado la forma en la que interactúa con el mundo que nos rodea, permitiendo conectar dispositivos entre sí de forma completamente autónoma a través de la red, proveyendo al humano de entornos inteligentes y adaptativos a las necesidades de la sociedad.

Sin embargo, el aumento exponencial de los dispositivos IoT conectados a las redes de comunicaciones móviles ha generado nuevas necesidades en términos de capacidad, rendimiento, latencia y eficiencia de las redes que deben ser solventadas. Las tecnologías de Quinta Generación de redes móviles (5G) ya se ha propuesto y desplegado comercialmente para dar

soporte a las necesidades de las redes IoT y sus aplicaciones. La arquitectura planteada en 5G daba solución a las necesidades preliminares del IoT haciendo uso de las funcionalidades que traía consigo, como por ejemplo, *enhanced Mobile BroadBand* (eMBB), *massive Machine-Type Communication* (mMTC), *Ultra-Reliable and Low-Latency Communication* (URLLC) [2]. Dichas funcionalidades proveían a los ecosistemas IoT de servicios de alto ancho de banda, baja latencia y optimización del consumo, siendo esta última muy importante para los dispositivos IoT. No obstante, con la rápida proliferación de nuevos de sensores, y con ello, el aumento de las redes IoT según se puede apreciar en la Figura 1.1, los requisitos técnicos necesarios se han visto aumentados para poder seguir manteniendo entornos M2M tal cual se conocían, completamente autónomos, dinámicos e inteligentes.

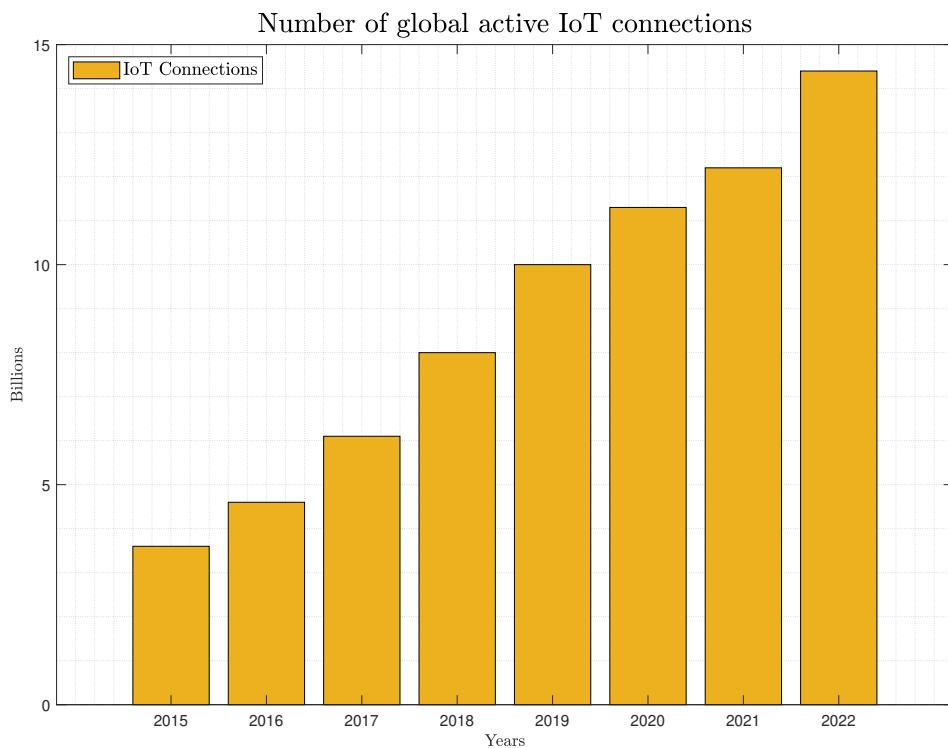


Figura 1.1: Estudio de las conexiones IoT máximas simultáneas a nivel global [3]

Por ello, se necesita una tecnología más avanzada que pueda satisfacer las futuras demandas de las redes IoT, y la arquitectura 6G se postula como una solución a todos los nuevos retos planteados. Por esa razón, para facilitar el desarrollo de las futuras redes IoT la investigación en la siguiente generación de redes móviles ha recibido mucha atención tanto por parte de academia e instituciones, como por parte de la industria [4].

Si bien es cierto que la arquitectura 6G está todavía siendo formulada, se espera que esta pueda proporcionar una Calidad de servicio (QoS) totalmente mejorada frente a la generación anterior, dado sus prestaciones son claramente superiores, como por ejemplo, comunicaciones de latencia muy baja, tasas de velocidad de datos mejoradas y entornos inteligentes de comunicación con satélites.

Por consiguiente, muchos de los esfuerzos de las instituciones están pasando por impulsar las redes 6G-IoT [5]. Desde Europa se está impulsando la carrera por el 6G poniendo sobre la mesa una hoja de ruta dividida en cuatro fases diferenciadas, esperando poder finalizar en 2030 [6]. Dichos esfuerzos están siendo coordinados desde la EU's Smart Networks and Services Joint Undertaking (SNS-JU) la cual tiene como misiones principales impulsar el despliegue del 5G, y situar a los países miembros de la EU a la vanguardía del desarrollo de la próxima generación de redes móviles [7]. Estas misiones planean llevarlas a cabo a través de un detallado una detallada hoja de ruta el cual se puede apreciar en la Figura 1.2.

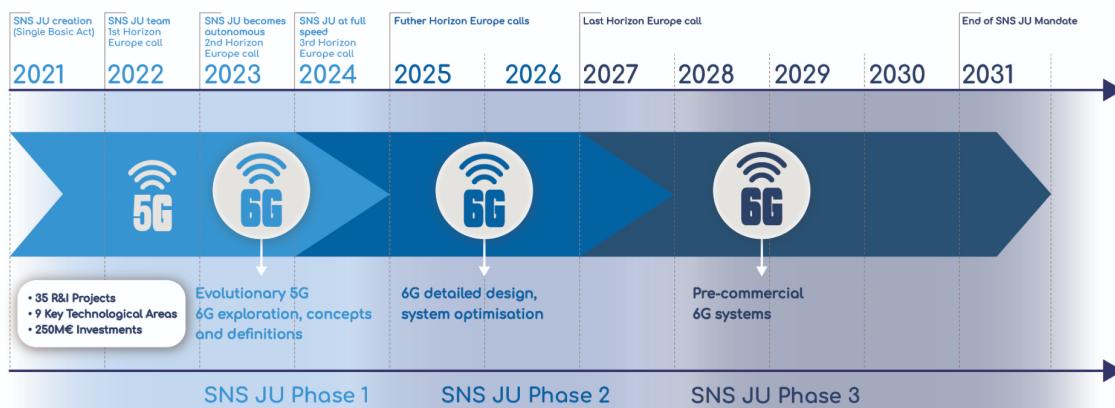


Figura 1.2: *Roadmap* propuesto por el SNS-JU para el desarrollo del 6G [7]

Dicho *roadmap* se compone de varias fases, las cuales se han subdividido en *streams*. Los streams principales son cuatro, actualmente nos encontramos en el stream B del roadmap [8], el cual tiene como objetivo el impulso de la investigación en las áreas tecnológicas habilitadoras del 6G además de la cooperación internacional con estados estratégicos como Estados Unidos. De entre todos los proyectos iniciados en 2023, destacan por ejemplo, Hexa-X [9], iniciativa principal de europa liderada por Nokia financiado por el programa europeo Horizon 2020 que busca desarrollar el prototipado de los sistemas 6G. Otro ejemplo, es el proyecto 6Genesis [10] financiado por Finlandia, que busca la generación de las primeras pruebas de concepto experimentales de redes 6G-IoT. Si nos vamos a proyectos más específicos, podemos encontrar ADROIT6G [11] y 6GTandem [12], ambos han comenzado en enero de 2023 y buscan mejorar y optimizar los sistemas distribuidos de acceso al medio mediante sistemas du-

les en frecuencia empleando procesos de señal *Multiple-Input and Multiple-Output* (MIMO).

Pero el interés en la próxima generación de redes móviles, no es meramente local. En el contexto de Estados Unidos, podemos ver cómo ya la *Federal Communications Commission* (FCC) libera espectro en la banda de los THz en US para hacer pruebas de concepto para el 6G [13]. Algo similar sucede en Corea del Sur, donde ya han planeado lanzar un proyecto piloto de 6G para el 2026 [14].

Como se puede ver, el interés por el 6G es real, y se están realizando esfuerzos exhaustivos para la proliferación de las tecnologías habilitadoras del 6G, por lo que distintas instituciones apuntan a que las primeras redes 6G que podrán ser desplegadas en 2028, pero que su comercialización y la llegada a las personas de a pie no llegará hasta el 2030 [4]. Uno de los aspectos relevantes en la nueva generación de redes móviles, es la arquitectura de interconexión física que se va a plantear. En el 5G, ya se reaprovecho el *backbone* existente de SDN, al cual haciendo uso de flexibilidad y de la programabilidad, se le indujeron modificaciones software para atender las nuevas especificaciones de la arquitectura planteada [2]. Teniendo esto en cuenta, al lector le pueden surgir dudas de si la próxima red de 6G hará uso de las bondades de las redes SDN para impulsar el procesamiento de datos en su arquitectura como parte del *backbone*. Según los informes preliminares [15] [16] [17] que se han presentado en cuanto al diseño de la red 6G, se indica que harán uso del SDN, junto a la tecnología Programming Protocol-independent Packet Processors (P4) y técnicas de inteligencia artificial, del inglés *Artificial Intelligence* (AI)/*Machine Learning* (ML), para la definición de plano de procesamiento de datos y mejorar el rendimiento y orquestación del *backbone* ya existente.

1.2. Redes SDN

Como se ha podido ver, las redes SDN seguirán siendo una tecnología clave en las próximas redes de comunicaciones móviles 6G. La Figura 1.3 muestra cómo con estas redes: se pretende separar el plano de control de los dispositivos intermedios de procesamiento de la red y centralizarlo en entidades denominadas controladores, lo que permitirá una administración más flexible y centralizada de la red.

Antes de que apareciera el concepto de SDN, las redes convencionales solían tener un plano de control unificado en los propios dispositivos, llamado generalmente *Control plane*, en el que se definía la lógica que dictaba cómo se debía llevar a cabo el forwarding de los paquetes, y un plano de datos, conocido como *Data plane*, que se implementaba definiendo su data-path, compuesto por varios bloques de procesamiento para reenviar los paquetes. Ambos

planos, estarían unificados en un sentido lógico en un mismo dispositivo. Sin embargo, con la aparición del paradigma de las redes SDN, como se muestra en la Figura 1.3, los nodos tradicionales de la red verían cómo su plano de control sería delegado a una entidad externa llamada controlador. Este controlador tendría una perspectiva global de toda la red en su conjunto, lo que permitiría una gestión más flexible, dinámica y optimizada.

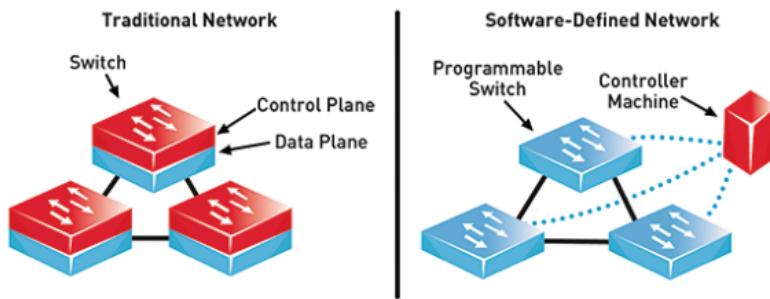


Figura 1.3: Paradigma en las redes SDN [18]

A la hora de la transmisión de la información de control desde el controlador hacia los nodos SDN, se plantean dos puntos importantes. El primero de ellos es qué esquema en la red control se va a plantear, y la segunda que protocolo de comunicaciones se va a emplear. Empezando por los protocolos de comunicación de información de control en redes SDN, el más utilizado es OpenFlow, aunque más adelante se verán en detalle alguno que otro más.

En cuanto a los paradigmas de control SDN, tenemos dos vertientes [19], *out-of-band* e *in-band*. La diferencia entre cada paradigma, ver Figura 1.4, es que en el modelo *out-of-band*, cada nodo SDN tiene un enlace dedicado con el controlador, es decir la información de control tiene una red dedicada por y para ella. El modelo *in-band* por el contrario, se tiene que solo alguno/s de los nodos SDN gestionados tiene un enlace con el controlador, y el resto de equipos emplean ese enlace para hacerle llegar al controlador la información de control. Se quiere señalar que en este último modo la información de control, al no tener una red dedicada para ella misma, viajará de forma conjunta por el plano de datos hasta llegar al controlador.

No hay un paradigma mejor que otro, cada paradigma de control tiene unos pros y unos contras, y será el caso de uso quien predisponga cuál de los dos usar [20]. Por ejemplo, el modelo *out-of-band* es un modelo mucho más caro dado que se tiene un enlace dedicado para comunicación controlador - nodo SDN, pero por ello, también es más seguro dado que el tráfico de control está aislado. Por el contrario, el modelo *in-band*, es mucho más barato dado que se emplea de red de datos para la transmisión de la información de control. Sin

embargo, es un modelo más inseguro, ya que la información de control está expuesta al plano de datos.

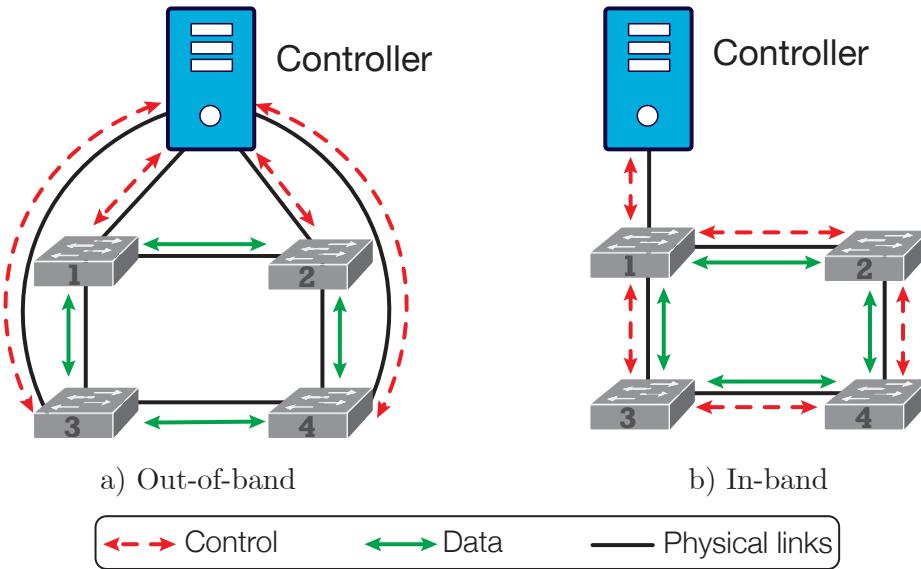


Figura 1.4: Paradigma control en las redes SDN [19]

Uno de los puntos diferenciadores entre ambos paradigmas de control, es la configuración requerida en cada caso. En el modelo de *out-of-band*, no hay apenas configuración requerida, dado que, si la información es de control, tendremos una interfaz de red exclusiva con la cual trabajar. Por el contrario, en el modelo *in-band*, si tenemos que tratar información de control no sabemos con qué interfaz operar. El nodo SDN deberá saber a priori por donde reenviar los paquetes de control. Dicha configuración se obtiene de algún tipo de protocolo de comunicación *in-band*, el cual proporcione a cada nodo de la red la capacidad de alcanzar el nodo que les da acceso al controlador de la red SDN.

Como se ha indicado anteriormente, no hay paradigma mejor que otro, cada cual ofrece unas características propias que tendrán un comportamiento mejor o peor en función del caso de uso. Por ejemplo, en un entorno IoT, donde los dispositivos a lo sumo tienen una única interfaz de comunicaciones, el modelo *in-band* sería ideal. El coste de añadir otra interfaz de comunicaciones no es solo monetario, sino que también impacta en la vida útil del sensor, al tener que alimentar una interfaz de comunicaciones adicional.

1.3. Objetivos

El objetivo principal de este proyecto es conseguir el desarrollo de un protocolo de control *in-band* eficiente para la integración de la tecnología IoT en las futuras de redes de 6G. Como ya hemos introducido, con la llegada del IoT la dimensión de las redes va a crecer exponencialmente. Por lo consiguiente, la complejidad de la administración de dichas redes

va a suponer un gran desafío. Los dispositivos IoT se podrán beneficiar de la integración con las redes SDN en 6G, ya que estas les reportará la flexibilidad y programabilidad requerida para una correcta gestión y administración de cada elemento de la red. Por ello, se pueden resumir los objetivos del TFM en los siguientes puntos:

- **Analizar el estado del arte y necesidades actuales de IoT en 6G.** Se realizará una búsqueda y recolección de información, artículos e informes sobre las demandas del IoT y las bondades preliminares del 6G.
- **Diseño de un protocolo in-band eficiente para IoT integrado con SDN.** Se realizará un estudio previo de las soluciones *in-band* ya implementadas, se analizarán, y se propondrá una solución a medida que cubra los equipos IoT en las redes SDN.
- **Emulación mediante plataformas como Contiki-NG, Mininet y ONOS.** El desarrollo inicial se probará sobre las plataformas más utilizadas en el prototipado de protocolos de comunicaciones.
- **Implementación y despliegue en hardware real** (tarjetas Raspberry Pi, o remotas IoT-LAB), en función de la ejecución del proyecto se estudiará la implementación del protocolo en *hardware* en real.

1.4. Estructura del TFM

En esta sección se indica la estructura organizativa de la memoria del proyecto TFM, haciendo un resumen de los aspectos más significativos de cada capítulo.

Capítulo 1: Introducción. Se hará una breve introducción de la motivación principal que ha originado la realización de este TFM, así como una breve explicación de los aspectos generales y de los objetivos que se quieren alcanzar con el trabajo presentado. Por último, se indicarán las contribuciones que se han conseguido con el mismo, en revistas científicas.

Capítulo 2: Estado del arte. Se indicarán los conceptos fundamentales en relación al proyecto. La motivación de este capítulo es la de establecer un marco teórico lo suficientemente consistente para abordar el diseño del protocolo de comunicación *in-band*.

Capítulo 3: Diseño del protocolo de control In-Band. Se analizará soluciones anteriores *in-band*, debatiendo las funcionalidades básicas que debe tener el protocolo. Se explicará el funcionamiento del mismo y se decidirá la plataforma para implementarlo.

Capítulo 4: Desarrollo y evaluación del protocolo. Se describirá el desarrollo realizado en la plataforma elegida. Señalando aquellas partes que se consideran de mayor importancia. Por último, se incluirá una evaluación del mismo.

Capítulo 5: Conclusiones y trabajo futuro. Se terminará la memoria de este TFM con las conclusiones, y se presentarán las vías de trabajo a futuro que tiene este proyecto.

Bibliografía y referencias. Se añadirán todos los artículos, libros, materiales consultados y empleados en la elaboración de esta memoria. Se seguirá el estilo de citación del Institute of Electrical and Electronics Engineers (IEEE), siguiendo las recomendaciones oficiales de la normativa sobre TFMs de la Universidad de Alcalá (UAH).

Anexos. Se incluirán todos los manuales de usuario e instalación que se consideren oportunos. De forma adicional, se añadirán las características técnicas del *hardware* con el cual se ha desarrollado este TFM. Por último, se hará un presupuesto que incluya el coste de mano de obra, material y gastos generales.

1.5. Contribuciones

Este TFM ha proporcionado significativas contribuciones a la comunidad científica, incluyendo tres publicaciones en revistas de alto impacto indexadas en el JCR (3 Q2). A continuación, se presentan estas contribuciones en resumen.

Artículos de revistas científicas de alto impacto.

1. Rojas, E., Hosseini, H., Gomez, C., **Carrascal, D.** and Cotrim, J.R., 2021. Outperforming RPL with scalable routing based on meaningful MAC addressing. *Ad Hoc Networks*, 114, p.102433. (JCR **Q2**)
2. Alvarez-Horcajo, J., Martinez-Yelmo, I., Rojas, E., Carral, J.A. and **Carrascal, D.**, 2022. ieHDDP: An Integrated Solution for Topology Discovery and Automatic In-Band Control Channel Establishment for Hybrid SDN Environments. *Symmetry*, 14(4), p.756. (JCR **Q2**)
3. **Carrascal, D.**, Rojas, E., Lopez-Pajares, D., Alvarez-Horcajo, J. and Carral, J.A., 2023. A Comprehensive Survey of In-Band Control in SDN: Challenges and Opportunities. *Electronics*, 12(6):1265. (JCR **Q2**)

2. Estado del arte

En el presente capítulo se describirán los conceptos fundamentales relacionados con el proyecto, así como las diversas herramientas que se utilizarán mayoritariamente. El propósito de este capítulo es establecer un marco teórico sólido que permita abordar el análisis y el diseño del protocolo de comunicación *in-band* de manera óptima antes de su desarrollo. Finalmente, se hará referencia a las contribuciones y la documentación generada con el fin de difundir los contenidos del proyecto a través de la plataforma *GitHub*.

2.1. Red de comunicación 6G

Las redes de comunicación de sexta generación, están en desarrollo para ofrecer una conectividad aún más rápida, confiable y eficiente que las generaciones anteriores. A medida que la demanda de datos continúa creciendo exponencialmente y surgen nuevas aplicaciones y tecnologías, como el IoT, la realidad aumentada y la AI, se espera que el 6G proporcione una infraestructura de red sólida y escalable para satisfacer estas necesidades futuras [4]. Los principales puntos claves de esta nueva generación se pueden resumir en los siguientes puntos.

- **Velocidad y capacidad extremadamente altas:** Una de las principales características del 6G será la velocidad y capacidad extremadamente altas, superando con creces las capacidades del 5G. Se espera que el 6G alcance velocidades de terabits por segundo, lo que permitirá una transmisión de datos ultra rápida y un soporte eficiente para aplicaciones de alta demanda, como son la transmisión de video en 8K, realidad virtual y realidad aumentada de alta calidad.
- **Latencia ultrabaja:** El 6G aspira a lograr una latencia ultrabaja, reduciendo aún más el tiempo de respuesta de la red. Se espera que la latencia en el 6G sea de solo unos pocos milisegundos, lo que permitirá aplicaciones en tiempo real de misión crítica, como cirugías remotas, vehículos autónomos y aplicaciones de realidad virtual y aumentada altamente interactivas.
- **Conectividad ubicua:** El 6G tiene como objetivo proporcionar conectividad ubicua, extendiéndose más allá de las áreas urbanas y llegando a áreas remotas y rurales.

haciendo uso de radio enlaces satelitales de órbita baja (LEO). Se espera que el 6G aborde la brecha digital y brinde conectividad global a nivel mundial, permitiendo una mayor inclusión digital y oportunidades equitativas para todos.

- **Integración de tecnologías emergentes:** El 6G se construirá sobre tecnologías emergentes, como inteligencia artificial (IA), ML, llegando a proponer la computación cuántica y nanotecnología. Estas tecnologías avanzadas permitirán el desarrollo de sistemas de red más inteligentes y autónomos, optimizando la eficiencia y la capacidad de adaptación de la red.
- **Sostenibilidad y eficiencia energética:** El 6G se enfocará en la sostenibilidad y la eficiencia energética para reducir su impacto ambiental. Se espera que las redes 6G sean más eficientes en términos de consumo de energía, al tiempo que brinden una mayor capacidad y rendimiento. Además, se explorarán nuevas técnicas de transmisión de energía y comunicación inalámbrica para impulsar la eficiencia energética en dispositivos y redes.

2.1.1. Tecnologías habilitadoras

Como se comentó en el capítulo de introducción (Capítulo 1), esta nueva generación de red móvil aún es prematura y no tiene unos estándares claros y definidos sobre como se va a llevar a cabo, sin embargo, ya empieza a haber propuestas sobre las tecnologías habilitadoras que harán del 6G una realidad tarde o temprano. Desde el proyecto líder europeo 6G-Flagship[9] y la organización One6G [21] ya se apuntan a una serie de tecnologías claves, las cuales, se indican a continuación.

2.1.1.1. Banda de los THz

Según se ha indicado, con el 6G, se espera que fusione los mundos digital y físico en todas sus dimensiones, brindando a los usuarios una experiencia holográfica, háptica y multisensorial. Según la Unión Internacional de Telecomunicaciones, del inglés *International Telecommunication Union* (ITU), estas aplicaciones surgirán en la próxima década y se caracterizarán por requerir comunicaciones altamente exigentes [22]. Además, algunas aplicaciones demandarán funcionalidades que los sistemas móviles actuales no proporcionan, como una detección precisa, mapeo y localización. Un ejemplo destacado de caso de uso es la telepresencia holográfica¹. La transmisión de hologramas 3D en su forma más básica requiere una capacidad de más de 4 Tbps [23]. La capacidad de detectar y comprender el entorno permitirá a la red predecir el movimiento de los usuarios sin información explícita, creando así una experiencia inmersiva a distancia. Asimismo, las comunicaciones de alta

¹<https://blogthinkbig.com/peoplefirst/telepresencia-holografica>

velocidad de datos y baja latencia necesarias para la automatización de fábricas se beneficiarán de las transmisiones a terabits por segundo (Tbps).

Con el fin de satisfacer esta necesidad, las comunicaciones en terahercios (THz) se han identificado como una opción prometedora para la capa física en 6G, ya que tienen el potencial de permitir velocidades de datos de terabits por segundo y ofrecer servicios de detección, mapeo y localización [24]. En la Conferencia Mundial de Radiocomunicaciones de 2019 (CMR2019)², la ITU identificó un espectro de 137 GHz entre 275 y 450 GHz que se puede utilizar para las comunicaciones en terahercios [25]. Esto se suma al espectro ya asignado por debajo de los 275 GHz, lo que proporciona un total de 160 GHz en la gama de subterahercios. En 2017, el grupo de trabajo IEEE 802 completó el primer estándar inalámbrico para frecuencias portadoras alrededor de los 300 GHz (IEEE Std 802.15.3d-2017) [26, 27]. Esto establece una base sólida para el desarrollo y la implementación de las comunicaciones en terahercios en el contexto de 6G. Aunque, también aparecen nuevos retos de capa física a solventar, dado que según subimos en frecuencia la atenuación de la señal aumentará de forma quasi-proporcional.

2.1.1.2. Próxima generación de MIMO

Utilizar múltiples antenas conlleva una serie de ventajas fundamentales. Estas ventajas están condicionadas por el conocimiento del canal en el transmisor y/o receptor, las propiedades del canal de propagación (como características multirayecto y atenuación) y el tipo de enlace, ya sea punto a punto o multiusuario. Además, la cooperación entre las señales transmitidas o recibidas y el procesamiento conjunto de las antenas también juegan un papel importante.

El diseño de soluciones de múltiples entradas y múltiples salidas (MIMO, por sus siglas en inglés *Multiple Input - Multiple Output*) depende en gran medida del escenario específico que se considere, pero generalmente se suele plantear una descomposición genérica del canal donde se vaya a trabajar en una serie de subcanales M paralelos (Ver Figura 2.1). No existe una única solución MIMO universal. Al diseñar sistemas MIMO, es esencial tener en cuenta las limitaciones prácticas y encontrar un equilibrio entre las ganancias básicas del MIMO [28]. Las ganancias de estos sistemas se pueden resumir en los siguientes puntos.

- **Ganancia por procesado en Array.** La combinación coherente de M antenas, sin importar la distancia entre ellas o si funcionan en modo de transmisión o recepción, puede mejorar la relación señal-ruido (SNR) en un máximo de 10 veces. En otras palabras, la SNR puede mejorarse en un máximo de $10 * \log_{10}(M)$ dB mediante la

²<https://www.itu.int/es/ITU-R/conferences/wrc/2019>

técnica de combinación de las M antenas. Esto se logra mediante la formación de haz (*beamforming*), la cual nos permite “apuntar” de una forma más precisa con el lóbulo principal de nuestro diagrama de radiación al sistema receptor. Existen múltiples técnicas para llevar a cabo un *beamforming*, desde alimentaciones diferentes en las M antenas, a desfases, jugando con la combinación resultante de todas ellas.

- **Multiplexación y ganancia por acceso múltiple.** La combinación de antenas proporciona otra ventaja fundamental, la capacidad de eliminar señales no deseadas controlando los pesos de las antenas de tal manera que los frentes de onda superpuestos se cancelen en canales específicos. Esto se aprovecha de varias formas, como en el caso del acceso múltiple por división espacial (SDMA) en canales multiusuario o en la multiplexación espacial en canales punto a punto. En general, se asume que el número de señales es igual o menor al número de antenas. Con un conjunto de M antenas, es posible separar hasta M usuarios sin interferencias. Esta capacidad de separación espacial permite mejorar significativamente la eficiencia y el rendimiento de los sistemas de comunicación, al permitir la transmisión simultánea y la recepción de múltiples señales independientes en el mismo espectro.
- **La diversidad de antenas y *hardening* del canal.** Son conceptos relacionados con los desvanecimientos causados por la propagación multirayecto. A pequeña escala, estos desvanecimientos espaciales pueden ocurrir debido a la presencia de múltiples trayectos de señal. Si las distancias entre las antenas son lo suficientemente grandes, cada antena experimentará un canal que no está correlacionado con las demás antenas. Este fenómeno se aprovecha transmitiendo o recibiendo información redundante en paralelo a través de las múltiples antenas. Al hacerlo, se logra una diversidad espacial en el sistema, lo que implica que cada antena experimenta diferentes condiciones de desvanecimiento. Como resultado, si una antena se ve afectada por un desvanecimiento, es probable que otras antenas no se vean afectadas de la misma manera. Esta diversidad de antenas proporciona robustez al canal frente a los desvanecimientos. Al transmitir o recibir información redundante a través de múltiples antenas, se pueden mitigar los efectos de los desvanecimientos selectivos de trayecto y mejorar la calidad de la señal.

Con el fin de aprovechar al máximo las ganancias del MIMO, se ha observado una tendencia hacia arrays de antenas cada vez más grandes, ya sea en configuraciones cúbicas, conocido como “massive MIMO” [29], o distribuidos en toda la zona de servicio. En respuesta a esta tendencia, se introdujo el MIMO de dimensión completa en la versión 13 del estándar LTE. En el caso de 5G NR, el 3GPP especificó 32 antenas en la versión 15, y se espera que este número aumente en futuras versiones para habilitar el MIMO masivo. En el ámbito de la 6G, se ha introducido el concepto de múltiples matrices D-MIMO, conocido como

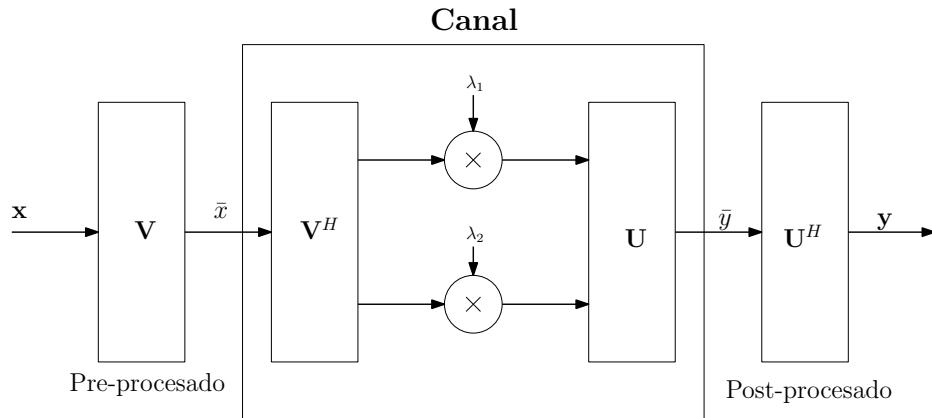


Figura 2.1: Descomposición genérica de un canal MIMO en subcanales paralelos

MIMO masivo modular (mmMIMO) [30]. Este concepto incluye D-MIMO estructurado y diversas variantes de CoMP (Coordinated Multi-Point), incluyendo la transmisión conjunta como casos especiales. En [30] se presentan ejemplos de implementación de prototipos y se describen los pasos de normalización asociados.

Estas tendencias en la evolución del MIMO reflejan la importancia de utilizar arrays de antenas cada vez más grandes y distribuidos para mejorar el rendimiento y la capacidad de los sistemas de comunicación. El enfoque en el MIMO masivo y el mmMIMO en el 6G tiene como objetivo proporcionar una mayor capacidad y eficiencia espectral, así como mejorar la cobertura y la calidad de la señal en entornos complejos.

2.1.1.3. AI federada y distribuida

La AI y el ML se encuentran entre las tecnologías fundamentales que están moldeando el futuro de Internet tal cual lo conocemos. Estas tecnologías están revolucionando la forma en que se recopilan y analizan los datos para obtener una comprensión más precisa y relevante de los procesos clave, respaldando así la toma de decisiones en diversos campos como las *smart cities*, la *Industry 4.0*, *e-Health*, y la agricultura inteligente. La creciente heterogeneidad de las redes a gran escala y la necesidad de satisfacer los diversos requisitos de los usuarios demandan el uso de enfoques basados en AI/ML [31]. La AI representa una herramienta invaluable para abordar problemas en redes que anteriormente se consideraban intratables debido a su complejidad o a la falta de modelos y algoritmos adecuados.

Un enfoque común para construir un sistema de AI implica transmitir todos los datos generados por los dispositivos finales a la nube, donde se lleva a cabo la construcción/formación del modelo y la posterior inferencia. Sin embargo, las grandes cantidades y la complejidad

de los datos a intercambiar a menudo superan las capacidades de la infraestructura de red, lo que ocasiona problemas de sobrecarga y congestión en la comunicación de datos.

Para superar estas dificultades, se han propuesto técnicas de aprendizaje e inferencia distribuidos. En este enfoque, los componentes de AI dedicados al entrenamiento/inferencia se distribuyen en los dispositivos del *edge*, lo cual alivia la necesidad de transferir enormes volúmenes de datos hasta la nube y permite aprovechar de manera eficiente los recursos de computo disponibles a lo largo de la red. Por lo tanto, el aprendizaje y la ejecución distribuidos tienen un gran potencial para descargar las tareas de cálculo, lo que permite aumentar la velocidad de aprendizaje, acercar el procesamiento de datos reduciendo la latencia y la sobrecarga de transmisión, y respetar las restricciones de privacidad. Una de las técnicas más sonadas para llevar a cabo este cometido es el aprendizaje federado.

El *Federated Learning* (FL) es un enfoque descentralizado y recientemente introducido en el cual el aprendizaje se realiza de manera distribuida en cada terminal, dispositivo o entidad, permitiéndoles compartir conocimiento sin necesidad de intercambiar datos raw hacia la nube [32]. Una característica importante de FL es que, al mantener los datos generados por los usuarios de forma local, puede preservar la privacidad y la seguridad de los datos, siempre y cuando se sigan ciertas directrices de diseño. Sin embargo, el entrenamiento en entornos tan heterogéneos, como móviles, coches inteligentes, centros de datos, etc., plantea nuevos desafíos para el aprendizaje automático a gran escala y las optimizaciones asociadas. Por ello, se está enfatizando los algoritmos de agregación de parámetros de cada cluster.

A continuación, en la Figura 2.2, se puede apreciar la operativa básica de un sistema FL. Primero se descarga en cada cluster un modelo genérico, el cual inicialmente será compartido por todos los clusters de la red. Localmente se irá entrenando cada modelo con sus restricciones y características intrínsecas (de ahí la heterogeneidad anteriormente mencionada). Según el sistema de FL, periódicamente o de forma asíncrona se irán mandando los parámetros de la red entrenada de forma local, que no los datos raw, de ahí la ganancia en privacidad y seguridad de los mismos. Por último, con todos los parámetros de todos los clusters centralizados, se sigue una política de agregación para optimizar el funcionamiento de los modelos de forma global.

2.1.1.4. Plano de datos inteligente

Uno de los conceptos más importantes dentro del desarrollo de un plano de datos inteligente es el término *In-network computing*, el cual describe el paradigma de delegar funciones

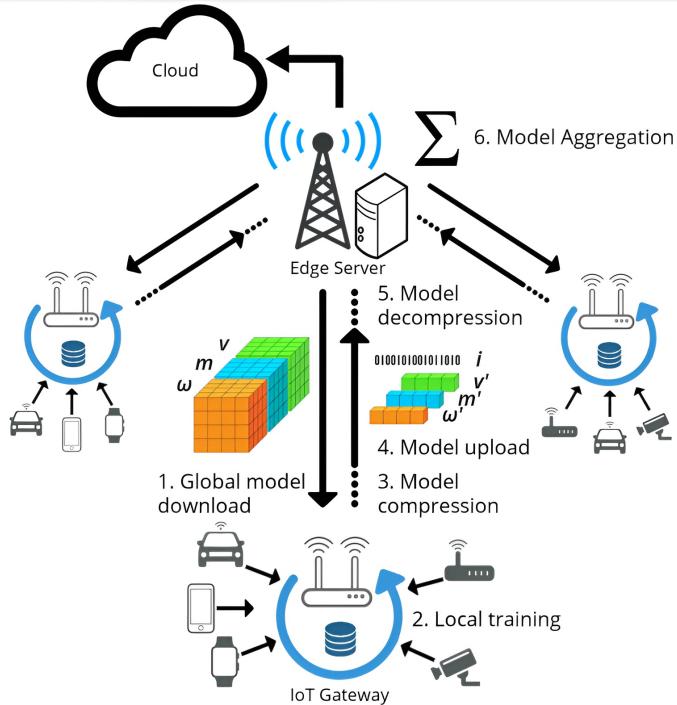


Figura 2.2: Esquema genérico de un sistema FL [33]

de procesamiento sobre la capa de aplicación al plano de datos [34]. Este concepto no es una novedad, pero el auge de la AI/ML junto con el creciente rendimiento de los equipos intermedios de la red, abren la posibilidad de no solo la computación en red, sino la posibilidad de realizar predicciones en aplicaciones y casos de uso de carácter háptico generando una QoS mejorada reduciendo la latencia considerablemente. Hasta la fecha, este tipo de operativas se lleva a cabo con tarjetas de red programables de alto rendimiento, que permiten procesar hasta 10.000 millones de paquetes por segundo [34]. Esto nos permite procesar el tráfico a medida que se transmite, lo que reduce el consumo de recursos de computación y energía de los dispositivos finales.

Según [35], los principales potenciales de la computación en red son (1) una reducción significativa de la latencia y un aumento del rendimiento para determinadas operaciones, lo que resulta especialmente interesante en contextos orientados al rendimiento, y (2) una reducción de la carga de la red. Sin embargo, los casos de uso en los que la computación en red puede explotarse de forma razonable están limitados por una serie de requisitos: (1) Debe lograrse una reducción significativa de la carga de tráfico de red, (2) no deben requerirse cambios significativos a nivel de aplicación. Algunos trabajos pretenden utilizar la computación en red para mejorar la eficiencia de las aplicaciones. A continuación, se indica alguno de ellos.

P4DNS [36] presenta una solución para reducir la latencia hasta que se obtiene una respuesta DNS. El parser del equipo P4 inspecciona todos los paquetes entrantes y extrae sus cabeceras hasta la cabecera DNS. Si el paquete es una consulta DNS y la respuesta correspondiente está disponible en la tabla de caché DNS que tiene el switch, el dispositivo de red responde activamente a la consulta, ahorrando todo el proceso de resolución de la consulta DNS. Para ello basta con modificar el paquete en el sentido de que se intercambian las direcciones de origen y destino, y se añade la cabecera de respuesta DNS. Los autores han evaluado su desarrollo preliminar de la herramienta y demuestran que P4DNS es capaz de reducir drásticamente la latencia de todas las resoluciones DNS.

2.1.1.5. Infraestructuras flexibles y programables

En las redes móviles de la próxima generación, la conectividad estará presente en todas partes, lo que plantea la necesidad de gestionar estas redes extremadamente complejas que abarcan múltiples dominios, incluso llegando hasta el *edge* más extremo. Esto implica utilizar agentes de control para administrar toda la red heterogénea de recursos. A medida que aumenta el número de dispositivos que participan en la topología de la red, también aumenta proporcionalmente la complejidad de su gestión y control. Por lo que se requiere de un paradigma de red que nos brinde una flexibilidad y programabilidad suficiente para hacer frente a estos nuevos retos, para lo cual se requieren las tecnologías SDN y las funciones virtualizadas de red, del inglés *Network Function Virtualization* (NFV).

El objetivo principal que tienen que cumplir las infraestructuras del 6G, es traducir automáticamente los requisitos del usuario en estrategias de despliegue y operación de la red. Para lograr este objetivo, es necesario: i) aprovechar el uso de big data para recopilar información sobre el funcionamiento de la red y el rendimiento de los usuarios, lo cual es necesario para la automatización y adquirir visibilidad; ii) desarrollar modelos de inteligencia artificial capaces de predecir la experiencia del usuario (QoE) y el rendimiento de la red, con el fin de determinar si los cambios en la red podrían afectar el rendimiento; y iii) explorar los planos de datos programables de la red, como aquellos basados en P4 y/o OpenFlow”, e integrar sus respectivos controladores de red en el plano de control de las redes de próxima generación.

Además de admitir e interconectar dinámicamente una amplia variedad de dispositivos que pueden escalar según la demanda de capacidad y admitir interfaces declarativas de administración y orquestación (MANO), otro caso de uso que se explorará es la capacidad de garantizar el aislamiento del rendimiento en redes tan heterogéneas y descentralizadas. La

red debe poder dividirse (*slicing*) en segmentos de rendimiento más pequeños que puedan garantizar el retardo, el aislamiento del tráfico y/o la capacidad de un subconjunto de servicios en comunicación (Ver Figura 2.3). Para lograr este aislamiento del rendimiento, es necesario explorar: i) algoritmos inteligentes de detección del rendimiento del tráfico que puedan predecir patrones de tráfico a una escala muy detallada; ii) mecanismos de control dinámico de asignación de recursos de red que reprogramen los recursos en función de métricas en tiempo real y tráfico previsto; y iii) *network digital twins*, que proporcione capacidades de simulación, planificación y reproducción de la red a operar.

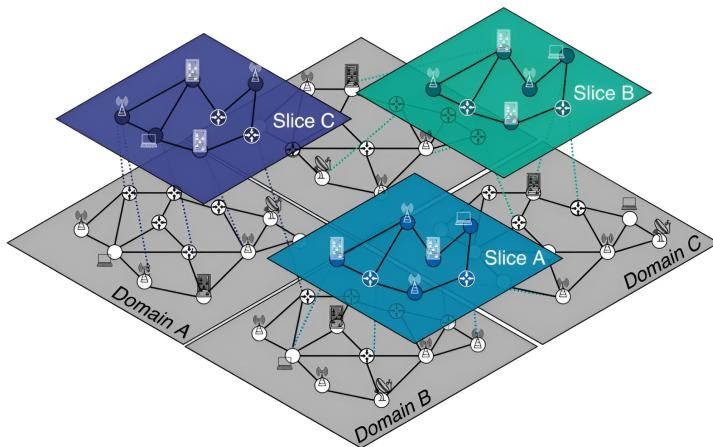


Figura 2.3: *Slicing* de red en segmentos funcionales [37]

En resumen, el 6G promete llevar la conectividad y las comunicaciones inalámbricas a un nuevo nivel, superando las capacidades del 5G y habilitando una amplia gama de aplicaciones y servicios avanzados. Con velocidades ultra altas, latencia ultrabaja, conectividad ubicua y la integración de tecnologías emergentes. Por todo ello, se puede afirmar que el 6G está destinado a impulsar la transformación digital en diversas industrias y habilitar un futuro aún más conectado e inteligente.

2.2. La tecnología IoT

La tecnología IoT tiene como premisa conectar cualquier dispositivo que tenga capacidad de cómputo. Esto implica que objetos que actualmente no están conectados a Internet puedan comunicarse e interactuar con personas y otros objetos. El IoT representa una transición tecnológica en la cual los dispositivos, al estar conectados a Internet, adquieren inteligencia y pueden crear entornos inteligentes para los seres humanos. Cuando los objetos pueden ser

controlados de forma remota a través de una red, se habilita una estrecha integración entre el mundo físico y las máquinas, lo que permite mejoras en áreas como medicina, automatización y logística [38].

El ecosistema del IoT es amplio y puede parecer caótico debido a la gran cantidad de componentes y protocolos involucrados. En lugar de considerar la IoT como un término único, es recomendable verla como un conjunto de conceptos, protocolos y tecnologías bajo un mismo propósito: la interconexión de “cosas” con Internet. Si bien la mayoría de los elementos de la IoT están diseñados para brindar numerosos beneficios en términos de productividad y automatización, también plantean nuevos desafíos, como la gestión de la gran cantidad de dispositivos que aparecerán en las redes y la gestión de la gran cantidad de datos y mensajes que generan [39].

2.2.1. Arquitectura de los ecosistemas IoT

Los ecosistemas IoT son muy variopintos en función de su caso de uso, por lo que actualmente en la industria hay diferentes propuestas sobre los posibles *stacks* de protocolos que se pueden utilizar. La mayor parte de ellos se pueden resumir en una arquitectura genérica que se va a exponer a continuación [39].

- Capa de Percepción (*Perception Layer*): En esta capa se asigna un significado físico a cada objeto. Consiste en sensores de diferentes tipos como etiquetas RFID, sensores infrarrojos u otras redes de sensores que pueden detectar información como temperatura, humedad, velocidad, ubicación, etc. Esta capa recolecta información útil a partir de los sensores vinculados a los objetos, convierte dicha información en señales digitales que posteriormente se delegarán a la Capa de Red para su posterior transmisión.
- Capa de Red (*Network Layer*): El propósito de esta capa es recibir la información en forma de señales digitales desde la capa de Percepción y transmitirla a los sistemas de procesamiento en la capa de *Middleware*. Esto se llevará a cabo a través de diversas tecnologías de acceso como WiFi, BLE, WiMaX, ieee802154 y con protocolos como IPv4, IPv6, MQTT.
- Capa de Middleware (*Middleware Layer*): En esta capa se procesa la información recibida de todos los sensores. Aquí se pueden incluir tecnologías como computación en la nube o sistemas de gestión, que aseguran un acceso directo a bases de datos donde se puede almacenar toda la información recolectada. Al tener una gran cantidad de información centralizada, generalmente se aplican sistemas de inteligencia artificial para procesar la información y tomar decisiones predictivas totalmente automatizadas.

Estos sistemas suelen utilizarse para analizar el clima, la contaminación o el tráfico en las ciudades.

- Capa de Aplicación (*Application Layer*): La finalidad de esta capa es desarrollar aplicaciones IoT para el usuario final. Estas aplicaciones se valen de los datos procesados para ofrecer funcionalidades al usuario final, por ejemplo, una aplicación del tiempo.
- Capa de Negocio (*Business Layer*): Esta capa, aunque un tanto abstracta, se suele añadir para representar la gestión de múltiples aplicaciones y servicios IoT.

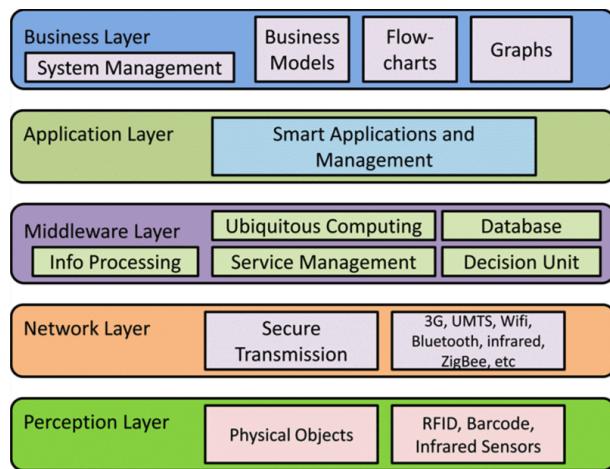


Figura 2.4: Arquitectura básica IoT [18]

2.3. Redes SDN

El paradigma SDN [40] se refiere a una arquitectura de red en la que se separa el plano de control de la red para centralizarlo en un controlador único. Esta estructura permite lograr una administración de red más centralizada y flexible [40]. La idea del SDN comenzó a gestarse en la Universidad de Stanford en 2003, cuando el profesor asociado de ese entonces, Nick McKeown, planteó las limitaciones de las redes convencionales y la necesidad de replantear cómo operaban los *backbones* [41]. En 2011, se acuñó el término SDN, al mismo tiempo que se lanzó la organización Open Networking Foundation (ONF) [42], encargada de establecer estándares y promover la difusión del SDN.

2.3.1. Arquitectura del paradigma SDN

La arquitectura SDN se destaca por su dinamismo, rentabilidad y adaptabilidad, lo que la convierte en una solución ideal para las demandas actuales de las redes de comunicaciones.

Como se ha mencionado anteriormente, esta arquitectura se basa en la separación del plano de control y el plano de datos, trasladando el control a una entidad central llamada controlador. A través de esta entidad, se ofrecen interfaces que permiten a las aplicaciones de servicios de red utilizarlas. Esto hace que el control de la red sea programable directamente, lo que agiliza y dinamiza su gestión.

La arquitectura SDN se divide en tres capas, como se muestra en la Figura 2.5. La primera capa es el plano de datos, donde se encuentran todos los elementos de red responsables del reenvío de los datos. La segunda capa es el plano de control, compuesto por diferentes controladores SDN. Por último, la capa de aplicación alberga todas las aplicaciones que se comunican con el controlador SDN.

Estas capas se comunican entre sí a través de interfaces abiertas. Por ejemplo, la interfaz *Southbound* permite programar el estado de reenvío de los elementos de red en el plano de datos. Por otro lado, la interfaz *Northbound* permite la comunicación entre las aplicaciones y los controladores SDN, permitiendo la obtención de datos y el ajuste de parámetros mediante una API-Rest. Además, existen otras interfaces, como *Westbound* y *Eastbound*, que se han consolidado en los últimos años para interconectar controladores y establecer políticas comunes entre diferentes dominios SDN.

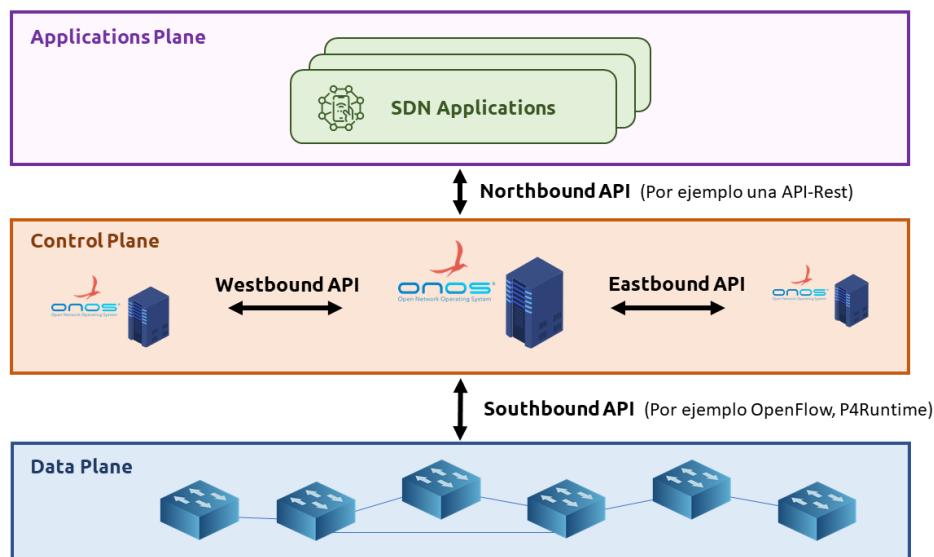


Figura 2.5: Arquitectura típica SDN [18]

2.3.2. Protocolo OpenFlow

Existen varios protocolos para el control de los elementos de red desde el controlador, pero el más ampliamente utilizado es OpenFlow. OpenFlow es un protocolo de la interfaz *Southbound* que establece la comunicación entre los controladores SDN y los elementos de red para configurar el plano de forwarding de estos últimos. La especificación de este protocolo está definida por la ONF³, y cuenta con varias versiones, siendo la más reciente la versión 1.5.1 lanzada en 2015.

El concepto fundamental en OpenFlow es el flujo (*flow*), el cual está compuesto por paquetes que se han clasificado según reglas específicas. Estas reglas se encuentran en las tablas de flujo (*flow table*) y suelen estar relacionadas con los puertos de entrada o los valores de los campos de cabecera del paquete. Cuando los criterios de una regla coinciden con los valores de un paquete entrante, se produce una coincidencia (**match**).

Cuando se produce una coincidencia (*match*), el paquete se somete a una serie de instrucciones asociadas a la regla con la que ha coincidido. Estas instrucciones pueden incluir la medición del paquete, la aplicación de acciones específicas o el enrutamiento hacia otra tabla de flujo. De esta manera, al completar las tablas de flujo con reglas suministradas por el controlador SDN, se configura el estado de reenvío del switch en cuestión [40]. A continuación, en la Figura 2.6, se puede apreciar los elementos básicos que suele contener un switch Openflow. Más adelante en la Sección 2.5.2 sobre *software switches* se estudiarán en profundidad los elementos básicos que componen un agente SDN que implementa un agente Openflow.

2.4. Controladores SDN

Los controladores SDN, también conocidos como sistemas operativos de red, son una pieza clave en los entornos SDN dado que tienen una vista global de toda la red que gestionan, que flujos atraviesan la red, estadísticas, usuarios finales, modelos y características de dichos equipos. Este controlador tiene la funcionalidad de interconectar los recursos disponibles en la red que gestiona, a aplicaciones o servicios que corran encima de él [40]. De esta forma, cada vez que se quiera añadir una nueva funcionalidad, solo se tendrá que programar un nuevo servicio que corra encima del controlador que gestiona la red, y este a su vez se encargará de traducir las demandas del servicio a políticas de red a cada dispositivo impactado. En este matiz se puede llegar a apreciar el sentido de nombre del paradigma SDN, ya que estamos definiendo por software el comportamiento intrínseco de la red.

³<https://www.opennetworking.org/software-defined-standards/specifications/>

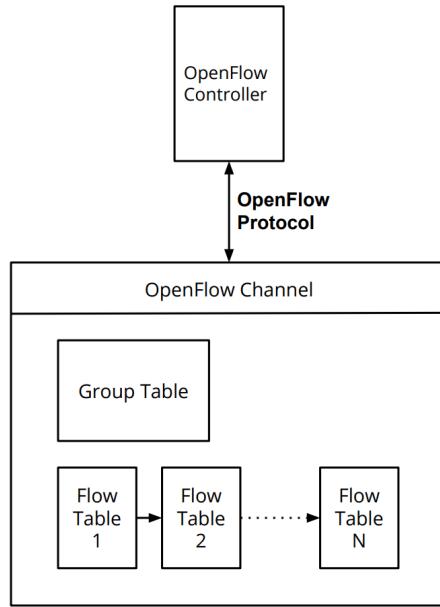


Figura 2.6: Arquitectura básica de agente OpenFlow [43]

En la literatura hay muchas propuestas de arquitecturas para los controladores, pero en vez de ir una por una viendo las diferencias vamos a presentar la arquitectura genérica que se puede encontrar en la gran mayoría de controladores SDN. Si nos fijamos en la Figura 2.7, podemos ver dos partes claramente diferenciadas, el núcleo del controlador y las interfaces del mismo [44]. Empezando por el *core*, se puede resumir que las funciones básicas del controlador están relacionadas principalmente con el descubrimiento de la topología y la gestión de los flujos de tráfico. El módulo de descubrimiento de la topología suele trabajar con el protocolo Link Layer Discovery Protocol (LLDP). La implementación puede variar entre controladores y aplicaciones de descubrimiento topológico, pero en términos generales, se transmite regularmente consultas utilizando mensajes `packet_out`, los cuales viajarán por la topología física y volverán al controlador en forma de mensajes `packet_in`, que permiten al controlador construir la topología de la red.

Una vez que se conoce la topología de red, el controlador puede empezar a poner en marcha distintos módulos de toma de decisiones para encontrar los caminos óptimos entre los nodos de la red. Con todos los caminos ya construidos, entran en juego otros módulos del controlador como son QoS y de seguridad, los cuales pueden optar por instalar una ruta sub-óptima para satisfacer los criterios de QoS o de seguridad. De forma adicional, el controlador puede

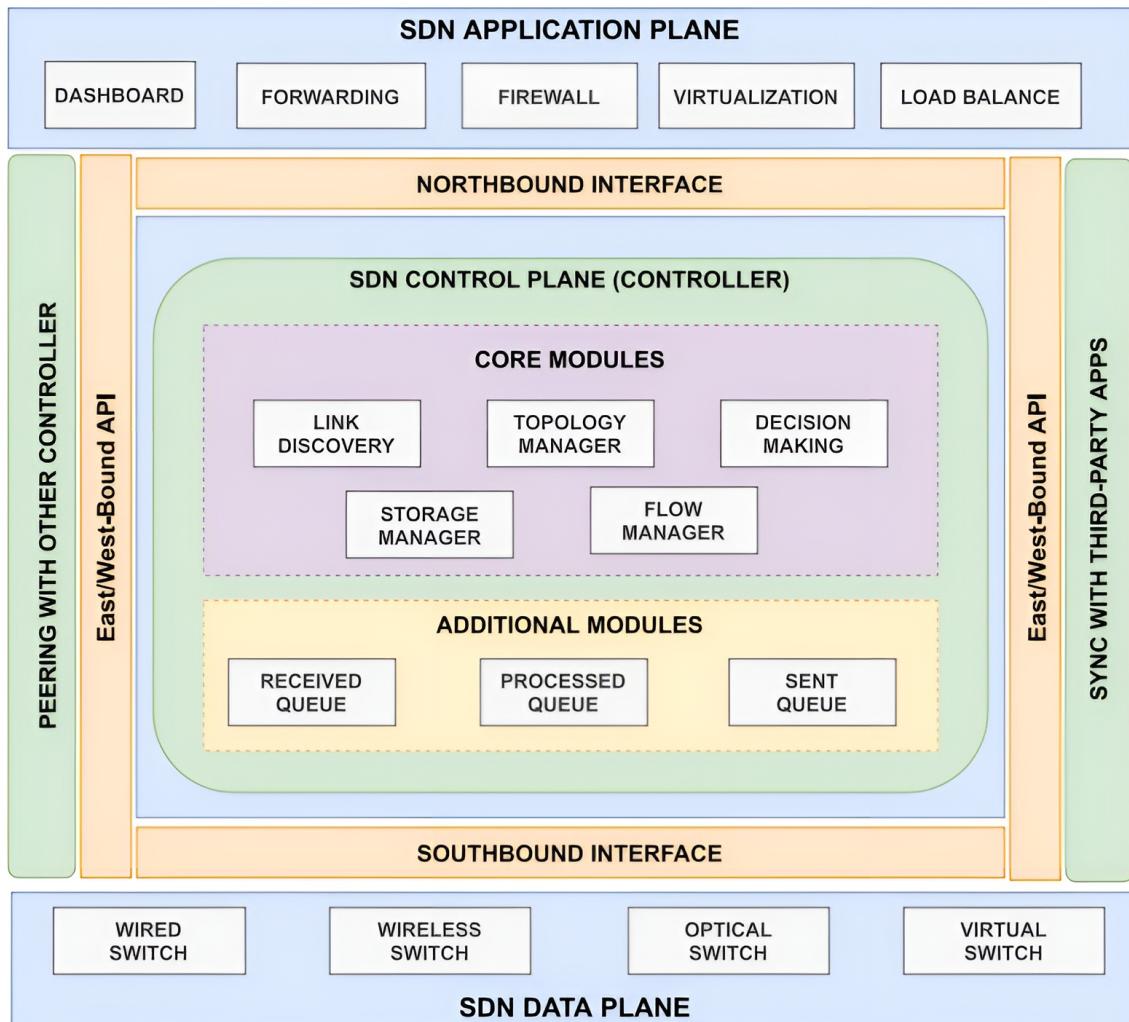


Figura 2.7: Arquitectura generica de controlador SDN [44]

tener un recopilador de estadísticas y un gestor de colas para recopilar información sobre el rendimiento de las diferentes colas de paquetes entrantes y salientes de los dispositivos de red que gestiona, y con ellos realimentar a los módulos de QoS. Por último, tenemos uno de los módulos más importantes del controlador, el gestor de flujos. El gestor de flujos, puede variar su implementación en función de los protocolos que se utilicen en la *Southbound Interface* (SBI), pero su misión es la misma, instalar reglas en los dispositivos de red que gestionan las directrices necesarias para gestionar los paquetes de un determinado flujo de una determinada manera.

Siguiendo con otra parte fundamental del controlador SDN genérico, son las interfaces. El controlador está rodeado de interfaces para interactuar con otras capas, superior e inferior.

rior, y otros controladores, este y oeste (E-WBIs). Empezando por la interfaz SBI, la cual es la encargada de interconectar dispositivos SDN con el controlador, define un conjunto de reglas, que variarán en función del protocolo que se utilice, las cuales permiten definir el procesamiento y las políticas de reenvío de los dispositivos SDN. El protocolo OpenFlow es unas de las SBI más utilizadas, y es un estándar de facto para la industria, con el cual podemos definir flujos y clasificar el tráfico de red basándose en un conjunto de reglas predefinidas. Pero también se pueden encontrar otras SBI, como por ejemplo, P4Runtime de facto el futuro para las SBIs, o podemos encontrar algunas más *legacy*, como por ejemplo, Netconf o incluso Simple Network Management Protocol (SNMP).

Si nos vamos de la API sur, al norte, encontraremos la conocida como *Northbound Interface* (NBI), la cual interconecta el controlador SDN con las aplicaciones de los desarrolladores o los servicios que definen el comportamiento intrínseco de la red. Los controladores admiten varias interfaces de programación de aplicaciones (API) northbound, pero la mayoría de ellas se basan en la API REST. Generalmente se quiere que la interfaz NBI sea una interfaz genérica para que limite a los desarrolladores. Para la comunicación entre controladores, se utilizan las interfaces conocidas como de este y oeste (E/WBI), las cuales no tienen una interfaz de comunicación estándar, por lo que, en función del controlador se tendrá una implementación u otra.

A continuación, se presentan dos de los controladores SDN más populares: Ryu y Open Network Operating System (ONOS), indicando algunas de sus virtudes y funcionamiento en particular, si bien existen otros como Floodlight, OpenDaylight, etc.

2.4.1. Ryu

Ryu⁴ es un controlador de red de código abierto diseñado específicamente para redes SDN. Se desarrolló en Python por el equipo de NTT (*Nippon Telegraph and Telephone*) y proporciona una plataforma flexible, sencilla y extensible para desarrollar aplicaciones de red basadas en SDN. Como se comentó anteriormente, la funcionalidad primordial que lleva a cabo es la de ser un intermediario entre los elementos de red, como por ejemplo switches SDN o nodos virtuales, y las aplicaciones o servicios que controlan la red a través de la NBI [45]. De esta forma, a través de la NBI, permite a los desarrolladores programar el comportamiento de la red de manera dinámica y centralizada, facilitando la implementación de políticas de red, la configuración de routing y la gestión de tráfico.

⁴Del Japonés, significa “flujo” y también “dragón”, ambos símbolos Kanji se leen igual como RYU, de ahí que el logo sea un dragón.

Ryu es altamente modular y proporciona una API bien definida que permite a los desarrolladores construir aplicaciones de red personalizadas hasta el más mínimo nivel. También es compatible con varios protocolos de comunicación utilizados en SDN, como OpenFlow (versiones 1.0, 1.2, 1.3, 1.4, 1.5), NETCONF y OF-config [45]. Las aplicaciones Ryu son entidades que implementan varias funcionalidades dentro de Ryu y se comunican entre sí a través de eventos. Los eventos sirven como mensajes intercambiados entre las aplicaciones Ryu. Estas aplicaciones se envían eventos asíncronos entre sí, creando un flujo de comunicación. Además de las aplicaciones Ryu, existen ciertas fuentes de eventos internas al propio Ryu. Un ejemplo de estas fuentes de eventos es el controlador OpenFlow. Cada aplicación Ryu posee una cola de recepción específicamente diseñada para eventos. La cola funciona según el principio *First In, First Out* (FIFO), lo que garantiza que se mantenga el orden de los eventos. Para procesar estos eventos, cada aplicación Ryu tiene un único hilo dedicado responsable de la gestión de eventos. Este subprocesso vacía continuamente la cola de recepción retirando los eventos de la cola e invocando al manejador de eventos apropiado en función del tipo de evento [46].

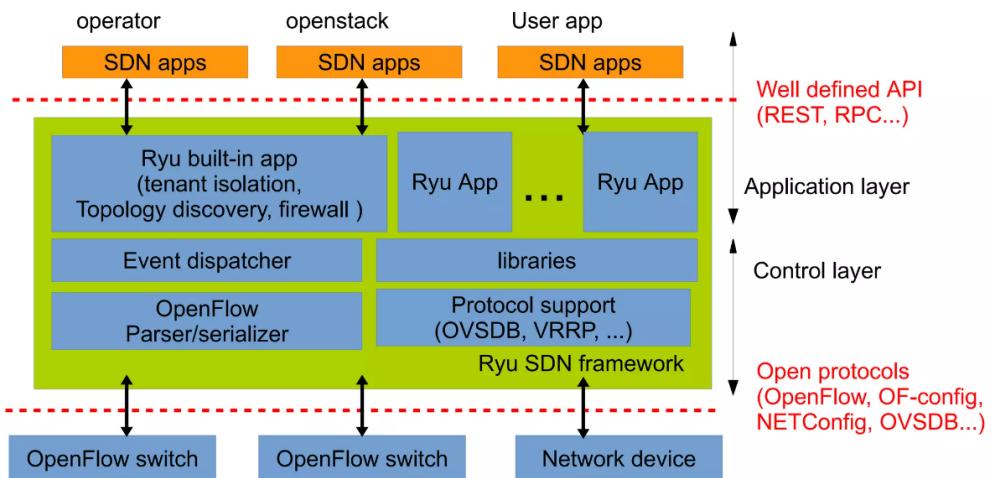


Figura 2.8: Arquitectura del controlador SDN RYU [47]

En la Figura 2.8, se puede apreciar como la arquitectura del controlador está dividida en dos grandes bloques de acuerdo se han estudiado los controladores SDN, la parte de *core* y la parte de interfaces. En la parte del núcleo se puede apreciar como se a programado todas las librerías y el manejador de eventos que se mencionaba anteriormente. Por otro lado también se puede apreciar las capas de adaptación a la SBI y a la NBI, la primera de ellas se adapta a través de una API REST a servicios de aplicaciones externas, mientras que la interfaz SBI implementa todos los protocolos de control SDN. Algunas de las características y funcionalidades de Ryu incluyen:

- Compatibilidad con múltiples protocolos SDN de la interfaz SBI.
- Capacidad para implementar políticas de red y reglas de enrutamiento de forma sencilla.
- Soporte para la recopilación y el análisis de datos de red en tiempo real.
- Funcionalidad de control de QoS.
- La más importante de todas, fácil desarrollo y portabilidad sencilla al estar escrita en Python. Esta última se puede ver también como una desventaja, dado el pobre rendimiento de un lenguaje interpretado.

Ryu es utilizado en una amplia gama de entornos, desde laboratorios de investigación hasta en las clases de forma educativa. Esta herramienta suele ser el punto de estrada para muchas personas que se inician en el SDN, sin embargo, al tener un pobre rendimiento, y la falta de abstracción con los protocolos que se usen en la interfaz *southbound*, hace que en entornos comerciales no se suela ver con tanta frecuencia [45].

2.4.2. ONOS

ONOS es un controlador de red de código abierto diseñado específicamente para redes SDN. Como su nombre indica, ONOS es un sistema operativo para redes que proporciona funcionalidades avanzadas de control y gestión de redes. ONOS está diseñado para ser escalable, confiable y de alto rendimiento, lo que lo hace adecuado para despliegues de red a gran escala. Es compatible con una amplia variedad de protocolos y tecnologías de red, como OpenFlow, NETCONF, BGP y P4 [48]. El controlador está respaldado por la ONF, la cual es una organización sin ánimo de lucro fundada en 2011 con el objetivo de promover y acelerar la adopción de la tecnología SDN y el enfoque de redes abiertas. Además de la ONF, numerosos proveedores de internet están impulsando el proyecto, así como, grandes empresas del sector TIC. Se pueden resumir las principales características y funcionalidades de ONOS en los siguientes puntos.

- Control centralizado de la red: ONOS proporciona un punto central lógico de control para la gestión de toda la red. Permite la configuración dinámica de la red, el enrutamiento y la asignación de recursos.
 - Escalabilidad: ONOS está diseñado para manejar redes de gran escala, distribuyendo la carga de trabajo entre múltiples nodos para lograr un rendimiento óptimo y una alta disponibilidad.
-

- Programabilidad: ONOS permite a los desarrolladores crear aplicaciones personalizadas utilizando una amplia gama de APIs y marcos de desarrollo. Esto facilita la implementación de políticas de red, la orquestación de servicios y la integración con otras aplicaciones y sistemas.
- Gestión de topología: ONOS proporciona una visión global de la topología de la red, permitiendo el descubrimiento de dispositivos, enlaces y rutas. Esto facilita la toma de decisiones basadas en el estado actual de la red.
- Gestión de flujos: ONOS admite la programación y gestión de flujos de red, lo que permite la implementación de políticas de enrutamiento y QoS de manera dinámica y centralizada.
- Segmentación de red: ONOS es compatible con la segmentación de red, lo que permite crear *network slices* para proporcionar aislamiento y asignación de recursos personalizada en una infraestructura compartida.

ONOS se concibe como un sistema operativo de red completo que va más allá de ser solo un controlador SDN. Ofrece una amplia gama de funcionalidades que incluyen herramientas como APIs que proporcionan abstracción para el desarrollo de aplicaciones SDN, así como APIs para la administración, supervisión y programación de dispositivos de red. Además, ONOS ofrece capacidades de virtualización, aislamiento, acceso seguro y abstracción de los recursos de red administrados por el sistema operativo. ONOS tiene la capacidad de multiplexar recursos tanto hardware como software entre las aplicaciones SDN, permitiendo una utilización eficiente de los recursos disponibles. Además, facilita la configuración de políticas de red basadas en las intenciones de las aplicaciones, lo que implica la aplicación de políticas de red diseñadas para satisfacer los requisitos específicos de las aplicaciones, así como el procesamiento de eventos de red [49].

Si nos fijamos en la Figura 2.9, podemos ver que este sistema operativo está diseñado para operar con dispositivos de red *whitebox*, con el objetivo de reducir los costes asociados con las soluciones propietarias. ONOS posee una arquitectura flexible que facilita la integración sencilla de nuevos dispositivos de hardware en el framework SDN. Solo se requiere que se añada un driver del equipo o target propietario. Además, puede funcionar como un sistema distribuido a través de múltiples servidores en modo cluster, lo que permite aprovechar los recursos de CPU y memoria de varios equipos simultáneamente. Esta capacidad también proporciona resiliencia ante posibles fallos de los servidores y permite realizar cambios en hardware y software sin interrumpir el tráfico de red [50].

ONOS cuenta con una comunidad activa de desarrolladores y usuarios que contribuyen al desarrollo y mejora del sistema. Además, ONOS es utilizado en diversos casos de uso, como

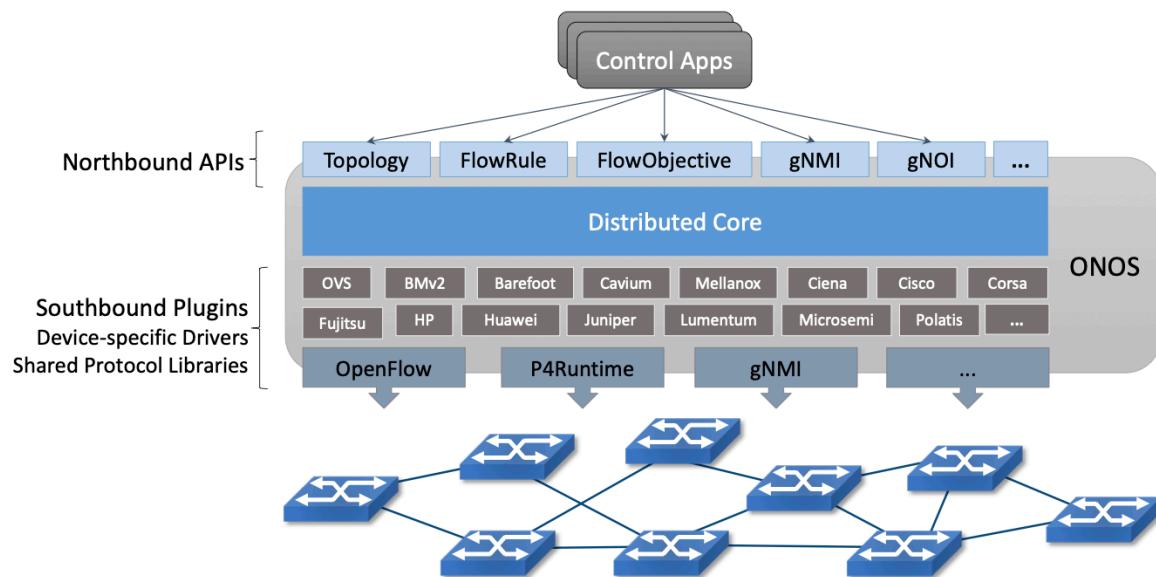


Figura 2.9: Arquitectura del controlador SDN onos [50]

redes de transporte, redes de centros de datos y redes de telecomunicaciones, entre otros muchos casos de uso. Si bien es cierto que tiene un muy buen rendimiento, no es controlador sencillo de manejar, y la curva de aprendizaje puede ser bastante pronunciada.

2.5. Software Switches SDN

En las redes definidas por software, los software switches desempeñan un papel fundamental al permitir la virtualización y la gestión centralizada de las redes. Si bien es cierto, que también existen switches SDN físicos, la flexibilidad y el creciente auge de los entornos virtualizados, hacen que sean los más utilizados hasta la fecha. Estos switches, a diferencia de los switches de hardware tradicionales, se implementan como software y se ejecutan en servidores convencionales. Un software switch en SDN, en adelante *softswitch*, es una entidad lógica que reside generalmente en una instancia virtual o en un servidor, y se comunica con el controlador SDN para recibir instrucciones sobre cómo procesar los paquetes de datos que fluyen a través de la red. Al estar basados en software, estos switches pueden ser escalados y desplegados de manera flexible según las necesidades y demandas de la red. La principal ventaja de los *softswitches* radica en su capacidad para adaptarse y responder de manera dinámica a las necesidades de la red. Pueden implementar diferentes funciones de red, como enrutamiento, conmutación, balanceo de carga y seguridad, a través de la instalación de reglas desde el controlador SDN.

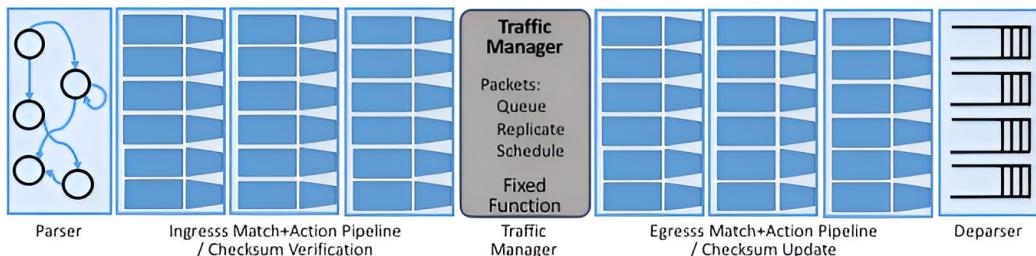


Figura 2.10: Arquitectura genérica de un *softswitch* SDN [51]

Según se puede apreciar en la Figura 2.10, la arquitectura genérica de un *softswitch* se puede resumir en los siguientes bloques. El primero de todos tiene que ser un parser, que vaya inspeccionando los paquetes entrantes a la pipeline de procesamiento del switch para identificar que tipo es. Una vez se ha identificado el paquete que se va a procesar, el siguiente bloque son las tablas de *match-action*, las cuales tienen una interfaz de comunicación con el controlador SDN para establecer criterios y campos de *match*, y en caso de haber un *match*, definir una serie de acciones para llevar a cabo. En función del *softswitch*, la verificación de *checksum*⁵ se puede llevar a cabo en el parser o en la etapa de las tablas de *match-action*. El siguiente bloque que nos podemos encontrar en un *softswitch* es el conocido como gestor de tráfico, que variará en función de la implementación, pero nos proveerá de gestión de colas, QoS, duplicado de paquetes, etc. La siguiente etapa ya es más opcional, que se suele denominar como *egress match-action*, la cual se puede utilizar para definir algún tipo de lógica a la salida de los switches, aunque en la realidad se suele utilizar para actualizar los campos de TTL y recalcular el *checksum* dado que el paquete se habrá visto modificado. El último bloque es el deparser, el cual se encarga de ensamblar de nuevo el paquete y prepararlo para sacarlo por el puerto de salida del switch.

2.5.1. OvS

Open vSwitch (OvS), es uno de los *softswitches* de referencia en el mundo de las redes, ampliamente utilizado tanto en la industria como en la academia. El switch está ofrecido como un proyecto de código abierto (en inglés *open source*), y respaldado por la Linux Foundation. Es uno de los software switches más utilizados y ampliamente adoptados en entornos SDN debido a su flexibilidad y funcionalidades avanzadas, aunque si tiene que destacar por algo, es por su gran rendimiento al trabajar a nivel de Kernel siendo idóneo para entornos de producción [52]. OvS actúa como un switch virtual, proporcionando capacidades de commutación y reenvío para máquinas virtuales y contenedores en entornos de virtualización. Puede ejecutarse en hipervisores populares, como KVM (Kernel-based Virtual Machine), Xen y VMware, y también puede ser implementado como un switch inde-

⁵Suma de comprobación, campo para comprobar la integridad del paquete de datos

pendiente en instancias o servidores en modo standalone. Las características clave de Open vSwitch incluyen:

- Agente SDN: el switch ofrece una interfaz que permite que sea controlado mediante un controlador SDN, como OpenDaylight o ONOS. Esto permite una gestión centralizada y un control más granular de la red, así como la implementación de políticas de red definidas por software.
- Funcionalidades avanzadas de alto rendimiento: OvS ofrece una amplia gama de funcionalidades, como fast-forwarding al trabajar con frameworks como eXpress Data Path (XDP), DPDK o extended Berkeley Packet Filter (eBPF). Estas características permiten una gestión eficiente de la red, y entorno de altas prestaciones ofreciendo un alto rendimiento perfecto para despliegues de producción.
- Integración con tecnologías de virtualización: OvS se integra estrechamente con tecnologías de virtualización como OpenStack y Docker. Puede proporcionar conectividad de red entre máquinas virtuales, contenedores y hosts físicos, facilitando la migración y la gestión de recursos en entornos virtualizados.
- Extensibilidad y soporte para estándares: El OvS es altamente extensible y se puede ampliar mediante la integración de módulos y complementos personalizados. Por ejemplo, para la integración del lenguaje de P4 se está haciendo de forma paulatina mediante módulos. Además, cumple con los estándares de la industria, como el protocolo OpenFlow, para garantizar la interoperabilidad con otros componentes SDN.

Si nos fijamos en la Figura 2.11, podemos apreciar los componentes principales de la arquitectura del OvS. Como se puede apreciar en la figura hay dos partes claramente diferenciadas en el *softswitch*, una de espacio de usuario y otra parte de espacio de Kernel. Esta última es la que proveerá al OvS de un alto rendimiento en comparación con otros software switches. Los componentes principales del OvS se pueden resumir en los siguientes puntos [52].

- **ovs-vswitchd**, es un *daemon* que corre en espacio de usuario que implementa el switch, este proceso corre de forma conjunta con un módulo que corre en espacio de kernel, los cuales se comunican por netlink⁶ para establecer la política de gestión de flujos.
- **ovsdb-server**, es un servidor de base de datos ligera, a la cual el proceso de espacio de usuario **ovs-vswitchd** le hace queries para obtener su configuración.

⁶<https://man7.org/linux/man-pages/man7/netlink.7.html>

- **ovs-dpctl**, es una herramienta de gestión del módulo del kernel que implementa el datapath. El nombre de la herramienta hereda del nombre puesto a la herramienta original **dpctl**, la cual fue propuesta por Stanford en 2009⁷, y es un acrónimo de *datapath-control*.
- **ovs-vsctl**, es una utilidad para gestionar la configuración del proceso de espacio de usuario **ovs-vswitchd**.
- **ovs-appctl**, es una herramienta para mandar comandos a los *daemons* OvS.
- **ovs-ofctl**, es una herramienta con la cual podemos mandar comandos y controlar vía OpenFlow el funcionamiento de switch.
- **ovs-testcontroller**, es un controlador experimental que implementan para testear la recepción y manejo de mensajes OpenFlow.

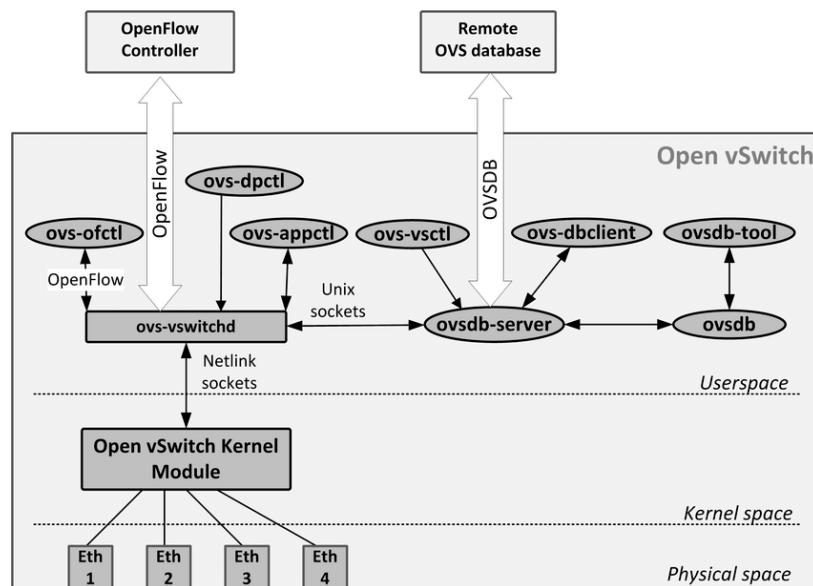


Figura 2.11: Arquitectura del OvS [53]

En resumen, el OvS es un software switch SDN ampliamente utilizado que proporciona capacidades avanzadas de conmutación y enrutamiento para entornos de virtualización. Su flexibilidad, funcionalidades avanzadas, rendimiento y compatibilidad con estándares lo convierten en una opción popular para implementaciones SDN en diversos entornos, desde centros de datos hasta infraestructuras de proveedores de servicios.

⁷<https://github.com/mininet/openflow/blob/master/utilities/dpctl.c>

2.5.2. BOFUSS

Aunque OvS domina ampliamente el mundo de los software switches SDN, BOFUSS se presenta como una alternativa con otras ventajas y desventajas, que puede ser más interesante para ciertos casos de uso.. Este switch nació como una primera implementación por el 2008 en la universidad de Stanford, acuñado como *The Stanford Reference OpenFlow Switch*. Esta implementación era un mínimo producto viable para demostrar y ayudar en el proceso de estandarización del protocolo *OpenFlow 1.0*. Dicho mínimo producto viable, fue retomado por los laboratorios de Ericsson, *Ericsson Research TrafficLab*, para desarrollar la versión *OpenFlow 1.1* [54].

Este último desarrollo fue retomado por el investigador Eder Leão Fernandes, desde el CPqD de Brasil, fue parte de su trabajo fin de máster y tesis doctoral, donde completo lo que hoy conocemos como el BOFUSS, el cual da soporte para la versión *OpenFlow 1.3*. Por último, se verá más adelante que este switch fue tomado por Boby Nicusor Constantin y modificado para hacer una implementación de control In-band. En la siguiente figura, ver Figura 2.12, se puede apreciar un pequeño resumen de la historia del software switch BOFUSS.

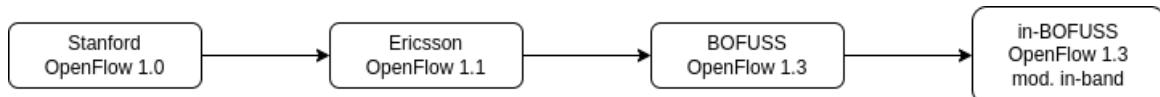


Figura 2.12: Evolución del BOFUSS

La arquitectura del BOFUSS se puede apreciar en la Figura 2.13. Según se ha podido averiguar, por el propio creador, esta arquitectura si bien es cierto que no trata de cumplir la especificación de OpenFlow al 100%, es la implementación más cercana a la especificación oficial de *OpenFlow 1.3*. Como se puede ver en la figura siguiente (Figura 2.13) , el software switch se compone de dos bloques fundamentales.

- Plano de datos, *Datapath*, en la herramienta al plano de datos lo podemos encontrar como `udatapath/ofdatapath`.
- Plano de control, *Control plane*, en la herramienta al plano de control lo podemos encontrar como `secchan/ofprotocol`.

El primero de ellos, el `udatapath/ofdatapath` se caracteriza por ser el bloque funcional de gestionar el procesamiento de los paquetes datos, y en ocasiones de control (en función del paradigma de control). Dentro de este bloque funcional se pueden encontrar elementos internos como por ejemplo, los puertos, conocidos como `Port`, los `Flow`, `Meter`, `Group`,

Table y el Packet Parser. El bloque del agente de control, es el encargado de gestionar la información de control entre el controlador y el dispositivo. Los mensajes de OpenFlow viajarán desde el plano de secure channel, a la librería de **oflib**, y de ahí al datapath para instanciar en las tablas de flujos correspondientes. Más adelante se explica en detalle cada bloque de la arquitectura.

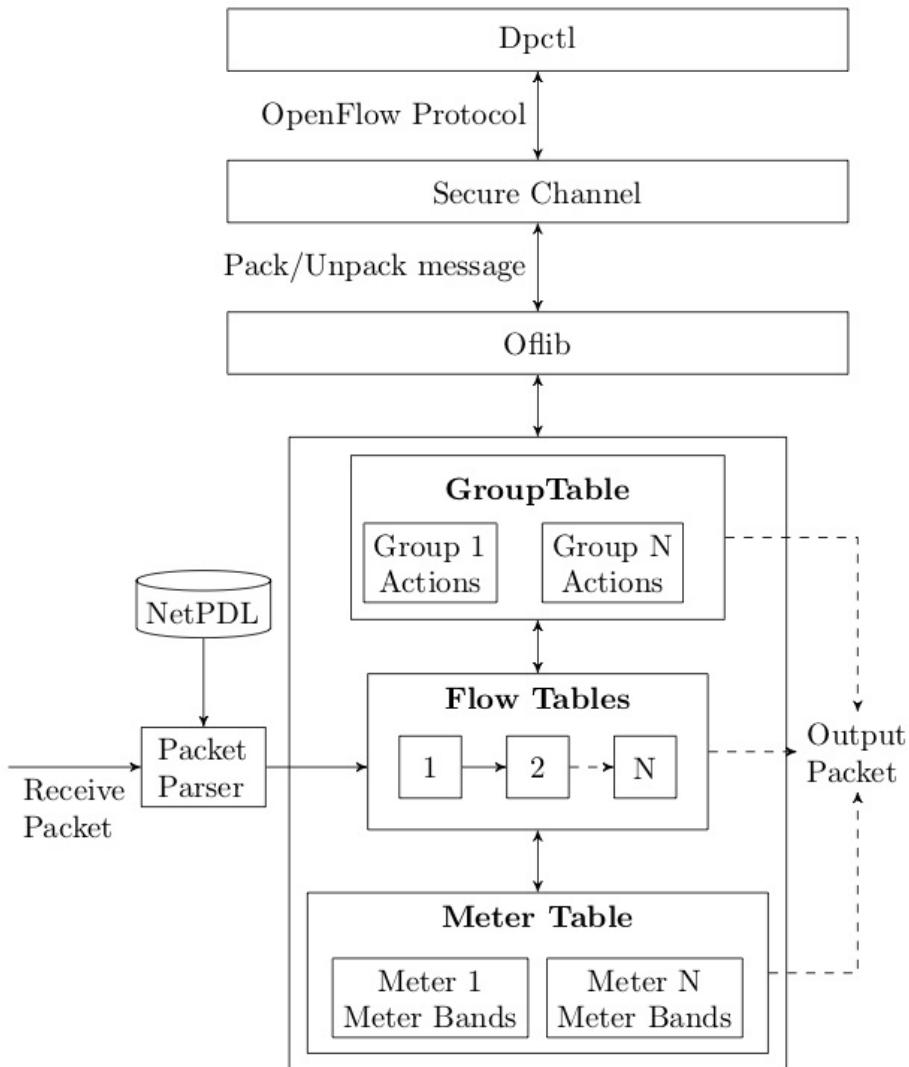


Figura 2.13: Arquitectura del BOFUSS [43]

2.5.2.1. Ports

Los puertos OpenFlow desempeñan un papel fundamental como puntos de entrada y salida para los paquetes de datos en un entorno OpenFlow. Cuando se ejecuta un software

switch en una máquina, puede utilizar interfaces físicas o virtuales como sus puertos (interfaces físicas o virtuales como Virtual Ethernet Device (Veth) o también radio taps emulados). Los puertos físicos permiten el control de interfaces Ethernet o WiFi, lo que facilita la gestión de tanto topologías de red realistas como emuladas. Aunque la velocidad del software switch puede ser limitada dado que trabaja en espacio de usuario, la posibilidad de crear un entorno de pruebas mejora la experiencia de los usuarios que desarrollan y evalúan aplicaciones OpenFlow. Se podría pensar que los puertos del switch se limitan simplemente a enviar y recibir paquetes de red, pero en verdad, también tienen una serie de responsabilidades relacionadas con la gestión del protocolo OpenFlow. Estas responsabilidades se pueden resumir en los siguientes puntos.

- OpenFlow permite cierto nivel de control sobre el comportamiento que tiene que tener un puerto en particular. Si se recibe un mensaje de modificación de puerto, este tiene que permitir configurar el estado del puerto. Los puertos pueden configurarse para que descarten todos los paquetes recibidos, prohíban la generación de mensajes de tipo openflow **Packet-In** a partir de los paquetes que llegan, además de marcar el estado del puerto como fuera de servicio. El agente que gestione el puerto deben gestionar estos mensajes de configuración que le lleguen, y cambiar el comportamiento del puerto según la configuración recibida.
- Los puertos OpenFlow tienen que llevar un monitoreo del estado de la interfaz física o emulada que gestionan. Si bien es cierto que el controlador no puede actuar sobre el estado real de la interfaz, el switch tiene que informar sobre los cambios de estado del enlace.
- Generalmente cuando se lleva a cabo un **Packet-In** porque hay un *miss*, solo se manda al controlador las cabeceras del mensaje a consultar junto al propio mensaje del **Packet-In**. Los puertos, durante dicha consulta tendrán que gestionar los buffers que almacenarán los paquetes a consultar para ser procesados más tarde.
- El controlador a través del agente de control, también puede consultar sobre la descripción de un puerto. Por tanto, el software switch tendrá que recolectar la información que considere oportuna como por ejemplo la velocidad actual y máxima de las interfaces reales, almacenarla para enviarla posteriormente cuando el controlador se lo requiera.
- Las colas según se ha podido consultar no son parte de la definición estándar de OpenFlow. Sin embargo, OpenFlow puede configurar colas asociadas a unos puertos dados. Los puertos por tanto tendrán la responsabilidad de llevar a cabo la asociación de configuración de cola y asociación de cola con un puerto además de actualizar los contadores de paquetes de puerto y cola asociada.

2.5.2.2. Packet Parser

Antes que el paquete en cuestión llegue a la pipeline de procesamiento del software switch, este debe ser procesado para adaptarlo a las estructuras de datos que se manejan en el switch. Para ello, el cómo se tiene que parsear los paquetes se definieron en el estándar de OpenFlow 1.1. Esto es importante, ya que debe haber consistencia en como los paquetes deben ser pasados, pero esto a su vez supuso una limitación para nuevos diseños de switches, y supone modificaciones cada vez que se añade un nuevo protocolo. Por ello, más adelante con especificaciones posteriores del protocolo esta limitación se vio eliminada. El parser que se tiene en el BOFUSS hace uso de **NetBee** como disector y parseador de paquetes. Una vez que se han identificado los campos de protocolo que contiene el paquete, se crea una estructura de matching con la cual se pasará a la pipeline de procesamiento del switch. A continuación, en la Figura 2.14 se puede ver la estructura básica del parser.

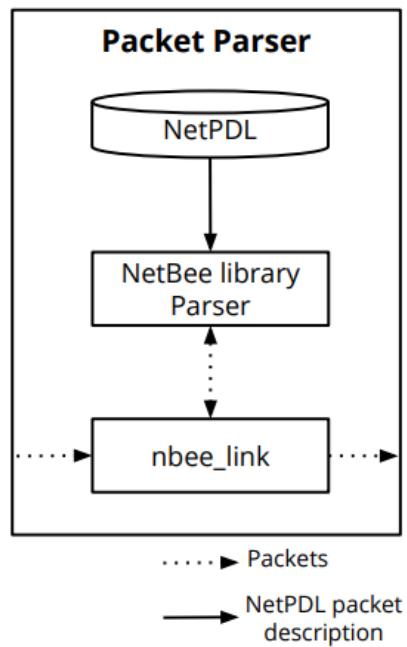


Figura 2.14: Parseador de paquetes del BOFUSS [43]

Este paso de procesado del paquete anteriormente descrito se puede dar en dos ocasiones. A continuación se indican.

- Que el paquete de red entre por uno de los puertos gestionados por el software switch.
- Que un paquete que ya ha sido modificado y redirigido por la pipeline de procesamiento, o enviado a una nueva tabla de adelante con la instrucción de Go To Table.

Esto se hace esta forma, dado que una revalidación del paquete es necesaria, y el parsear también se encarga de comprobar la validez del campo TTL. Además de añadir información de metadatos al mismo. Cada vez que se quiera dar soporte a nuevos protocolos, se tendrá que modificar el parsing del switch. Para ello, se tiene que llevar modificaciones a cabo en el fichero `*.xml` en lenguaje NetPDL. NetPDL es un lenguaje de descripción que describe cómo Netbee debe analizar los protocolos, la implementación actual del BOFUSS tiene su propio fichero ya creado, se puede consultar aquí⁸.

2.5.2.3. Flow Tables

Las tablas de flujos, del inglés *Flow Tables*, son el core de procesamiento del software switch. Las flow tables siempre son el siguiente paso después del procesamiento y podrían considerarse el corazón del entorno openflow dado que son el primer componente en la pipeline de procesamiento del switch. Aunque el uso de múltiples tablas de flujos es opcional, la especificación indica que se utilice al menos una tabla, a la hora de la implementación se considera más que recomendado, dado que, es inviable el hecho de gestionar el escalado de una aplicación con únicamente solo una tabla de flujo.

En pocas palabras podríamos definir una *Flow Table* como una lista de flujos, donde cada flujo se compone de unos campos de matching y de unas instrucciones asociadas en caso de que haya match. Unas instrucciones que como se indican, tienen que estar definidas previamente en la especificación de OpenFlow. Una vez el paquete se ha validado y se ha parseado en una estructura de matching, se comprueba con una entrada de un flujo con el campo de matching, en caso de que coincida con dicho flujo, las instrucciones asociadas se ejecutan. A continuación, se indican algunos de los aspectos que tienen que cumplir las tablas de flujo.

- La implementación de una tabla de *miss* es obligatoria⁹, ya que el switch tiene que hacer algo con los paquetes los cuales no coinciden con ninguna *flow entry*. Por el contrario, si no se implementa ninguna tabla de *miss*, se tiene que establecer una action por defecto. En el caso del switch, se ha establecido que se tiren los paquetes.
- Otro aspecto a considerar, el cual, tiene que ser gestionado por las *Flow table* es la gestión de los paquetes Flow-Mod. Estos mensajes son generados desde el controlador y gestionados por el agente de control del switch para creación o eliminación de entradas de flujo en alguna *Flow table*.

⁸<https://github.com/NETSERV-UAH/in-BOFUSS/blob/main/customnetpd1.xml>

⁹En OpenFlow 1.3, pero no en OpenFlow 1.0

- El switch debe permitir al controlador de reconfigurar sus prestaciones. Es decir, las propiedades de las tablas deben ser conocidas por el controlador, y las tablas deben en todo momento responder a los mensajes de consulta de las características.
- Por cada paquete del plano de datos que les lleguen, deben llevar a cabo un *look up*, es decir, consultar si el paquete entrante de datos coincide con algún campo de match de algún flujo de la tabla. En caso de que exista algún match, se ejecutará la instrucción asociada.
- Otro aspecto a llevar a cabo por el switch, es llevar el recuento de las estadísticas de las entradas activas, las consultas realizadas, y paquetes que han hecho match.

En cuanto a aspectos de implementación, podemos destacar, las reglas se indexan en las tablas de flujos en orden de prioridad, si tienen la misma prioridad, se indexarán en orden de llegada. En cuanto a la complejidad del tiempo de consulta, es lineal es decir $O(n)$, donde n es el número de flujos. Esto no es muy eficiente, debido a que crece de forma lineal según el número de flujos aumenta. Aunque según ha indicado el autor del BOFUSS, es suficiente para llevar pruebas de concepto, sin embargo, no es adecuado para un entorno industrial de producción. Otro detalle de implementación, que tenemos que tener en cuenta es el número de entradas de flujos por tabla de flujos, actualmente está definido a 64 por una macro. Si se quisiera cambiar este parámetro solo habría que cambiar la macro y recomilar el proyecto. Por último, mencionar que las tablas de flujos tienen una lista de `idle` y `hard timeout`, que comprueban cada `100ms`, para ver si alguna de sus entradas de flujos han expirado.

2.5.2.4. Group Table

Las tablas de grupos, del inglés *Group Tables*, se utilizan para agregar flow entries que tienen una política de acción similar. Cada grupo tiene un identificador único y contiene una lista de *buckets* que definen las acciones a tomar en caso de que el paquete coincida con ese grupo en particular. Cada bucket en una *Group Tables* define una serie de acciones a ejecutar para un paquete que coincide con ese grupo. Estas acciones pueden incluir enrutamiento, reenvío a puertos específicos, encapsulación, copia de paquetes, descarte, entre otras. Un bucket puede tener múltiples acciones y también puede contener una acción especial para indicar que el paquete debe ser procesado por otros grupos en cascada.

La utilización de *Group Tables* permite un procesamiento más eficiente de los flujos de paquetes, ya que se pueden realizar acciones comunes de manera conjunta en lugar de procesar cada flujo de paquetes individualmente. Esto reduce la carga de procesamiento en los switches OpenFlow y facilita la implementación de políticas de red más complejas y flexibles. A continuación, en la Figura 2.15 se puede apreciar la estructura de las *Group Tables*.

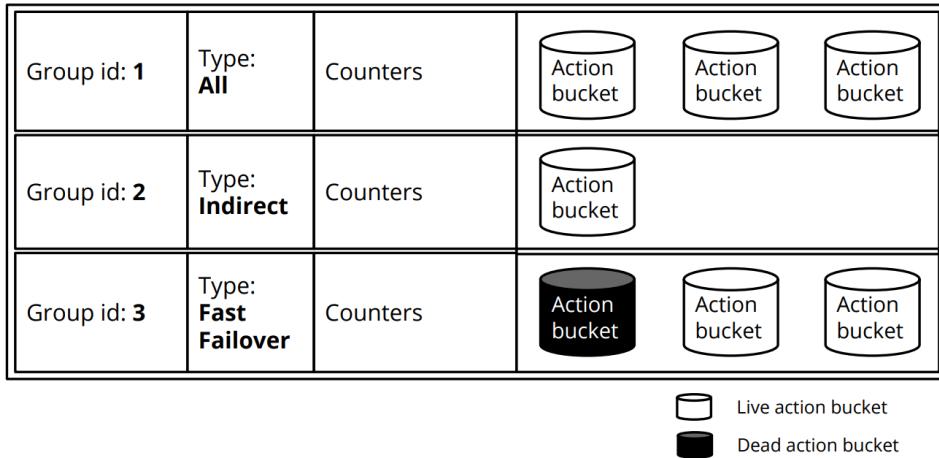


Figura 2.15: Estructura de las *Group tables* del BOFUSS [43]

2.5.2.5. Meter Table

Las tablas de medidores, del inglés *Meter Table*, es el core del QoS del software switch. Para cada flujo se tienen unas meter asociadas en la propia entrada en la tabla de flujo. Dichas meters, tienen una entrada en la meter table. Cada entrada se compone de una ID, un contador, y unas meter bands. Estas últimas, las meters-bands son las encargadas de llevar a cabo las operaciones de QoS. Cada *Meter Table* debe tener un tipo, un ratio, el cual será el límite que tiene que superarse para aplicar la action definida por el tipo de la meter. A continuación, se ver la Figura 2.16 que ilustra la arquitectura de una Meter table.

Meter id: 1	Counters	Meter Band			
		Type: DSCP Remark	Rate: 100 kbps	Precedence Level: 1	Counters
Meter id: 2	Counters	Meter Band			
		Type: Drop	Rate: 100 kbps	Counters	

Figura 2.16: Estructura de las *Meter tables* del BOFUSS [43]

Entre las responsabilidades de las meter tables podemos encontrarnos las siguientes:

- Creación, destrucción y modificación de las entradas de las meters
- Medir el ratio de aquellos paquetes que han matchado en una flow entry, y apuntan a una meter table.

- Mantener actualizado los contadores de las estadísticas de los paquetes procesados por cada entrada en la meter table.

2.5.2.6. oflib

Los mensajes de OpenFlow están definidos de una manera en particular para ser transmitidos por la red. Los mensajes tienen que estar en modo 8-byte alineados, por lo que habrá alguna ocasión donde se tenga que añadir padding para que se cumpla esta regla. Otro requisito es que el mensaje tiene que estar en Network byte order, es decir, Big Endian. Los mensajes OpenFlow que se mandan por la red tienen que estar en el formato indicado anteriormente, es decir, el byte de mayor peso de una palabra tiene que estar almacenado en la posición más pequeña (la dirección más baja).

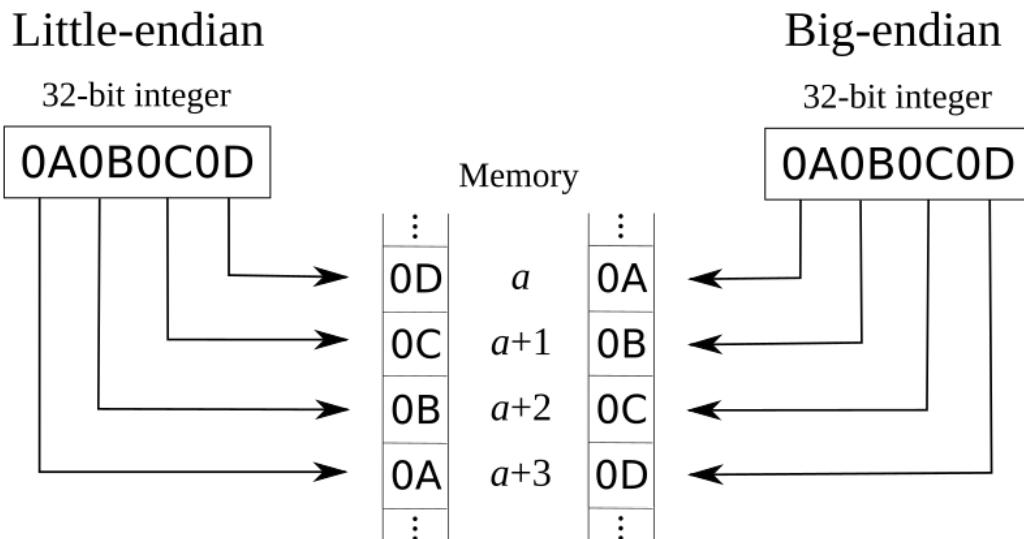


Figura 2.17: Proceso de Marshaling y Unmarshaling

Las arquitecturas de cada máquina pueden variar, y el formato de datos con en el que trabajan también. Por ejemplo para ARM e Intel el formato con el cual trabajan ambas arquitecturas es Little Endian byte order. Por ello, en aras de manejar y codificar mensajes openflow se requiere una conversión big-endian a little-endian. Debido a cuál, se necesita una capa de abstracción de la arquitectura donde se vaya a correr dicho software switch. Por ello, aunque el estándar de OpenFlow no lo indique, se ha añadido esta librería denominada como **oflib**. La función principal de esta librería es las operaciones de *marshaling* y *unmarshaling* de los mensajes OpenFlow. Para que la transmisión de información de mensajes OpenFlow a la red se lleve a cabo de forma completamente autónoma. Las responsabilidades de esta

librería por tanto son las siguientes.

- Cada mensaje openflow debe tener una función para empaquetarlo y desempaquetarlo. De aquí en adelante, y en el repositorio, nos referimos a hacer un *pack* es coger una estructura de datos openflow y prepararla para ser transmitida por la red. Cuando realizamos una operación de *unpack*, cogemos información que viene por la red, y la convertimos a estructuras que entienda la arquitectura sobre la cual está corriendo nuestro software switch.
- Otra responsabilidad que tiene la librería es señalar con errores o warnings en caso de los mensajes OpenFlow estén mal codificados.

2.5.2.7. Communication Channel

El software switch se comunica con el controlador SDN a través de este agente de control que actúa de proxy entre el datapath y el controlador SDN. Este agente se encarga de gestionar las conexiones con el controlador SDN, aunque, si bien es cierto que este agente no está pactado en el estándar de OpenFlow, esto da libertad a las diferentes implementaciones para configurar la conexión hacia al controlador como quieran. Por ejemplo, si se quiere que la conexión sea segura extremo a extremo, se tendría que utilizar TLS encima de TCP. Esta libertad de diseño en el canal de comunicación con el controlador, habré un *gap* que se puede utilizar para desarrollar implementaciones dispares que ofrezcan distintas bondades y funcionalidades. Entre las responsabilidades de esta capa podemos mencionar las siguientes.

- El agente de control se tiene que encargar de abrir una conexión TCP entre el switch y el controlador.
- El establecimiento de la conexión es responsabilidad del agente de control. Después del inicio de la conexión, el switch negocia la versión OpenFlow a utilizar entre el switch y el controlador. Este proceso se conoce como handshake.
- El agente de control debe soportar más de un controlador.
- Además, el agente de control debe soportar más de una conexión con el mismo controlador.

2.6. Tecnologías Linux

Esta sección recopilará todos los conceptos y herramientas relacionadas con la parte de red en Linux, que son fundamentales para el desarrollo, análisis y validación de este proyecto.

2.6.1. Interfaz virtual - tun/tap

En el mundo de las redes siempre se habla de las interfaces tun/tap de forma indistinta cuando van a utilizarse, sin embargo, cada una tiene su cometido. Como se ha indicado, en networking, las interfaces TUN/TAP son interfaces virtuales que se crean y se gestionan en espacio de kernel. Mencionar que como estas interfaces son virtuales y se gestionan directamente vía software, no como las interfaces reales que se gestionan con unos drivers diferentes, cada interfaz con su driver específico de la interfaz. Los drivers de las interfaces TUN/TAP se crearon en los 2000 como una unión de los avances de los drivers desarrollados en las comunidades de Solaris, Linux, BSD. Actualmente los drivers solo tienen mantenimiento por los kernels de Linux y FreeBSD. Ambos tipos de interfaces se utilizan para tunelado, pero no pueden ser utilizadas a la vez dado que trabajan en niveles distintos. Las TUN, de *network TUNnel*, emula la capa de red y puede llegar hacer reenvío de los paquetes. En cambio las interfaces TAP, trabajan en capa 2 solo en capa dos, y emulan un equipo que trabaja en dicha capa, como por ejemplo un switch [55]. Por tanto, hay que dejar claro lo que se puede llegar a realizar con cada interfaz (Ver Figura 2.18).

- TUN se puede llegar a utilizar para routing.
- TAP se puede llegar a utilizar para crear un bridge.

Generalmente, cuando los paquetes son enviados por el sistema operativo a través de una interfaz TUN/TAP, serán recibidos por algún programa de espacio de usuario, el cual, está enganchado directamente en la interfaz. Cualquier programa de espacio de usuario podrá pasar paquetes por las interfaces, y las interfaces virtuales se lo pasarán al *stack* de red por defecto, emulando la recepción de los paquetes injectados desde espacio de usuario.

Para la creación de estas interfaces lo podemos hacer por ioctl o podemos hacerlo más fácil a través del binario tunctl o en su defecto con el comando de **tuntap** del set de herramientas iproute2. A continuación, en el bloque de Código 2.1 se indican todos los comandos necesarios para trabajar con las interfaces TUN/TAP.

Código 2.1: Manejo de interfaces TUN - TAP

```

1 # En caso de querer utilizar tunctl hay que instalar el binario
2 sudo apt install -y uml-utilities
3
4 # Para crear una interfaz podemos hacer lo siguiente
5 tunctl -t {nombre_tun}
6
7 # Para eliminarla

```

TUN and TAP in the network stack

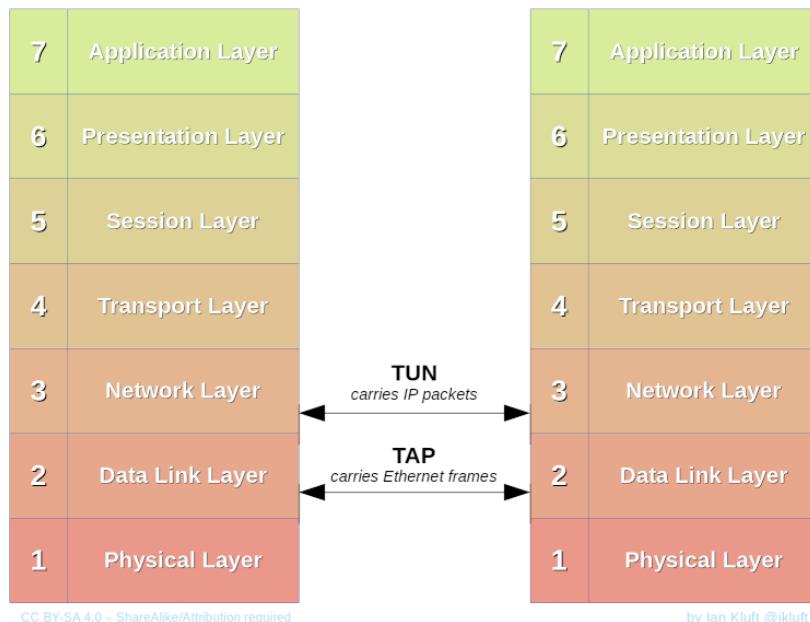


Figura 2.18: Diagrama de funcionamiento de las interfaces virtuales TUN/TAP [56]

```

8  tunctl -d {nombre_tun}
9
10 # Para crear interfaces de tipo TAP hay que hacer lo siguiente
11 tunctl -p -t {nombre_tun}
12
13 # El comando análogo con iproute2 sería el siguiente
14 ip tuntap add dev {nombre_tun} mode {tun|tap}

```

En caso de que queramos comprobar que las interfaces se han creado correctamente siempre se puede hacer uso de la herramienta `ethtool`. A continuación, en la Figura 2.19, se puede ver como en el campo `bus-info` nos indica que tipo de interfaz es.

2.6.2. Interfaz virtual - veth

Las interfaces virtuales Veth son interfaces Ethernet creadas como un par de interfaces interconectadas entre sí. El modelo funcional es sencillo: los paquetes enviados desde una interfaz son recibidos por la otra de forma directa, similar al funcionamiento de las *pipes* (mecanismo de comunicación interprocesos en Linux). Una característica interesante de estas interfaces es que su gestión está asociada, lo que significa que si se habilita una extremo de la Veth, el otro extremo también se habilitará, y si se deshabilita o se elimina un extremo de un par de Veth, el otro extremo también se verá afectado [57].

```

arppath@arppath-david:~$ sudo ip tuntap add dev tap9 mode tap
arppath@arppath-david:~$ ethtool -i tap9
driver: tun
version: 1.6
firmware-version:
expansion-rom-version:
bus-info: tap
supports-statistics: no
supports-test: no
supports-eeprom-access: no
supports-register-dump: no
supports-priv-flags: no
arppath@arppath-david:~$ sudo ip tuntap add dev tun9 mode tun
arppath@arppath-david:~$ ethtool -i tun9
driver: tun
version: 1.6
firmware-version:
expansion-rom-version:
bus-info: tun
supports-statistics: no
supports-test: no
supports-eeprom-access: no
supports-register-dump: no
supports-priv-flags: no
arppath@arppath-david:~$ 

```

Figura 2.19: Comprobación con ethtool de tipo de interfaz virtual.

Es muy común utilizar las Veth para interconectar *Network Namespaces*. Al saber que estas interfaces estarán conectadas de forma directa, se puede utilizar este enlace como una pasarela entre dos *Network Namespaces*. De esta manera, se pueden interconectar dos *stacks* independientes de red. Más adelante se verá que alguno de los emuladores (Mininet) más utilizados para comprobar despliegues de red SDN hace uso de estas interfaces. Y también es posible ir más allá, aunque no se busca tampoco entrar en el mundo de los contenedores dado que el TFM no va a utilizarlos, pero cabe destacar que estas interfaces también son utilizadas para la interconexión y contenedores de Docker [58]. La creación y eliminación de este tipo de interfaces se puede observar en el bloque de Código 2.2. Se recuerda que se necesitan permisos de root para realizar estas operaciones.

Código 2.2: Uso de las interfaces Veths

```

1 # Con este comando se crean un par interfaces veth
2 ip link add {veth1_name} type veth peer name {veth2_name}
3
4 # Solo hace falta indicar uno de los dos nombres del par de Veths
5 ip link delete {veth_name}

```

2.6.3. Herramienta TC

En Linux, Traffic Control (TC) es un subsistema del kernel, que tiene una interfaz en espacio de usuario que se utiliza para controlar y gestionar el tráfico de red. Proporciona una serie de herramientas y mecanismos para aplicar políticas de QoS, limitar el ancho de banda, establecer prioridades de tráfico, realizar filtrado y dar forma al tráfico de red [59]. Se ha añadido esta sección a la memoria dado que la mayoría de emuladores de red en Linux suelen hacer uso de esta herramienta para modelar los enlaces de las topologías emuladas. El TC se utiliza principalmente para las siguientes tareas.

- Control de ancho de banda: Nos permite limitar la cantidad de ancho de banda que un flujo de datos específico puede utilizar. Esto es útil para evitar que un flujo de tráfico acapare todo el ancho de banda y afecte el rendimiento de otros flujos.
- Priorización de tráfico: Permite asignar prioridades diferentes a los diferentes flujos de datos. Esto asegura que ciertos flujos de tráfico, como VoIP o similares al ser más sensibles, tengan prioridad sobre otros, como la descarga de archivos donde la latencia y el jitter no son un problema, pero si la integridad.
- Modelado y conformación de tráfico: Permite dar forma al tráfico de red mediante la definición de límites en el ritmo de transmisión. Esto es útil para evitar congestiones en la red y garantizar un flujo de tráfico constante y uniforme.
- Marcado y filtrado de paquetes: Permite clasificar y filtrar paquetes de acuerdo a diferentes criterios, como direcciones IP, puertos, protocolos, etc. Esto permite aplicar políticas específicas a ciertos flujos de datos o bloquear ciertos tipos de tráfico no deseado. Esta funcionalidad es útil, pero está duplicada con otros submódulos del kernel de Linux, como por ejemplo las `ip-tables`.

El procesamiento del tráfico para conseguir llevar a cabo las tareas anteriormente mencionadas, se emplean tres tipos de objetos: **qdiscs**, **classes** y **filters** [59].

2.6.3.1. Qdiscs

El objeto qdiscs, que significa “Quality-Deficit Inverse Congestion Control Scheduler” (Planoificador de Control de Congestión Inverso de Calidad-Deficit), es un concepto fundamental en el core de red de Linux que determina el orden en el que los miembros de la cola, en este caso los paquetes, son seleccionados para su servicio.

Por ejemplo, cuando una herramienta de espacio de usuario necesita transmitir un paquete en un momento dado, ese paquete se entrega al *stack* de red y, finalmente, llega a la interfaz

de red por la cual será transmitido. En ese momento, el paquete se encuentra encolado en una cola, esperando ser transmitido. Estas colas son gestionadas por un qdiscs. El qdiscs por defecto en Linux es un pfifo, que significa “Priority-First-In, First-Out”, señalar el concepto de prioridad, dado que es fundamental. Por tanto lo podemos ver como una cola pura en la que se sigue el orden de llegada (*first-in*) y se atienden los paquetes en ese orden siempre y cuando no haya alguna directriz de prioridad, con una limitación en el tamaño de la cola en términos del número de paquetes que puede contener.

2.6.3.2. Classes

Las clases pueden entenderse como sub-qdiscs dentro de un qdiscs. Una clase puede tener a su vez otras clases, formando sistemas de QoS detallados, como se muestra en la Figura 2.20. Cuando los paquetes son recibidos en una cola gestionada por un qdiscs, pueden ser encolados en base a las características del paquete en otras colas administradas por otras clases. Esto permite, por ejemplo, priorizar el envío de datos de una aplicación sobre otra. Para lograrlo, los paquetes de ambas aplicaciones se clasificarán en clases diferentes, asignándole a una clase más prioridad que a la otra. Esto implica asignar más recursos de transmisión y recepción a la clase prioritaria.

En resumen, las clases en el contexto de los qdiscs representan subdivisiones que permiten una gestión más detallada del tráfico de red. Al agrupar paquetes en diferentes clases y asignar prioridades, se puede establecer un control más minucioso sobre la transmisión y recepción de datos, lo que facilita la implementación de políticas de QoS y la asignación de recursos de manera más eficiente.

2.6.3.3. Filters

En el contexto de TC en Linux, los filtros son objetos utilizados para seleccionar y clasificar paquetes de red con el fin de aplicar acciones específicas sobre ellos. Los filtros permiten establecer reglas que determinan cómo se deben tratar los paquetes en función de diferentes criterios, como direcciones IP, puertos, protocolos, etiquetas VLAN, campos de encabezado, entre otros. Los filtros en TC se utilizan en conjunción con los qdiscs para controlar y gestionar el tráfico de red de manera más granular. Se pueden aplicar diferentes acciones a los paquetes que coinciden con los criterios definidos por los filtros, como encolarlos en clases específicas, modificar sus encabezados, descartarlos, redirigirlos a interfaces de red diferentes, entre otras opciones. Existen varios tipos de filtros disponibles en TC, entre ellos podemos ver los siguientes.

- Tipo U32: Permite filtrar paquetes basados en campos de encabezado, como direcciones IP, puertos y protocolos.
-

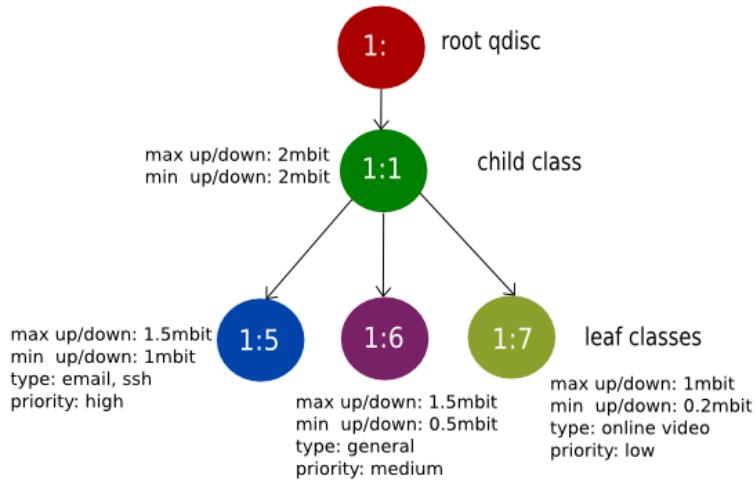


Figura 2.20: Mecanismo de calidad de servicio implementado con clases y sub-clases [60]

- Tipo Basic: Permite filtrar paquetes utilizando criterios básicos como direcciones IP de origen y destino, puertos y protocolos.
- Tipo Route: Permite filtrar paquetes en función de las rutas de red.
- Tipo Berkeley Packet Filter (BPF): Permite utilizar programas BPF para filtrar y procesar paquetes.

Estos son solo algunos ejemplos de los tipos de filtros disponibles en TC. Cada tipo de filtro tiene su propia sintaxis y opciones de configuración específicas. Por tanto, los filtros en TC se utilizan para seleccionar y clasificar paquetes de red en función de diferentes criterios. Estos filtros permiten aplicar acciones específicas a los paquetes que coinciden con los criterios definidos, lo que facilita la gestión y el control del tráfico de red en Linux.

2.6.4. Namespaces

Una *Namespace* se utiliza para encapsular un recurso del sistema operativo en una abstracción que engaña a los procesos dentro de esa *Namespace* haciéndoles creer que tienen su propia instancia aislada del recurso en cuestión, separada del sistema real. Los cambios realizados en los recursos aislados solo son visibles para los procesos que pertenecen a esa *Namespace*, mientras que son invisibles para otros procesos que pertenecen al sistema o a otra *Namespace*.

Existen varios tipos de *Namespaces*, los cuales se detallan en la Tabla 2.1. Cada tipo de

Namespace tiene diversos usos y aplicaciones, pero uno de los más relevantes es el de los contenedores. Los contenedores proporcionan un entorno aislado para ejecutar aplicaciones o herramientas. Actualmente, los contenedores más populares son los de Docker, una plataforma ampliamente utilizada en el mundo del desarrollo de software. Docker aprovecha las bondades del kernel de Linux para aislar recursos y crear instancias aisladas del sistema Host sobre el cual se ejecutan. Utiliza la API proporcionada por el kernel para crear y destruir *Namespaces* según sea necesario. En esencia, Docker es un conjunto de llamadas al sistema que se encarga de gestionar *Namespaces*. Además de la gestión de *Namespaces*, Docker proporciona otras características útiles, como el mecanismo de *copy-on-write* y una configuración de red en modo *bridge* hacia el exterior. Sin embargo, en su núcleo, Docker es simplemente un *wrapper* que facilita la gestión de *Namespaces* con el objetivo de implementar contenedores.

Tipo de Namespace	Descripción
Cgroup	Namespace utilizado generalmente para establecer límites de recursos, como CPU, memoria, lecturas y escrituras a disco, para todos los procesos dentro de la misma Namespace.
Time	Namespace utilizado para establecer una hora del sistema diferente a la del sistema global.
Network	Namespace utilizado para crear una réplica aislada del stack de red del sistema dentro del propio sistema.
User	Namespace utilizado para aislar un grupo de usuarios.
PID	Namespace utilizado para tener identificadores de proceso independientes de otras namespaces.
IPC	Namespace utilizado para aislar los mecanismos de comunicación entre procesos.
Uts	Namespace utilizado para establecer un nombre de host y nombre de dominio diferentes de los establecidos en el sistema.
Mount	Namespace utilizado para aislar los puntos de montaje en el sistema de archivos.

Tabla 2.1: Resumen de los tipos de Namespaces en el Kernel de Linux

2.6.4.1. Persistencia de las Namespaces

Según la página man-page sobre las *Namespaces* [61], es importante tener en cuenta que las *Namespaces* tienen una vida finita. La duración de una *Namespace* dependerá de si está siendo referenciada, lo que significa que vivirá mientras siga siendo referenciada y será destruida cuando deje de serlo. Comprender este concepto de vida finita es crucial para tener una mejor comprensión del funcionamiento interno de Mininet o Mininet-WiFi, ya que estas herramientas aprovechan estos conceptos para optimizar las operaciones y mejorar el rendimiento de sus respectivos emuladores.

Actualmente, una *Namespace* puede ser referenciada de tres maneras diferentes:

- Siempre que haya un proceso en ejecución dentro de dicha *Namespace*.
- Siempre que haya abierto un descriptor de archivo para el archivo identificador de la *Namespace* (por ejemplo, `/proc/pid/ns/tipo_namespace`).
- Siempre que exista un *bind-mount* del archivo (`/proc/pid/ns/tipo_namespace`) de la *Namespace* en cuestión.

Si ninguna de estas condiciones se cumple, el kernel eliminará automáticamente la *Namespace*. En el caso de una *Network Namespace*, las interfaces que se encuentren dentro de la *Namespace* que está siendo eliminada volverán a la *Network Namespace* predeterminada [61]. Es fundamental tener en cuenta estos detalles, ya que ayudan a comprender cómo se manejan las *Namespaces* y cómo se gestionan los recursos en el contexto de Mininet o Mininet-WiFi.

2.6.4.2. Concepto de las Network Namespaces

Una vez comprendido el concepto de *Namespace* en Linux, es necesario introducir las *Network Namespace*, las cuales desempeñarán un papel fundamental en las plataformas utilizadas para emular. Estas *Network Namespace* consisten en réplicas lógicas del *stack* de red predeterminado de Linux, que incluye rutas, tablas ARP, Iptables e interfaces de red [62].

Linux se inicia con una *Network Namespace* predeterminada, conocida como el espacio *root*, que tiene su propia tabla de rutas, tabla ARP, Iptables e interfaces de red. Sin embargo, también es posible crear *Network Namespace* adicionales que no son predeterminadas, crear nuevos dispositivos dentro de esos espacios de nombres o trasladar dispositivos existentes de un espacio de nombres a otro. La herramienta más sencilla para llevar a cabo todas estas tareas de gestión con las Netns es `iproute2` (consulte el Anexo C.1). Esta herramienta, utilizando el módulo `netns`, permite gestionar todo lo relacionado con las *Network Namespace* con nombre (*named network namespaces*).

La característica “con nombre” significa que todas las *Network Namespace* administradas mediante `iproute2` serán persistentes, ya que se realizará un *bind-mount* del archivo identificador de la *Namespace* en cuestión con su nombre correspondiente en el directorio `/var/run/netns`. Un *bind-mount* es una forma de montar un directorio o archivo específico en un lugar diferente dentro del sistema de archivos. Permite enlazar una ubicación existente en el sistema de archivos a otra ubicación, creando una conexión entre ambos. Cualquier cambio realizado en uno de los puntos de montaje se reflejará en el otro. Es útil para compartir datos entre diferentes ubicaciones o para acceder a un directorio o archivo específico desde múltiples ubicaciones dentro del sistema de archivos. A continuación, en el

bloque de Código 2.3, se enumeran los comandos más comunes utilizados para gestionar las *Network Namespace*. Se asume que estos comandos se ejecutan con privilegios de root.

Código 2.3: Casos de uso de las Netns

```

1  # Con el siguiente comando creamos una network namespace
2  ip netns add {netns_name}
3
4  # Listamos las Named Network Namespaces :)
5  ip netns list
6
7  # Movemos una interfaz a una Network Namespace determinada
8  ip netns set {netns_name} Veth
9
10 # Ejecutamos un comando dentro de una Network Namespace
11 ip netns exec {netns_name} {cmd}
12
13 # De esta forma, eliminamos una Network Namespace
14 ip netns del {netns_name}
```

2.6.4.3. Comunicación inter-Namespaces: Veth

Por lo tanto, se puede representar el siguiente diagrama básico para ilustrar el funcionamiento de un par de interfaces Veth asignadas a diferentes *Network Namespaces*. Como se muestra en la Figura 2.21, ambas interfaces están directamente conectadas internamente en el Kernel. Si se generan paquetes desde una *Network Namespace* hacia la otra, estos paquetes viajarán desde un extremo de la Veth directamente al otro extremo de la Veth a través del Kernel, sin ser percibidos por la *Network Namespace* predeterminada. Esta configuración resulta muy útil para establecer enlaces entre nodos de red independientes. Los nodos se replicarán utilizando *Network Namespaces* y los enlaces se establecerán utilizando pares de interfaces Veth. Estos mecanismos son ampliamente utilizados por herramientas de emulación de redes como Mininet o Mininet-WiFi, lo cual se explicará en detalle más adelante.

2.6.5. Subsistema inalámbrico de Linux

El subsistema inalámbrico de Linux, del inglés *Linux Wireless Subsystem*, consiste en un conjunto de módulos presentes en el kernel de Linux. Estos módulos se encargan de manejar la configuración del hardware según el estándar ieee80211, así como de la gestión de la transmisión y recepción de paquetes de datos. En la arquitectura de este subsistema, el primer bloque que encontramos al analizar de abajo hacia arriba es el módulo mac80211_hwsim. Este módulo es responsable de crear las interfaces inalámbricas virtuales en Mininet-WiFi.

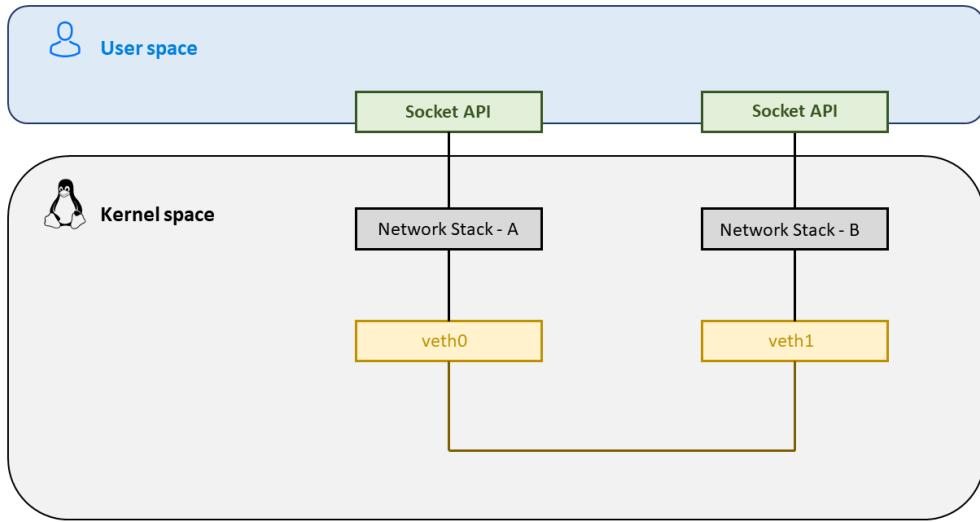


Figura 2.21: Enlace entre interfaces Veth separadas en dos Network Namespaces [18]

El objetivo principal del módulo `mac80211_hwsim` es facilitar a los desarrolladores de controladores de tarjetas inalámbricas la prueba de su código y la interacción con el siguiente bloque llamado `mac80211`. Las interfaces virtualizadas no tienen ciertas limitaciones presentes en el hardware real, lo que facilita la realización de diversas pruebas con diferentes configuraciones sin estar limitados por recursos materiales. Este módulo generalmente recibe un parámetro único, que es el número de “radios” o interfaces virtuales que se desean virtualizar. Dado que las posibilidades ofrecidas por este módulo eran algo limitadas, se han creado varios wrappers que proporcionan funcionalidades adicionales más allá de las que ofrece el módulo en sí. La mayoría de las herramientas creadas hacen uso de la biblioteca Netlink para comunicarse directamente con el subsistema en el kernel y lograr configuraciones adicionales, como agregar un RSSI en particular, o asignar un nombre a la interfaz. Un ejemplo de dicha herramienta es `mac80211_hwsim_mgmt`, que es utilizada por Mininet-WiFi para gestionar la creación de interfaces inalámbricas en cada nodo que las requiere.

Es importante destacar el cambio de paradigma que existe en el subsistema inalámbrico de Linux en relación al concepto de interfaz. Generalmente, se piensa en una interfaz como un elemento que gestiona el acceso al medio en la capa dos y el propio hardware en la capa física, como sería el caso de una interfaz Ethernet. Sin embargo, en el subsistema inalámbrico se descompone la interfaz en dos capas [63]. Una de ellas es la capa física (*PHY*), donde se puede gestionar, por ejemplo, en qué canal está operando la tarjeta inalámbrica emulada. La otra capa es el acceso al medio, representado por las interfaces virtuales que se derivan de una tarjeta inalámbrica. La idea detrás de este paradigma es que se pueden tener

N interfaces virtuales asociadas a una misma tarjeta inalámbrica, aunque es importante destacar que estas interfaces virtuales funcionan principalmente como interfaces Ethernet (a excepción de las que están en modo monitor).

2.6.5.1. Limitaciones del módulo `mac80211_hwsim`

Como se mencionó anteriormente, la mayoría de las interfaces virtuales asociadas a una tarjeta inalámbrica emulada son del tipo Ethernet, lo que implica que todos los paquetes que se reciben están encapsulados con cabeceras Ethernet. Esta situación plantea una limitación, ya que en ciertos casos de uso se desea manipular las cabeceras WiFi. Sin embargo, debido a que las interfaces virtuales son principalmente del tipo Ethernet, no es posible lograr este objetivo.

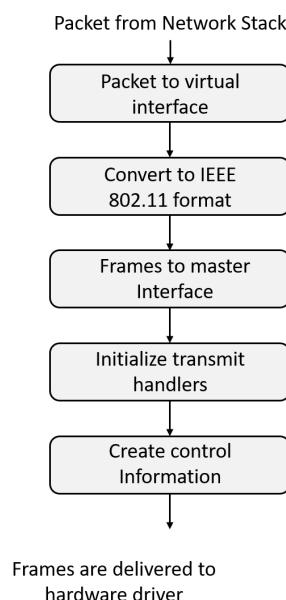


Figura 2.22: Pipeline de transmisión del módulo `mac80211_hwsim` [64]

Sin embargo, ¿cuál es el motivo de convertir las cabeceras WiFi a cabeceras Ethernet? Hasta ahora, la única razón que se ha encontrado para esta decisión de diseño es simplificar la implementación de los controladores que funcionan con el módulo `mac80211_hwsim`. Estos controladores convierten las cabeceras a Ethernet y las entregan al stack de red para que las gestione como paquetes de una red cableada convencional. Este enfoque permite que las aplicaciones que operan a nivel de interfaz sean más fácilmente compatibles y extrapolables.

Es importante mencionar que esto implica un consumo considerable de recursos, ya que el paquete se encola hasta tres veces (controlador, cola Ethernet, cola de la disciplina de planificación de colas) y se requiere tiempo y recursos para llevar a cabo la traducción de las

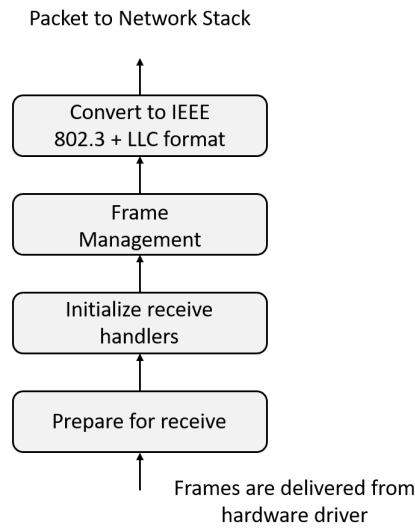


Figura 2.23: Pipeline de recepción del módulo mac80211_hwsim [64]

cabeceras. Es posible encontrar la función en el kernel donde se realiza este proceso [aqui](#)¹⁰. Esta situación ocurre generalmente, pero no siempre, ya que el único modo en el que una interfaz puede recibir paquetes WiFi es en el modo monitor. Sin embargo, el modo monitor está diseñado únicamente para la escucha de paquetes, no para la transmisión. Este modo se puede llevar al límite al realizar una inyección de paquetes (packet injection) a través de la interfaz. Por tanto, cuando se trabaje con este módulo del kernel, se tiene que ser consciente que los paquetes que se verán por las tarjetas WiFi emuladas serán paquetes con cabeceras 802.3 en vez de 802.11.

2.7. Mininet y Mininet-WiFi

En esta sección se abordará el marco teórico relacionado con las principales herramientas de emulación utilizadas en este proyecto. Se prestará especial atención a Mininet, que es la herramienta principal y la base sobre la cual han surgido otras herramientas de emulación, como Mininet-WiFi y Mininet-IoT.

2.7.1. Mininet

Mininet es una herramienta utilizada para emular redes, principalmente del tipo SDN. Permite emular hosts, routers, switches y enlaces en una sola máquina a un coste reducido, siempre y cuando se cuente con el Kernel de Linux en dicha máquina. Para lograr esto, Mi-

¹⁰https://elixir.bootlin.com/linux/latest/C/ident/_ieee80211_data_to_8023

ninet utiliza una forma de virtualización “ligera” que aprovecha las capacidades del Kernel de Linux para virtualizar recursos, como las *Namespaces* (consultar 2.6.4) [65].

La cantidad de recursos virtualizados dependerá de las características de cada nodo, y esto también afectará el rendimiento de la emulación. Por ejemplo, los nodos tipo *Host* en Mininet requieren el uso de una *Network Namespace*, lo que les proporciona su propio *stack* de red y los hace completamente independientes¹¹ del sistema y otros nodos en la red emulada. Sin embargo, por defecto, todos los nodos *Host* comparten el sistema de archivos, la numeración de procesos (PIDs), los usuarios, etc. En términos técnicos, no están completamente aislados como un host real. Esto se debe a que Mininet virtualiza solo los recursos necesarios para llevar a cabo la emulación, lo que mejora el rendimiento y permite que máquinas con recursos limitados puedan realizar la emulación [65].

En cuanto a la creación de topologías en Mininet, existen dos enfoques. El primero es utilizar la API escrita en Python para interactuar con las clases de Mininet. Con esta API, se puede construir la topología importando los módulos y clases necesarios para definirla en un script de Python. El segundo enfoque es utilizar la herramienta llamada **MiniEdit**, que proporciona una interfaz gráfica (GUI) donde los usuarios pueden crear la topología arrastrando y soltando nodos de red. Desde la misma GUI, se puede exportar la topología generada a un archivo (*.mn) para recuperarla más tarde o a un script en Python (*.py) para cargarla en el intérprete de Python cuando sea necesario. Esta herramienta es especialmente útil para aquellos que no tienen conocimientos de programación en Python pero desean utilizar el emulador, lo que representa una ventaja significativa [65].

Por lo tanto, se puede concluir que los aspectos más fuertes de Mininet son los siguientes puntos:

- Mininet es rápido gracias a su diseño basado en *Namespaces*, lo que permite una gestión eficiente de los recursos. En la Sección 2.6.4, se explica cómo se lleva a cabo esta gestión.
- Mininet no consume recursos en exceso, ya que virtualiza únicamente los componentes necesarios para la emulación. Además, se pueden establecer límites máximos de recursos para la emulación en caso de ser necesario.
- Mininet brinda libertad al usuario para crear topologías y escenarios personalizados utilizando la API en Python de Mininet. Estos escenarios pueden ser fácilmente transferidos a otra máquina, ya que solo se requiere compartir el script que describe la

¹¹En lo que respecta a la parte de Red

topología. Es importante tener en cuenta que los resultados de las pruebas pueden variar entre diferentes máquinas, ya que Mininet emula la red en lugar de simularla. Por lo tanto, los resultados dependerán de las condiciones de la máquina donde se ejecuten las pruebas.

Aunque Mininet ofrece muchas ventajas, también tiene una limitación importante que debe tenerse en cuenta. Como se mencionó anteriormente, Mininet utiliza una forma de virtualización “ligera” basada en las *Namespaces* del Kernel de Linux. Si bien esta decisión de diseño proporciona beneficios significativos en términos de rendimiento al aprovechar el propio sistema operativo para virtualizar recursos, surge un problema cuando se intenta exportar el emulador a otra plataforma con un sistema operativo diferente. Es posible que este sistema operativo no admita un equivalente funcional a las *Namespaces* de Linux o, incluso si lo hace, su API para utilizarlas puede ser completamente diferente. Esto puede dificultar o incluso impedir la ejecución de Mininet en plataformas que no porten el kernel de Linux.

2.7.2. Funcionamiento de Mininet

Anteriormente se mencionó que Mininet utiliza *Network Namespaces* como método para virtualizar *stacks* de red independientes y así emular redes con un costo mínimo. En la Figura 2.24, se muestra la arquitectura interna de Mininet para una topología compuesta por dos nodos *Host* y un software switch conectado por TCP a un controlador remoto.

Como se puede observar, cada nodo *Host* está aislado en su propia *Namepace* de red, mientras que el switch se ejecuta en la *Namepace* por defecto (root). La comunicación entre los nodos de esta topología se realiza mediante Veths (consultar 2.6.4.3), las cuales permiten emular los enlaces entre los diferentes nodos de la red.

Una vez presentada toda la teoría sobre Mininet, puede surgir la pregunta de cómo se puede comprobar si realmente utiliza *Network Namespaces*. Para hacerlo, lo primero que se debe hacer es crear el escenario para que Mininet pueda crear las *Network Namespaces* necesarias. En este caso, se utilizará la topología mostrada en la Figura 2.24. Para crear esta topología, solo se necesita tener Mininet instalado y seguir los pasos que se indican en el bloque 2.4. Una vez levantado el escenario, se debería obtener el output indicado en la Figura 2.25.

Código 2.4: Ejecución de Mininet con la topología por defecto

```
1 # Lanzamos Mininet con la topo por defecto :)
2 sudo mn
```

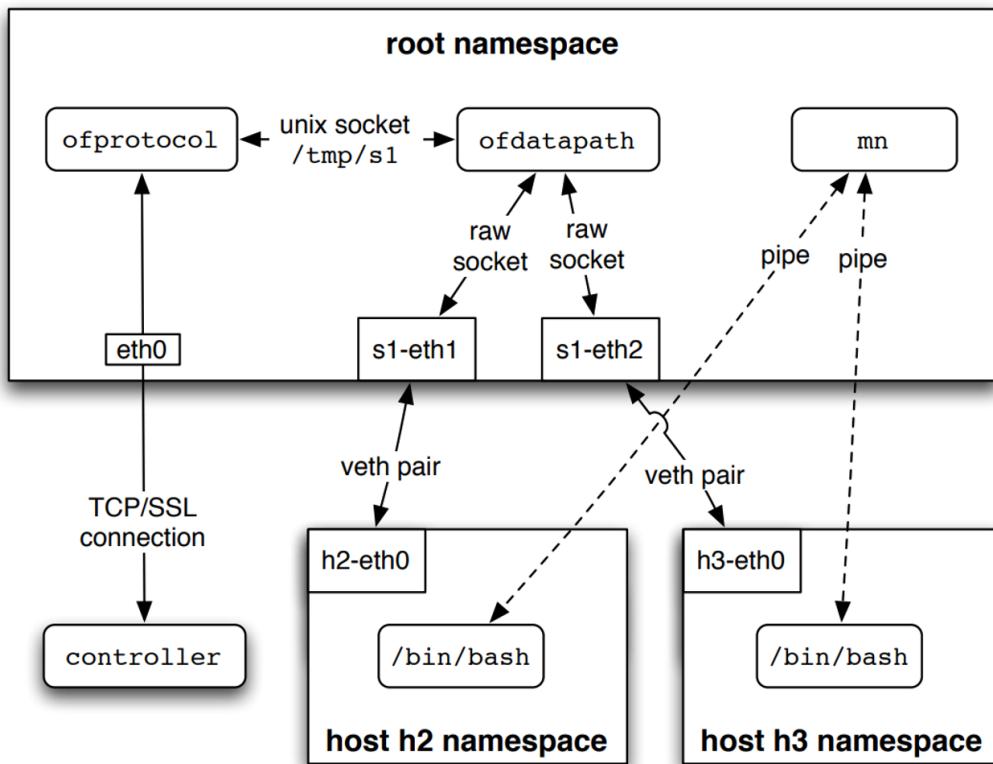


Figura 2.24: Arquitectura de Mininet [66]

Ahora que hemos creado el escenario, podemos verificar si hay *Network Namespaces* en el sistema utilizando el conjunto de herramientas iproute2 (consultar C.1). El comando más utilizado para listar las *Network Namespaces* utilizando el módulo **netns** se muestra en el bloque 2.5.

Código 2.5: Listar Named Network Namespaces

```

1  # Listamos las network namespaces con Nombre! ojito :)
2  sudo ip netns list

```

Al examinar la Figura 2.26, se puede observar que no hay ninguna instancia de "Network Namespace" en el sistema. Esto plantea la pregunta de dónde puede residir el problema. La razón por la cual el comando `ip netns list` no muestra ninguna información es que Mininet no está creando el enlace simbólico (softlink) necesario para que la herramienta pueda listar las "Network Namespaces". Según la documentación del comando, éste busca en el directorio `/var/run/netns/` donde se almacenan todas las "Network Namespaces" con nombres específicos. Estas "Netns" son aquellas en las que se ha realizado un "bind mount" con su nombre correspondiente en dicho directorio para que persistan incluso si no hay nin-

```
n0obie@n0obie-VirtualBox:~$ sudo mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet-wifi> dump
<Host h1: h1-eth0:10.0.0.1 pid=9518>
<Host h2: h2-eth0:10.0.0.2 pid=9520>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=9525>
<OVSCController c0: 127.0.0.1:6653 pid=9511>
mininet-wifi> █
```

Figura 2.25: Salida por pantalla de la ejecución de la topología por defecto

```
n0obie@n0obie-VirtualBox:~$ sudo ip netns list
n0obie@n0obie-VirtualBox:~$ █
```

Figura 2.26: Listado de Named Network Namespaces existentes en el sistema

gún proceso en ejecución en ellas. Como se mencionó anteriormente, las Namespaces tienen una existencia limitada y solo existen mientras sean referenciadas (consulte la Tabla 2.1). Por lo tanto, si no se cumple ninguna condición de referencia, la Namespace en cuestión se eliminará.

Mininet se encarga de recrear la red emulada y, cuando el usuario finaliza la emulación, la red emulada debe desaparecer. Este proceso debe ser lo más rápido y eficiente posible para brindar una mejor experiencia al usuario. La naturaleza del diseño de Mininet sugiere que la creación y destrucción de las *Network Namespaces* están asociadas con la primera condición de referencia de una *Namespace*.

En otras palabras, no tendría sentido realizar enlaces o enlaces simbólicos que luego se deban eliminar, ya que esto implicaría una carga de trabajo significativa para emulaciones de redes grandes y aumentaría el tiempo necesario para limpiar el sistema una vez finalizada la emulación. Además, hay que tener en cuenta que existe una condición que se adapta bien a las necesidades de Mininet: solo se requiere un proceso en ejecución por cada *Network Namespace*, y al realizar la limpieza, solo se deben finalizar los procesos que mantienen las *Network Namespaces*. Cuando no haya más procesos en ejecución en una *Namespace*, el kernel se encargará de eliminarla.

De acuerdo con el razonamiento expuesto, se deberían ver varios procesos que se crean al iniciar el escenario en Mininet. Cada uno de estos procesos deberá tener un archivo de *Network Namespace* (`/proc/pid/ns/net`) con un número de nodo único (*inode*) para los procesos que se ejecutan en diferentes *Network Namespaces*.

```
n0obie@n0obie-VirtualBox:~$ sudo ps aux | grep mininet
n0obie  2868  0.0  0.0  21312   944 pts/13   S+  13:42   0:00 grep --color=auto mininet
root    9511  0.0  0.0  28140  3436 pts/9   Ss+ 12:29   0:00 bash --norc --noediting -is mininet:c0
root    9518  0.0  0.0  28136  3572 pts/10  Ss+ 12:29   0:00 bash --norc --noediting -is mininet:h1
root    9520  0.0  0.0  28136  3364 pts/11  Ss+ 12:29   0:00 bash --norc --noediting -is mininet:h2
root    9525  0.0  0.0  28136  3416 pts/12  Ss+ 12:29   0:00 bash --norc --noediting -is mininet:sl
n0obie@n0obie-VirtualBox:~$
```

Figura 2.27: Listado de procesos con referencias a Mininet

Si examinamos el archivo `/proc/pid/ns/net` para cada proceso mencionado en la Figura 2.27, podremos determinar cuáles de ellos se encuentran en una *Network Namespace* distinta, según el valor del *inode*. Por ejemplo, verifiquemos los procesos asociados a los Host1 y Host2.

```
n0obie@n0obie-VirtualBox:~$ sudo ps aux | grep mininet | grep h1
root    9518  0.0  0.0  28136  3572 pts/10  Ss+ 12:29   0:00 bash --norc --noediting -is mininet:h1
n0obie@n0obie-VirtualBox:~$ sudo ls -la /proc/9518/ns/net | grep -e 'net:[0-9]+\|'
lwxrwxrwx 1 root root 0 jun 13 13:47 /proc/9518/ns/net -> net:[4026532227]
n0obie@n0obie-VirtualBox:~$
```

Figura 2.28: Información de contexto sobre el proceso del Host1

```
n0obie@n0obie-VirtualBox:~$ sudo ps aux | grep mininet | grep h2
root      9520  0.0  0.0  28136  3364 pts/11    Ss+   12:29   0:00 bash --norc --noediting -is mininet:h2
n0obie@n0obie-VirtualBox:~$ sudo ls -la /proc/9520/ns/net | grep -e 'net:[\[\d-\d\]\+\]' 
lwxrwxrwx 1 root root 0 jun 13 13:49 /proc/9520/ns/net -> net:[4026532343]
n0obie@n0obie-VirtualBox:~$
```

Figura 2.29: Información de contexto sobre el proceso del Host2

Como se puede observar, hay diferentes *inodes*, archivos distintos y *Network namespaces* diferentes. Esta prueba demuestra cómo Mininet utiliza procesos de bash para mantener las *Network Namespaces* de los nodos que lo requieren.

2.7.3. Mininet-WiFi

Mininet-WiFi [67], es una extensión (*fork*) del emulador de redes Mininet que permite la emulación de redes inalámbricas en entornos virtuales. Utiliza el módulo del kernel llamado mac80211_hwsim para simular interfaces inalámbricas virtuales y puntos de acceso. El módulo mac80211_hwsim es un componente del kernel de Linux que proporciona una funcionalidad de emulación de hardware para interfaces inalámbricas. Permite crear interfaces inalámbricas virtuales que se comportan como dispositivos físicos, lo que permite la emulación y prueba de escenarios de red inalámbrica sin necesidad de hardware físico.

Mininet-WiFi hace uso de este módulo para crear interfaces inalámbricas virtuales y puntos de acceso virtuales en las emulaciones de redes. Esto permite simular la configuración de redes inalámbricas complejas, incluyendo la conexión de dispositivos a puntos de acceso virtuales, la emulación de diferentes canales inalámbricos y la aplicación de configuraciones específicas de los dispositivos inalámbricos. Al utilizar el módulo mac80211_hwsim, Mininet-WiFi puede proporcionar una capa de emulación adicional para redes inalámbricas, permitiendo a los usuarios experimentar con aplicaciones, protocolos y algoritmos inalámbricos en un entorno virtual controlado. Esto es especialmente útil para la investigación, desarrollo y pruebas de nuevas soluciones de redes inalámbricas, ya que se pueden crear y reproducir escenarios de red de manera rápida y eficiente.

Como se puede ver en la Figura 2.30, Mininet-WiFi aparte de utilizar el módulo del kernel mac80211_hwsim para la emulación del entorno inalámbrico, también hace uso de una serie de herramientas para controlarlo y gestionarlo. De forma adicional, si nos fijamos en las herramientas que se despliegan para la emulación se puede ver como los puntos de acceso requieren de un proceso propio (**hostAPd**) para correr las funcionalidades de AP.

La arquitectura de virtualización utilizada en Mininet-WiFi es similar a la de Mininet. Se

utiliza la herramienta `mnexec`¹² para lanzar diferentes procesos de bash en nuevas Network Namespaces, uno por cada nodo independiente de la red. Estos procesos alojarán todos los procesos relacionados con los nodos de la red. Una vez finalizada la emulación, se finalizan los procesos de bash, lo que elimina las condiciones de referencia de las Network Namespaces y permite que el kernel las elimine.

De esta manera, los nodos de la red quedan aislados entre sí, y lo único que falta virtualizar son las capacidades inalámbricas de los nodos que las requieran. Para ello, se utiliza el subsistema inalámbrico del kernel de Linux, específicamente el módulo `mac80211_hwsim`, que crea las interfaces inalámbricas en el equipo. Este módulo se comunica con el framework `mac80211`, que proporciona las capacidades de gestión de acceso al medio de la interfaz inalámbrica. Además, en el espacio del kernel, hay un bloque adicional llamado `cfg80211`, que sirve como API para la configuración de las cabeceras 802.11. Esta configuración se puede realizar mediante la interfaz de usuario de espacio de usuario llamada `n180211`. Para la configuración de los puntos de acceso, se utiliza el programa `HostApd`, que, al recibir la configuración del punto de acceso y la interfaz en la que debe funcionar, emula el comportamiento de un punto de acceso estándar.

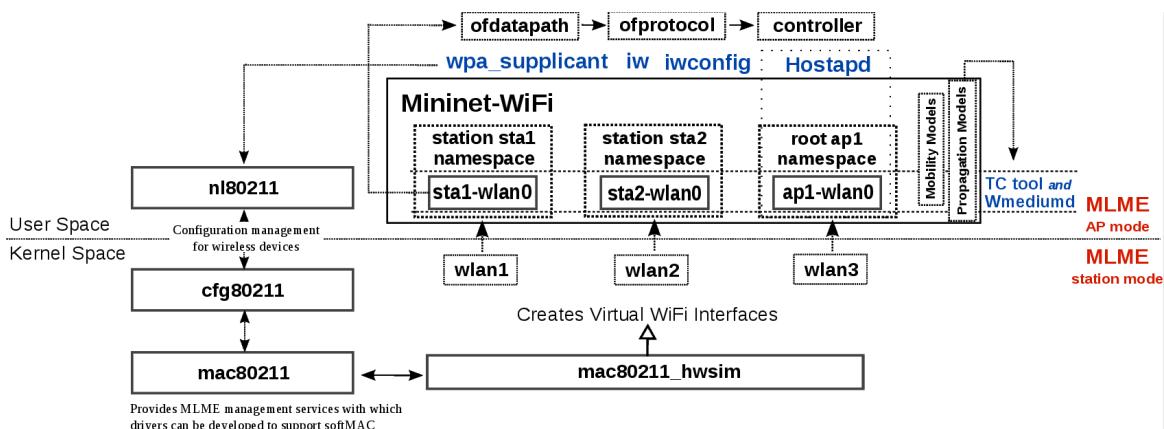


Figura 2.30: Arquitectura básica de Mininet-WiFi [67]

En cuanto a la jerarquía de clases, cabe mencionar que es bastante similar a la de Mininet. Destacan dos clases clave en la jerarquía de Mininet-WiFi: `Node_Wifi`, de la cual heredan todos los nodos con capacidades inalámbricas de Mininet-WiFi, y la clase `IntfWireless`, de la cual heredan todos los tipos de enlaces disponibles en Mininet-WiFi bajo el estándar `ieee80211`. A continuación, en las figuras 2.31 y 2.32, se muestran los UML correspondientes a estas clases.

¹²<https://github.com/intrig-unicamp/mininet-wifi/blob/master/mnexec.c>

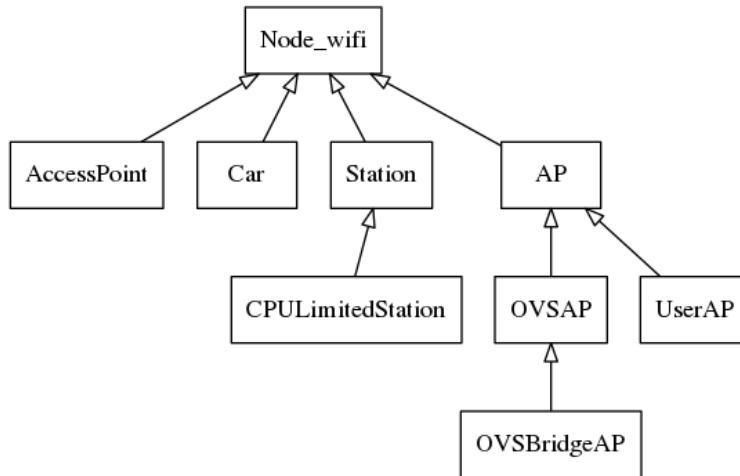


Figura 2.31: UML de clases de tipo Nodo.

Como se puede observar en los diagramas UML, se ha logrado aislar la funcionalidad común en las clases padre con el objetivo de optimizar el código de las clases hijas. Esto permite agregar nuevos tipos de enlaces y nodos en Mininet-WiFi de manera más sencilla y accesible.

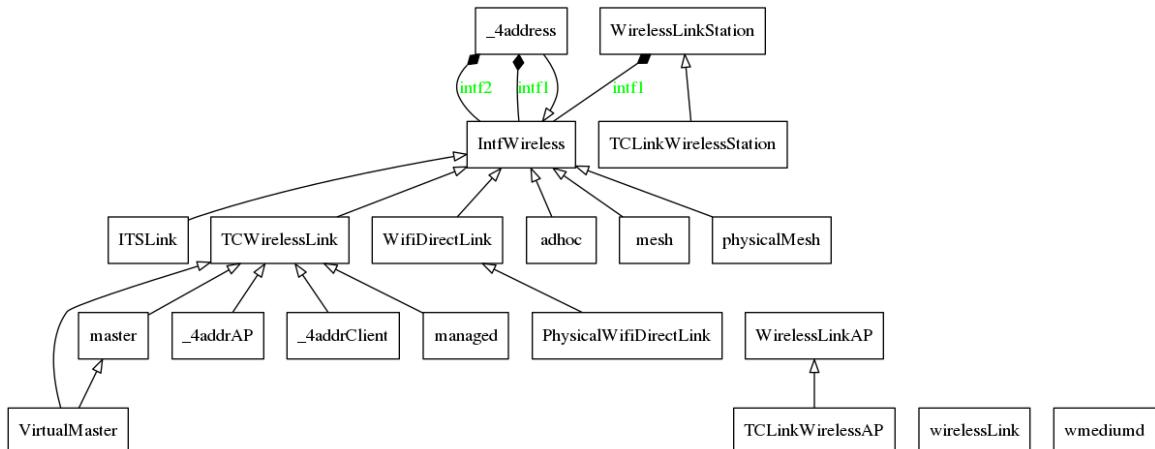


Figura 2.32: UML de clases de tipo Interfaz.

En conclusión, Mininet-WiFi utiliza el módulo del Kernel `mac80211_hwsim` para emular interfaces inalámbricas virtuales y puntos de acceso virtuales en las simulaciones de redes inalámbricas, lo que permite a los usuarios probar y experimentar con configuraciones, aplicaciones y protocolos inalámbricos en un entorno virtual controlado.

Sin embargo, existe otra opción llamada Mininet-IoT, que se deriva de Mininet-WiFi y está diseñada específicamente para emular redes de baja capacidad bajo el estándar `ieee802154` y la capa de adaptación 6LoWPAN. A diferencia de Mininet-WiFi, Mininet-IoT utiliza el módulo del kernel `mac802154_hwsim` para emular interfaces y medios inalámbricos. Aunque la gestión de nodos, interfaces y enlaces es similar en ambas herramientas. Para facilitar el mantenimiento y compartir código común, el desarrollador principal, Ramon Fontes, creó una clase agnóstica que administra los módulos del kernel en Mininet-WiFi y migró todo el proyecto de Mininet-IoT a Mininet-WiFi. Esto permite a los usuarios de Mininet-WiFi establecer enlaces de baja capacidad en sus topologías inalámbricas.

En resumen, Mininet-IoT es una extensión de Mininet-WiFi que permite emular redes de baja capacidad bajo el estándar `ieee802154` y 6LoWPAN. Aunque comparte muchas características con Mininet-WiFi, utiliza el módulo `mac802154_hwsim` para la emulación inalámbrica. Esta herramienta brinda a los usuarios la capacidad de simular y experimentar con topologías inalámbricas de baja capacidad, ofreciendo una alternativa adicional a Mininet-WiFi.

2.8. Contiki-ng

Contiki es un sistema operativo diseñado específicamente para dispositivos de baja capacidad, como sensores. Fue desarrollado por Adam Dunkels en colaboración con Bjorn Gronvall y Thiemo Voigt en 2002. A lo largo de los años, el proyecto Contiki ha crecido enormemente y ha involucrado a numerosas empresas y cientos de colaboradores en su repositorio de GitHub¹³. El objetivo principal de Contiki era proporcionar a los nodos de las redes de sensores inalámbricos (WSN) un sistema operativo liviano capaz de cargar y descargar servicios de forma dinámica [68].

El kernel de Contiki se basa en un modelo orientado a eventos y admite multitarea con prelación. Está escrito en lenguaje C y ha sido portado a diversas arquitecturas de microcontroladores, como el MSP430 de Texas Instruments y sus variantes.

En un sistema que ejecuta Contiki, se divide en dos partes claramente diferenciadas del firmware, como se muestra en la Figura 2.33: el núcleo (*core*) y los programas o servicios cargados. Esta partición se realiza durante la etapa de compilación y es independiente del objetivo (*target*) donde se desplegará el sistema. El núcleo (*core*) incluye el propio kernel, un conjunto de servicios base (como temporizadores y controladores), bibliotecas, controlado-

¹³<https://github.com/contiki-os/contiki>

res y el stack de comunicación. Los programas o servicios cargados se mapean en memoria mediante el cargador del kernel durante el tiempo de ejecución.

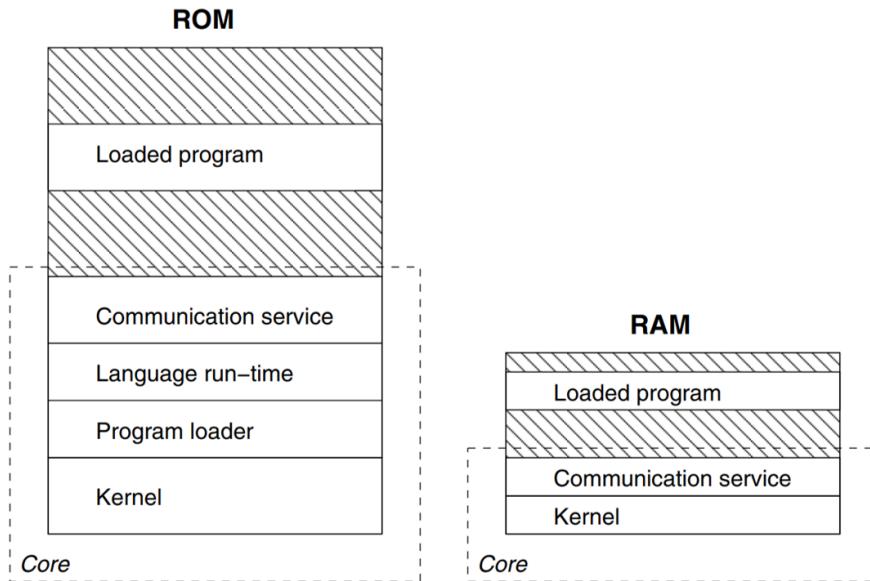


Figura 2.33: Gestión de la memoria en un sistema con Contiki OS [68]

En los últimos años ha surgido un nuevo proyecto llamado **Contiki-ng**¹⁴, el cual es un fork del sistema operativo Contiki. Actualmente, toda la comunidad de Contiki se está centrando en este nuevo proyecto, el cual proporciona un stack de comunicación más compatible con las especificaciones RFC, ofrece soporte para protocolos como IPv6/6LoWPAN y 6TiSCH, y se está volviendo cada vez más popular por su capacidad de brindar soporte a microcontroladores con arquitectura ARM [69].

2.8.1. Simulador Cooja

El flujo de trabajo con Contiki o Contiki-ng varía dependiendo de si se trabaja con hardware real o si se realiza una simulación de los programas desarrollados. En el caso de trabajar con hardware real, el proceso consiste en compilar el sistema operativo y los programas específicos utilizando el objetivo (target) correspondiente, lo que generará un archivo binario que se puede cargar en la memoria del hardware.

Por otro lado, si se opta por la simulación, se utilizará el simulador llamado Cooja¹⁵. Cooja

¹⁴<https://github.com/contiki-ng/contiki-ng>

¹⁵<https://github.com/contiki-ng/cooja/tree/master>

es un simulador escrito en Java que permite simular una serie de nodos IoT. Al simular, es posible observar el comportamiento del programa desarrollado en diferentes plataformas. El proceso de compilación del núcleo (core) de Contiki y los programas desarrollados está integrado en el propio simulador, lo que facilita al usuario la compilación de sus programas para diferentes tipos de nodos. Cada simulación se puede guardar en un archivo con extensión ***.csc**, que almacena todos los datos de la simulación, como la semilla (*seed*), las posiciones y los tipos de nodos, utilizando una estructura XML.

De esta manera, tanto si se trabaja con hardware real como si se realiza una simulación, Contiki y Contiki-ng ofrecen herramientas y entornos integrados que permiten desarrollar y probar programas para sistemas embebidos y dispositivos IoT. A continuación, en la Figura 2.34, se indica la interfaz gráfica del simulador.

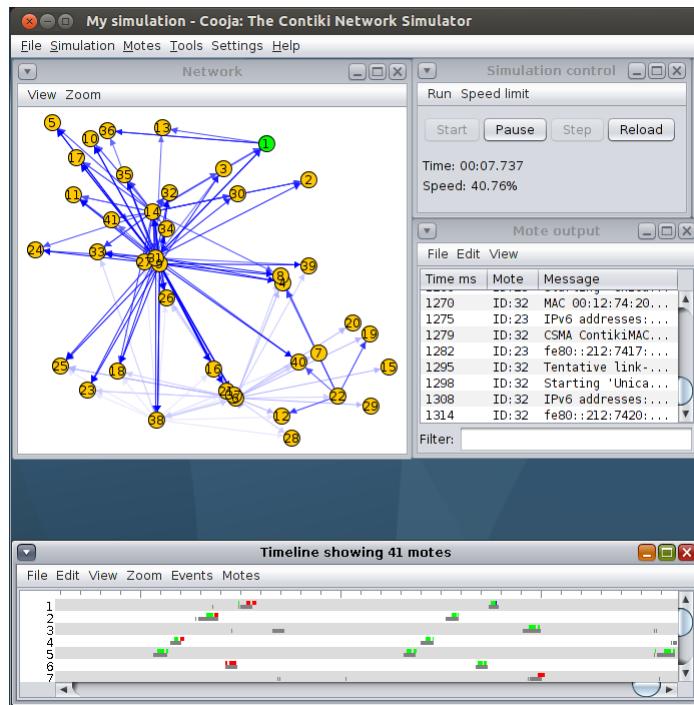


Figura 2.34: Interfaz gráfica del simulador Cooja [70]

2.9. Contribuciones en GitHub

El proyecto GitHub es una plataforma que proporciona alojamiento de repositorios. En el presente TFM, se utilizará la herramienta de control de versiones Git¹⁶, junto con GitHub

¹⁶<https://git-scm.com/>

como plataforma para alojar el código desarrollado. Sin embargo, GitHub no se utilizará únicamente para almacenar el código del TFM, sino que se aprovechará su carácter público para ofrecer documentación y ejemplos a todos los usuarios interesados que visiten el repositorio.

- Enlace al repositorio del TFM: <https://github.com/davidcawork/TFM>
- Enlace al repositorio de la tesis: <https://github.com/davidcawork/tfm-thesis>

El repositorio incluye ficheros `README.md` en todos los directorios, que proporcionan explicaciones y análisis teóricos que permiten al visitante comprender la naturaleza de las pruebas, sus objetivos y las conclusiones que se pueden extraer de ellas. El propósito del repositorio es doble, ya que no solo almacena el código, sino que también ayuda a difundir los contenidos del proyecto. Además, se han ofrecido contribuciones útiles que pueden beneficiar a otros repositorios a través de solicitudes de extracción (*pull requests*). Durante este proyecto se ha contribuido de forma muy activa en varios proyectos de código abierto dado que la naturaleza del mismo abarcaba varias herramientas. La tabla 2.2 muestra todas las contribuciones realizadas.

Contribución	Enlace al Pull-Request
Dar soporte al bmv2 entorno P4 en Mininet-WiFi	https://github.com/intrig-unicamp/mininet-wifi/pull/302
Arreglar la compilación del BOFUSS en Mininet-WiFi	https://github.com/intrig-unicamp/mininet-wifi/pull/495
Aclarar el uso de ONOS-gui	https://groups.google.com/a/opennetworking.org/g/onos-dev/c/mbfyCaoeCdU
Resolver problema interfaz dptcl con el BOFUSS	https://github.com/mininet/mininet/issues/745
Dar escenarios funcionales en la 22.04 de Mininet con Onos	https://groups.google.com/a/opennetworking.org/g/onos-dev/c/aOVjAOXw_-M

Tabla 2.2: Resumen de contribuciones realizadas durante todo el proyecto del TFM

3. Diseño del protocolo de control in-band

En este capítulo se abordará una fase fundamental del proyecto, centrada en el diseño de un protocolo de control in-band para la gestión de redes SDN. Además, se ha consultado un análisis de soluciones anteriores basadas en el enfoque in-band [19], donde se exploran diferentes propuestas y se evalúan sus fortalezas y debilidades.

El objetivo principal será definir las funcionalidades básicas que debe poseer el protocolo de control in-band, considerando los requisitos específicos del proyecto y las necesidades de los entornos de redes actuales. Se examinarán aspectos clave, como la capacidad de establecer una conexión entre los nodos de la red y el controlador, el manejo eficiente del plano de datos para la transmisión de información de control y la escalabilidad para adaptarse a entornos de redes heterogéneas y de gran tamaño. Además, se proporcionará una explicación detallada del funcionamiento del protocolo diseñado, describiendo los diferentes componentes, los mensajes intercambiados entre nodos y controlador, así como los procedimientos de configuración y gestión de la red. Se analizarán las decisiones de diseño tomadas y se justificarán en base a los objetivos del proyecto y las características de los entornos de redes abordados.

Por último, se tomará una decisión sobre la plataforma más adecuada para la implementación del protocolo de control in-band. Se evaluarán diferentes opciones, considerando factores como la disponibilidad de herramientas y tecnologías relevantes, la compatibilidad con los requisitos del proyecto y la viabilidad de su implementación en entornos reales.

3.1. Protocolo In-Band

En esta sección, se tomará la decisión de seleccionar el protocolo de control in-band que se utilizará en el proyecto. Después de una evaluación de las opciones disponibles, se ha decidido utilizar el protocolo IoTorii [71], basado en el enfoque de enrutamiento jerárquico, como la solución más adecuada. Esta elección se respalda por el hecho de que el autor del proyecto ha participado activamente en el desarrollo del protocolo IoTorii, lo que garantiza un conocimiento profundo y una experiencia práctica en su implementación.

El protocolo IoTorii ofrece una serie de ventajas significativas para el control in-band en entornos de IoT. Su enfoque jerárquico de etiquetado permite una gestión eficiente y escalable de la red, al tiempo que proporciona una mayor flexibilidad y adaptabilidad a las necesidades específicas del proyecto. Además, IoTorii ha sido probado y validado en diversas situaciones y escenarios, demostrando su eficacia y confiabilidad en la práctica.

En este contexto, se ha tomado la decisión de hacer uso de los caminos generados por el protocolo IoTorii, donde cada nodo de la red posee rutas distintas para alcanzar el nodo raíz de la topología. En el caso de este proyecto, el nodo raíz se designará como el controlador (o el nodo que brinda acceso al controlador). Por lo tanto, la innovación radica en la implementación de IoTorii en un entorno de redes SDN. La integración de IoTorii con el entorno SDN permitirá aprovechar las capacidades y ventajas de ambos enfoques. La combinación de la eficiencia y escalabilidad jerárquica de IoTorii con la flexibilidad y el control centralizado de SDN ofrece un enfoque prometedor para el desarrollo y la gestión de redes IoT. Este enfoque innovador proporcionará una base sólida para llevar a cabo el proyecto y permitirá explorar nuevas posibilidades y mejoras en el ámbito del control in-band para entornos de IoT.

3.1.1. Protocolo IoTorii

El protocolo IoTorii fue diseñado para trabajar en redes de baja capacidad en entornos IoT, tratando de solventar carencias de otros protocolos del mismo ámbito como *Routing Protocol for Low Power and Lossy Networks* (RPL), basándose en dos principios: (1) Intercambiar menos mensajes y reducir el tamaño de las tablas de rutas, y (2) Ofrecer resiliencia a través de caminos de *back-up*, con una exploración inicial relativamente rápida. Dichos principios básicos tratan de impactar sobre el consumo y la robustez de las redes de baja capacidad, factores clave en este tipo de redes.

3.1.1.1. Operativa del protocolo IoTorii

El objetivo de IoTorii es dar cada nodo de la red por lo menos una etiqueta jerárquica, en adelante Hierarchical Local MAC (HLMAC), para establecer una jerarquía en la topología. Otra forma de verlo puede ser ver la jerarquía como un árbol que se va ramificando, y está enraizado en el nodo que actúa como *root*. Las HLMAC tratan de proveer de más significado a las MACs convencionales. IoTorii está basado en el protocolo GA3 [72], el cual trataba de buscar el mismo etiquetado jerárquico pero en redes cableadas de centros de datos. El principio de estas etiquetas está basado en una revisión del estandar [ieee802](#) que se hizo para dar cabida al uso de direcciones MAC mejoradas para el reenvío de paquetes en red [73].

El proceso de la difusión de dichas etiquetas se pueden ver en la Figura 3.1. El primer paso es establecer en la topología quién va a actuar como nodo raíz o *root*. En este caso, según se puede apreciar, es el nodo A. Este nodo será el encargado de iniciar todo el proceso de asignación de etiquetas en la topología. Se quiere mencionar que puede haber más de un nodo raíz según la definición del protocolo, pero en este caso solo se va a tomar un único nodo *root*. La asignación de etiquetas empezará desde el *root* haciendo uso de un mensaje de tipo *SetHLMAC*, el cual consiste en mandar la dirección HLMAC del remitente más un sufijo que se asociará al vecino. A su vez, un vecino es todo nodo que se encuentra en el rango de cobertura de otro, y se notifican entre ellos de su presencia mediante un mensaje de tipo *Hello*. Dicho mensaje es una trama vacía donde solo anuncian su dirección MAC real.

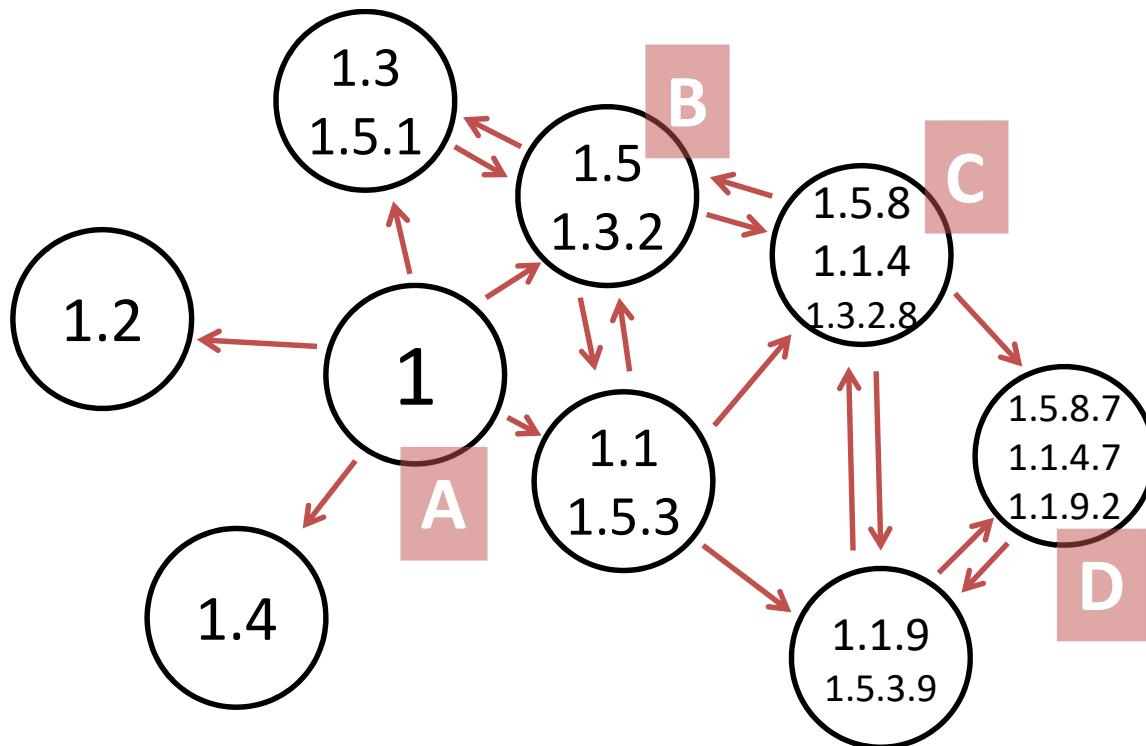


Figura 3.1: Operativa del protocolo de IoTorii [71]

Por tanto, según se ha explicado, habrá dos tipos de mensajes en este protocolo los mensajes de tipo *Hello* y de tipoq *SetHLMAC*. Este último, recordemos que asigna una nueva HLMAC, la cual se compone de la HLMAC del remitente más un sufijo único que se asocia al vecino en cuestión. Por ejemplo, en la Figura 3.1, el nodo A asigna a sus vecinos los su-

fijos 1, 2, 3, 4, 5, por lo que las HLMAC que mandará respectivamente serán las siguientes, [1.1], [1.2], [1.3], [1.4], [1.5]. Dichos sufijos únicos por vecino se van asignando según se reciben mensajes de tipo *Hello*, por lo que, se va construyendo una asociación de MAC real con sufijo único.

Los mensajes de tipo *Hello* se tienen que transmitir de forma periódica para que todos los nodos de la red tengan actualizada su lista de vecinos, ya que en este caso no se tiene una red cableada donde las relaciones entre los nodos son constantes, sino, nodos en un escenario inalámbrico, por lo que movilidad es inherente. Por ello, de forma periódica se tiene que reafirmar si los nodos siguen siendo vecinos y teniendo alcance entre ellos.

Cuando un nodo recibe un mensaje de tipo *SetHLMAC*, si la HLMAC es válida, lo almacenan en su tabla de posibles rutas al nodo raíz, y lo reenvía a todos sus posibles vecinos poniendo sus respectivos sufijos únicos. Este proceso se lleva a cabo de forma iterativa hasta que se han difundido todas las posibles HLMAC o hasta que se haya llegado al número máximo de HLMAC por nodos (parámetro configurable del protocolo). Por ejemplo, si nos fijamos de nuevo en la Figura 3.1, podemos ver como siguiendo con este proceso el nodo B recibe del nodo root la HLMAC [1.5], por lo que, una vez almacenada la etiqueta la difundirá a todos sus posibles vecinos. El nodo B generará las etiquetas [1.5.8], [1.5.1], [1.5.3], y las mandará a sus nodos vecinos. La etiqueta [1.5.8] le llegará al nodo C desde el nodo B, pero también le llegará la etiqueta [1.3.2.8] a través del nodo B (como se puede ver el sufijo es el mismo en ambos casos), las guardará y las difundirá a todos sus vecinos. Para prevenir bucles solo hay que inspeccionar si la HLMAC que te llega es hija de algunas que tienes guardadas en tu tabla de rutas. Por ejemplo, el nodo B descartará todas HLMAC que le lleguen siguiendo el formato [1.5.X.Y] o [1.3.2.X.Y]. Todo este proceso de difusión de etiquetas HLMAC se encuentra descrito en el bloque de algoritmo 1.

Una vez convergida la difusión de etiquetas HLMAC, se tendrá construido un árbol enraizado en el nodo establecido como *root*, y cada hoja del árbol tendrá una o más rutas para alcanzar al raíz.

Algoritmo 1: Asignación de HLMACs en IoTorii [71]

```

1 end Hello;
2 while frame received do
3   if Hello then
4     if MAC not in Hello table then
5       assign unique suffix;
6       save tuple {MAC,suffix};
7       if root node then
8         create SetHLMAC with HLMAC=1;
9         for each tuple in Hello table do
10          | add tuple to SetHLMAC;
11        end
12        broadcast SetHLMAC;
13      end
14    else
15      | discard;
16    end
17  else if SetHLMAC then
18    if HLMAC (or prefix) not in HLMAC table then
19      save HLMAC in HLMAC table;
20      create SetHLMAC with received HLMAC;
21      for each tuple in Hello table do
22        | add tuple to SetHLMAC;
23      end
24      broadcast SetHLMAC;
25    else
26      | discard;
27    end
28  else
29    | discard;
30  end
31 end

```

3.1.1.2. Configuración del protocolo IoTorii

La difusión de las HLMACs puede ser configurada para ajustarse mejor sobre la red que se vaya a trabajar. Se definen tres parámetros de diseño.

- Longitud de HLMAC, establece como de larga podrá ser una etiqueta HLMAC. Este parámetro nos indica a cuantos saltos estaremos desde una hoja del árbol, al nodo raíz.

- Número de bits para sufijos únicos, es decir, qué amplitud de HLMAC vamos a tener. Esto definirá cuantos vecinos recibirán etiquetas desde un nodo dado. Por ejemplo, si tenemos 2 bits, podremos dar cabida hasta 4 vecinos.
- Número de HLMAC, indica el número de rutas máximo que cada nodo guardará en su tabla de rutas al nodo raíz.

Todos los parámetros anteriormente mencionados afectarán también sobre el tamaño de la trama generada de tipo *SetHLMAC*, la cual se puede apreciar en la Figura 3.2. Otro parámetro configurable es la periodicidad de los mensajes de tipo *Hello*, los cuales son fundamentales para gestionar la movilidad de los nodos en la red.

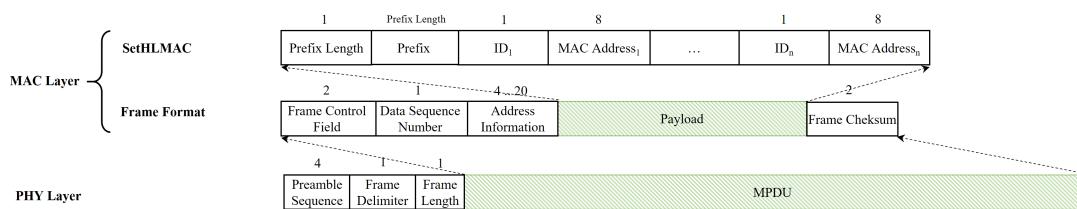


Figura 3.2: Mensajes de control en IoTorii [71]

3.2. Plataforma de desarrollo y validación

En este punto se va a valorar qué plataforma se va a utilizar para desarrollar y validar el protocolo de comunicación in-band. La primera cuestión que debe considerarse al elegir una plataforma de este tipo es la siguiente: ¿Se optará por la simulación o la emulación?. Aunque mucha gente considera que estos términos son equivalentes, no son lo mismo. Cuando hablamos de simulación nos referimos a un software que calcula el resultado de un evento dado un comportamiento esperado, donde la escala de tiempos de la simulación no se corresponde con la realidad. Es decir, el proceso a simular puede tener una duración de dos horas, pero como lo que se gestionan son los eventos asociados al proceso, el tiempo de la simulación solo vendrá regido por lo que tarde en procesar dichos eventos. Por otro lado, la emulación recrea el escenario bajo estudio en su totalidad en un hardware específico para luego estudiar su comportamiento. Donde la escala de tiempos de la emulación es el mismo que el tiempo real del proceso a emular. Un ejemplo para diferenciar ambos podría ser pensar en la cabina de un avión. Si jugáramos a un videojuego como Flight Simulator¹ estaríamos simulando el vuelo, pero si practicáramos utilizando una escala 1:1 (Ver Figura 3.3) con controles reales estaríamos hablando entonces de emulación.

¹<https://www.flightsimulator.com/>



Figura 3.3: Entorno de emulación real de una cabina de avión a escala 1:1 [74]

Una vez que ya tenemos claro cuales son las diferencias sobre la emulación y la simulación, vamos a ver que opciones tenemos en ambos ámbitos para elegir una plataforma que se adecue a las necesidades del proyecto. De entre todas las opciones barajadas, nos hemos quedado con las dos opciones en las cuales el autor tiene más experiencia, y en las cuales se pueden virtualizar tanto enlaces inalámbricos como entornos SDN.

- Contiki-ng haciendo uso de su simulador integrado llamado Cooja, explicado anteriormente en la Sección 2.8 del estado del arte.
- Mininet-WiFi, explicado anteriormente en la Sección 2.7.3 del estado del arte.

La primera opción según se ha explicado ya es un entorno de simulación para dispositivos IoT, donde se podría llegar a meter un agente SDN según se ha podido investigar [75, 76]. Esta opción es muy interesante dado que se utilizaría la tecnología radio más acorde a los dispositivos IoT, ya que implementan **ieee802154**. Sin embargo, lo malo de esta opción es que si bien es cierto que hay agentes SDN ya disponibles, no hay agentes de control SDN programados para Cooja, por lo que habría que programarse un controlador SDN haciendo uso de la plataforma base de Contiki-ng para poder llegar a simular el protocolo. Por tanto, el objetivo final del proyecto pasaría a un segundo plano, ya que programar un controlador SDN desde cero no es una tarea sencilla.

Por otro lado, tenemos a Mininet-WiFi, con su modo de trabajo `ieee80211` o `ieee802154`. Trabajar con un modo u otro es cuestión de indicar que módulo se carga en el Kernel de Linux, si `mac80211_hwsim` o `mac802154_hwsim`. En esta opción estaríamos emulando según se ha explicado anteriormente, en el estado del arte. Recordemos que Mininet y todas las herramientas que nacen de ella hacen uso de las bondades del Kernel para ofrecer mecanismos de aislamiento para llegar a emular. Por ello, tendremos a nuestra disposición un entorno Linux donde a la par que emulamos podemos tener cualquier otra herramienta corriendo en la misma máquina. Esto es interesante porque abre la puerta a tener cualquier controlador SDN corriendo en la máquina, teniendo tanto entorno inalámbrico como entorno SDN ya habilitados, teniendo que preocuparnos exclusivamente en el desarrollo del protocolo.

Ambas opciones presentan ventajas y desventajas, sin embargo, se ha tomado la decisión de seleccionar el entorno de emulación de Mininet-WiFi como plataforma para el desarrollo y validación. Esta elección se basa no solo en los aspectos mencionados anteriormente, sino también en el hecho de que todo el software desarrollado en un entorno Linux será fácilmente transferible a otros entornos Linux con hardware real. Por ejemplo, si se desea utilizar una Raspberry Pi, a pesar de que esta tiene una arquitectura ARM, se puede adaptar el bytecode generado para que sea compatible con dicha arquitectura, ya que el resto del software será compatible, haciendo que la implementación en nuevos targets sea casi inmediata.

3.3. Agente SDN

En este punto se tiene que valorar qué *software switch* SDN se va a utilizar. Se tendrán en cuenta las condiciones del entorno sobre el cual se van a desplegar los nodos SDN, así como las características propias de cada switch, así como la facilidad y flexibilidad que nos entregue cada uno para desplegarlo sobre la plataforma emulada. Las opciones que se han considerado para este cometido son las siguientes:

- OvS, explicado anteriormente en la Sección 2.5.1 del estado del arte.
- BOFUSS, explicado anteriormente en la Sección 2.5.2 del estado del arte.

Dado que cada herramienta tiene sus propias fortalezas y debilidades, es importante realizar una comparativa para determinar cuál de las dos opciones es más adecuada para nuestro caso de uso específico.

- El OvS tiene mucho más rendimiento que el BOFUSS dado que la mitad de su switch
-

trabaja en espacio de Kernel, mientras que el BOFUSS como su nombre indica, trabaja en espacio de usuario.

- El OvS es mucho más sólido, y por ende popular, dado que tiene una gran comunidad de desarrolladores que someten al software switch a una amplia variedad de test. Mientras que el BOFUSS no tiene un sistema de test, y hasta la fecha únicamente contaba con un maintainer.
- El OvS puede trabajar en modo *standalone*², o en modo software switch SDN, mientras que el BOFUSS requiere de un agente de control que le rellene las tablas de flujo OpenFlow.
- El OvS al trabajar en modo *standalone* se puede encontrar en la mayoría de entornos virtualizados, como infraestructura de red para interconectar instancias virtuales.
- Sin embargo, el OvS al igual que tiene puntos a favor por trabajar a nivel de Kernel, también supone puntos negativos, como por ejemplo la complejidad que tiene para depurarlo y añadir nuevas funcionalidades en él. Mientras tanto, el BOFUSS al trabajar en espacio de usuario, permite una sencilla depuración por ejemplo GDB, y es relativamente accesible añadirle nuevas funcionalidades.

Característica	Software Switch OvS	Software Switch BOFUSS
Solidez	Más sólido dado que tienen una comunidad más grande	Sistema de test inexistente
Popularidad	Ampliamente utilizado en entornos de virtualización	Utilizado por la academia para hacer pruebas de concepto
Rendimiento	Alto rendimiento	Pobre, corre en espacio de usuario
Depuración	Difícil, corre en el Kernel	Asequible, corre en espacio de usuario
Facilidad para añadir código	Difícil, corre en el Kernel	Relativamente sencillo
Versatilidad	Versátil, puede trabajar en standalone	Requiere de un agente de control
Curva de aprendizaje	Muy lenta	Muy rápida

Tabla 3.1: Comparativa del OvS con el BOFUSS

Después de analizar las fortalezas y debilidades de cada opción (véase Tabla 3.1), hemos llegado a una decisión sobre qué controlador utilizar en nuestro proyecto. Tanto OvS como BOFUSS tienen características distintivas que deben ser consideradas en nuestro caso de uso. Pero finalmente se ha elegido BOFUSS como *software switch* SDN, dado que para nuestro proyecto es la opción más adecuada por haber una implementación preliminar de in-band basada en parte en IoTorii³.

²Es decir, puede trabajar de forma autónoma sin la necesidad de un agente de control para operar como un switch

³<https://github.com/NETSERV-UAH/in-BOFUSS>

3.4. Agente de control SDN

En este punto se tiene que valorar qué agente de control SDN, es decir controlador SDN se va a utilizar. Se tendrán en cuenta las condiciones del entorno sobre el cual se va a desplegar, así como las características propias de cada controlador, así como la facilidad y flexibilidad que nos entregue cada uno para desplegarlo sobre la plataforma emulada. Las opciones que se han considerado para este cometido son las siguientes:

- Ryu, explicado anteriormente en la Sección 2.4.1 del estado del arte.
- ONOS, explicado anteriormente en la Sección 2.4.2 del estado del arte.

Como se ha mencionado previamente, cada herramienta presenta sus fortalezas y debilidades. Por lo tanto, llevar a cabo una comparativa es pertinente para nuestro caso de uso, a fin de determinar cuál de las dos opciones resulta más conveniente para este proyecto.

- El controlador ONOS exhibe una mayor potencia en comparación con Ryu, siendo ampliamente utilizado por los operadores de red debido a su destacado rendimiento y solidez.
- El controlador ONOS supera a Ryu en términos de procesamiento de paquetes, evidenciando un rendimiento superior en este aspecto.
- Por otro lado, el controlador Ryu presenta una menor carga y es más sencillo de depurar y ampliar con nuevas funcionalidades en caso de ser necesario.
- Asimismo, debido a su menor tamaño, el controlador Ryu se despliega e instala de manera más rápida en comparación con el controlador ONOS.
- La curva de aprendizaje de Ryu es también más pequeña en comparación con ONOS, dado su tamaño reducido.
- Aunque una ventaja potencial de ONOS es su aplicación de descubrimiento topológico, que cuenta con una interfaz web bastante completa, esta aplicación utiliza el protocolo LLDP para el descubrimiento. Sin embargo, dicho protocolo no permite descubrir la configuración de enlaces inalámbricos, sino solo los nodos, lo cual lo hace inútil en este contexto. Aunque existen algunas publicaciones [77] que abordan esta necesidad, resultaría inviable implementar dicho protocolo en el proyecto, ya que excedería los objetivos temporales y de alcance establecidos.

Característica	Controlador ONOS	Controlador Ryu
Solidez	Más sólido dado que tienen una comunidad más grande	Sistema de test bastante pobre
Popularidad	Ampliamente utilizado por los operadores de red	Utilizado por la academia para hacer pruebas de concepto
Rendimiento	Alto rendimiento	Es un script de Python, lenguaje interpretado
Depuración	Al ser tan grande es difícil depurarlo	Es un script de Python, es fácil de depurar
Facilidad para añadir código	Añadir código al core es muy complicado	Añadir código es muy sencillo, solo hay que añadir las funciones nuevas
Tamaño	Es muy grande	Relativamente ligero
Despliegue e Instalación	Al ser tan grande, tarda mucho	Rápido
Curva de aprendizaje	Muy lenta	Muy rápida
Interfaz de usuario	Refinada y cercana al usuario	Bastante pobre

Tabla 3.2: Comparativa del controlador ONOS con el controlador Ryu

Después de analizar las fortalezas y debilidades de cada opción, ver tabla 3.2, hemos llegado a una decisión sobre qué controlador utilizar en nuestro proyecto. Tanto ONOS como Ryu tienen características distintivas que deben ser consideradas en nuestro caso de uso. Pero finalmente se ha elegido Ryu como controlador SDN, dado que para nuestro proyecto es la opción que lleva menos tiempo de desarrollo para una prueba de concepto y que, además, una vez probado el concepto en Ryu, técnicamente es sencillo de extender a la plataforma ONOS. Es decir, no son excluyentes, porque al final se utiliza como agente de control y ambos basados en OpenFlow.

3.5. Análisis funcional de la interfaz del BOFUSS

En esta sección, exploraremos en detalle el análisis funcional de la interfaz del BOFUSS, centrándonos específicamente en los binarios que componen su arquitectura. Estos binarios, conocidos como `ofdatapath` y `ofprotocol`, desempeñan un papel fundamental en la operativa básica de este switch de espacio de usuario.

El `ofdatapath`, como su nombre sugiere, es el proceso responsable de gestionar el plano de datos en el BOFUSS. Este componente se encarga de recibir, analizar y tomar decisiones en función de los paquetes que atraviesan su pipeline de procesado de paquetes. A través del *parser*⁴, tablas de flujo, y tablas de métricas, el `ofdatapath` garantiza una transferencia de datos fluida y eficiente en el entorno OpenFlow. Por otro lado, el `ofprotocol` se ocupa del agente de control en el BOFUSS. Su función principal consiste en establecer la comunicación entre el controlador y el switch. A través del `ofprotocol`, el controlador y BOFUSS pueden intercambiar información sobre el estado del switch, enviar nuevas reglas de procesado de paquetes, o recopilar estadísticas. Este binario es quien habilita al controlador llevar a cabo una gestión centralizada y dinámica de las políticas de red, facilitando la adaptación y optimización de la infraestructura según las necesidades del entorno.

⁴En términos de programación, un parser es un componente que analiza y procesa la estructura de un lenguaje de programación o formato de datos, descomponiéndolo en sus elementos principales para facilitar su comprensión y manipulación a posteriori.

3.5.1. Binario ofprotocol

El binario de `ofprotocol` establece un canal seguro de comunicación entre el *datapath* OpenFlow y el controlador remoto. Este conecta con el plano de datos mediante Netlink o TCP, y con el controlador remoto mediante TCP o SSL, actuando de proxy entre los dos mundos. Se quiere señalar el hecho de que el binario pueda comunicarse con el *datapath* mediante TCP, dado que según el creador de la herramienta indica que esos dos binarios pueden trabajar desacoplados en máquinas diferentes y comunicarse a través de la red. Sin embargo, esta configuración no es muy común, dado que la comunicación entre *datapath* y agente de control es crítica, y no admiten ni delays, ni perdidas.

Código 3.1: Interfaz CLI del binario ofprotocol

```
1   ofprotocol [options] datapath controller[,controller...]
```

Uno de los parámetros obligatorios a la hora de invocar al agente de control, es qué *datapath* se va a gestionar. Este se puede indicar de las siguientes formas.

- `unix:file`, se indica un descriptor de un socket UNIX, el cual deber ser el mismo que se indique a la hora que ejecute el `ofdatapath`. Mediante este socket unix se comunicarán *datapath* y agente de control.
- `tcp:HOST[:PORT]`, en este caso, también se puede conectar en red mediante puerto y dirección IP. Este tipo de identificación se usa cuando se quiere tener separado en máquinas diferentes *datapath* y agente de control. El puerto que se emplea por defecto es el 6653.

En cambio, el parámetro del controlador es opcional, y solo soporta conexiones TCP. Para la conexión con el controlador se contemplan dos paradigmas diferentes para la conexión del controlador.

- `out-of-band`: con esta configuración el tráfico OpenFlow utiliza una red privada para comunicarse con el controlador.
- `in-band`: con esta configuración el tráfico OpenFlow viaja por la misma red que la red de datos. Esta opción es la opción por defecto.

Para llevar a cabo la configuración manual del control in-band, es necesario realizar algunos pasos clave con antelación. En primer lugar, se requiere especificar la ubicación precisa del controlador al llamar al binario de `ofprotocol`. Esto asegurará una conexión adecuada entre el controlador y el agente de control. Además, es crucial configurar la interfaz de red

como el puerto local OpenFlow, el cual permite que `ofprotocol` establezca una conexión efectiva con el controlador. El puerto local OpenFlow es un puerto de red virtual que actúa como puente entre los puertos físicos del switch y el controlador. Para lograr esto, se puede especificar el nombre de la interfaz de red asignada al puerto local mediante la opción `--local-port` en la línea de comandos del binario `ofdatapath`. Generalmente, si no se indica ninguna interfaz, será el propio binario quien genere una interfaz de tipo TUN/TAP con el nombre `tap0/tun0`. Siguiendo estos pasos, se puede configurar adecuadamente el control in-band.

3.5.2. Binario `ofdatapath`

La herramienta `ofdatapath` representa una valiosa implementación en el ámbito de las datapaths OpenFlow. Esta herramienta, diseñada para funcionar en el espacio de usuario, tiene la capacidad de supervisar y gestionar una o más interfaces de red de manera eficiente. Estas interfaces actúan como canales de comunicación fundamentales a través de los cuales los paquetes de datos son transmitidos y reenviados según las políticas y reglas establecidas en las tablas de flujos (Ir a Sección 2.5.2). Gracias a esta funcionalidad, `ofdatapath` permite un control efectivo y granular a nivel de flujo de datos en la red, asegurando un enrutamiento adecuado y optimizado de los paquetes. Su flexibilidad y adaptabilidad a diferentes escenarios hacen de esta herramienta una elección preferida en entornos OpenFlow a la hora de hacer pruebas de concepto, donde se busca una implementación amigable y funcional de un agente OpenFlow. A continuación en el bloque de código 3.2 se indica la interfaz de comandos de este binario.

Código 3.2: Interfaz CLI del binario `ofdatapath`

```
1   ofprotocol [options] datapath controller[,controller...]
```

La combinación del binario `ofdatapath` junto con el binario `ofprotocol` da lugar a lo que se conoce como BOFUSS (Basic OpenFlow Software Switch), una solución integral de software switch SDN OpenFlow. Al utilizar BOFUSS, se obtiene un control y gestión a bajo nivel de las interfaces de red, lo que permite una administración eficiente de los flujos de datos que atraviesan la pipeline de procesamiento del switch. Es importante destacar que, para acceder a estas interfaces de red, el binario generalmente requiere ejecutarse con privilegios de super usuario. Además, es relevante considerar la forma en la que estos binarios se comunican entre sí. Por lo general, esta comunicación se establece a través de un socket UNIX, permitiendo una conexión directa y eficiente entre ambos componentes. Sin embargo, también es posible establecer una conexión pasiva mediante TCP, ofreciendo una alternativa para la comunicación entre los binarios. Esta flexibilidad en los mecanismos de

comunicación brinda opciones adaptativas y versátiles, permitiendo que BOFUSS se adapte a diferentes entornos y necesidades.

A continuación, se indican algunos de los parámetros más importantes de la interfaz CLI del binario `ofdatapath`. Empezando por el único de ellos obligatorio, que es, cómo indicamos el punto de comunicación del plano de datos hacia el exterior, véase un agente de control, como por ejemplo el binario `ofprotocol`.

- `punix:file`, escucha por una conexión en el descriptor del socket UNIX indicado.
- `ptcp:[port]`, escucha por conexiones TCP en el puerto determinado. Según la wiki de la herramienta, indican que el puerto por defecto es el 975, lo que en la práctica no es verdad⁵, es el 6653.

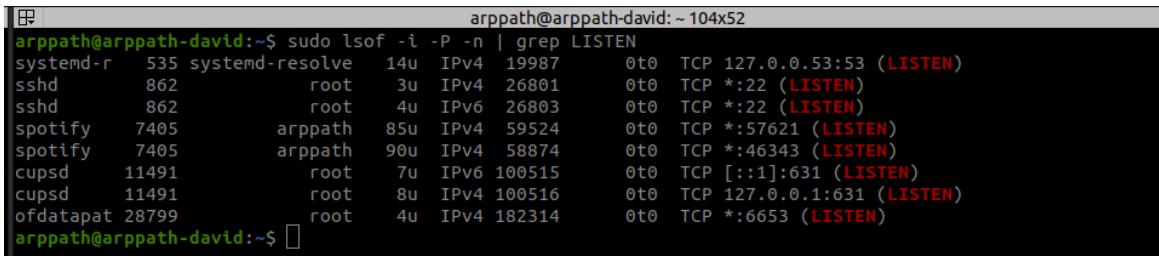
El valor del puerto por defecto se puede comprobar fácilmente, a continuación en las figuras 3.4 y 3.5. Como se puede ver se lanza el binario de forma *standalone* sobre la interfaz de loopback del sistema, y si comprobamos con la herramienta `lsof` los puertos TCP en esto de escucha en la Network namespace por defecto, se puede ver como el binario está utilizando el puerto 6653. Pero se puede ir un paso más allá, podemos ir al propio código fuente de la herramienta, y ver en qué macro definen el puerto por defecto, lo cual se puede comprobar en el repositorio de código de BOFUSS⁶.

```
arppath@arppath-david:~$ sudo ofdatapath ptcp:localhost
[...]
```

Figura 3.4: Ejecución del binario `ofdatapath` en modo standalone

⁵Se tuvo que modificar la wiki de la herramienta https://github.com/CPqD/ofsoftswitch13/wiki/Ofdatapath-Manual_history

⁶<https://github.com/CPqD/ofsoftswitch13/blob/master/include/openflow/openflow.h#LL75C1-L75C27>



```
arppath@arppath-david:~$ sudo lsof -i -P -n | grep LISTEN
systemd-r 535 systemd-resolve 14u IPv4 19987      0t0  TCP 127.0.0.53:53 (LISTEN)
sshd      862          root  3u  IPv4 26801      0t0  TCP *:22 (LISTEN)
sshd      862          root  4u  IPv6 26803      0t0  TCP *:22 (LISTEN)
spotify   7405         arppath 85u  IPv4 59524      0t0  TCP *:57621 (LISTEN)
spotify   7405         arppath 90u  IPv4 58874      0t0  TCP *:46343 (LISTEN)
cupsd    11491         root  7u  IPv6 100515     0t0  TCP [::1]:631 (LISTEN)
cupsd    11491         root  8u  IPv4 100516     0t0  TCP 127.0.0.1:631 (LISTEN)
ofdatapath 28799        root  4u  IPv4 182314     0t0  TCP *:6653 (LISTEN)
arppath@arppath-david:~$
```

Figura 3.5: Comprobación del puerto de escucha del binario `ofdatapath`

Continuando con los parámetros de configuración del software switch, uno de los más importantes permite indicar los puertos a gestionar el switch, es decir, qué interfaces va a manejar.

- `-i, --interfaces=netdev[,netdev]`, con este comando indicamos cada puerto que tendrá el switch. Cada interfaz se le asignará un número de puerto. Otro detalle a tener en cuenta es que las interfaces no pueden tener IPs.
- `-L, --local-port=netdev`, con este comando indicamos el puerto local que tendrá el switch el cual será la interfaz física o virtual, que se usará para el control in-band. Cuando esta opción no está indicada, por defecto, se creará una interfaz de tipo tap, con nombre `tap0` o similar, la cual se utilizará para el control del software switch. Si no se quiere dejar como responsabilidad al Kernel la de asignar un nombre a la interfaz tap, se puede indicar como `--local-port=tap:name`. Para más información sobre las interfaces tun/tap, ver la Sección 2.6.1.
- `--no-local-port`, se le indica al software switch que no va a utilizar un puerto local, ergo, no podremos trabajar en modo in-band.
- `--no-slicing`, se utiliza para deshabilitar la configuración de las colas asociadas a los puertos. Por ello, contendrá un total de 0 colas. Esta opción se suele utilizar cuando algunas de las configuraciones de colas (tc y kernel) no se encuentran disponibles. (Mininet y Mininet-wifi corre el BOFUSS con esta opción por defecto).
- `-d, --datapath-id=dpid`, especifica el Datapath ID Openflow, conocido como dpid. Es un identificador del datapath de 16 dígitos hexadecimales. Si no se especifica, el `ofdatapath` pilla uno aleatorio.

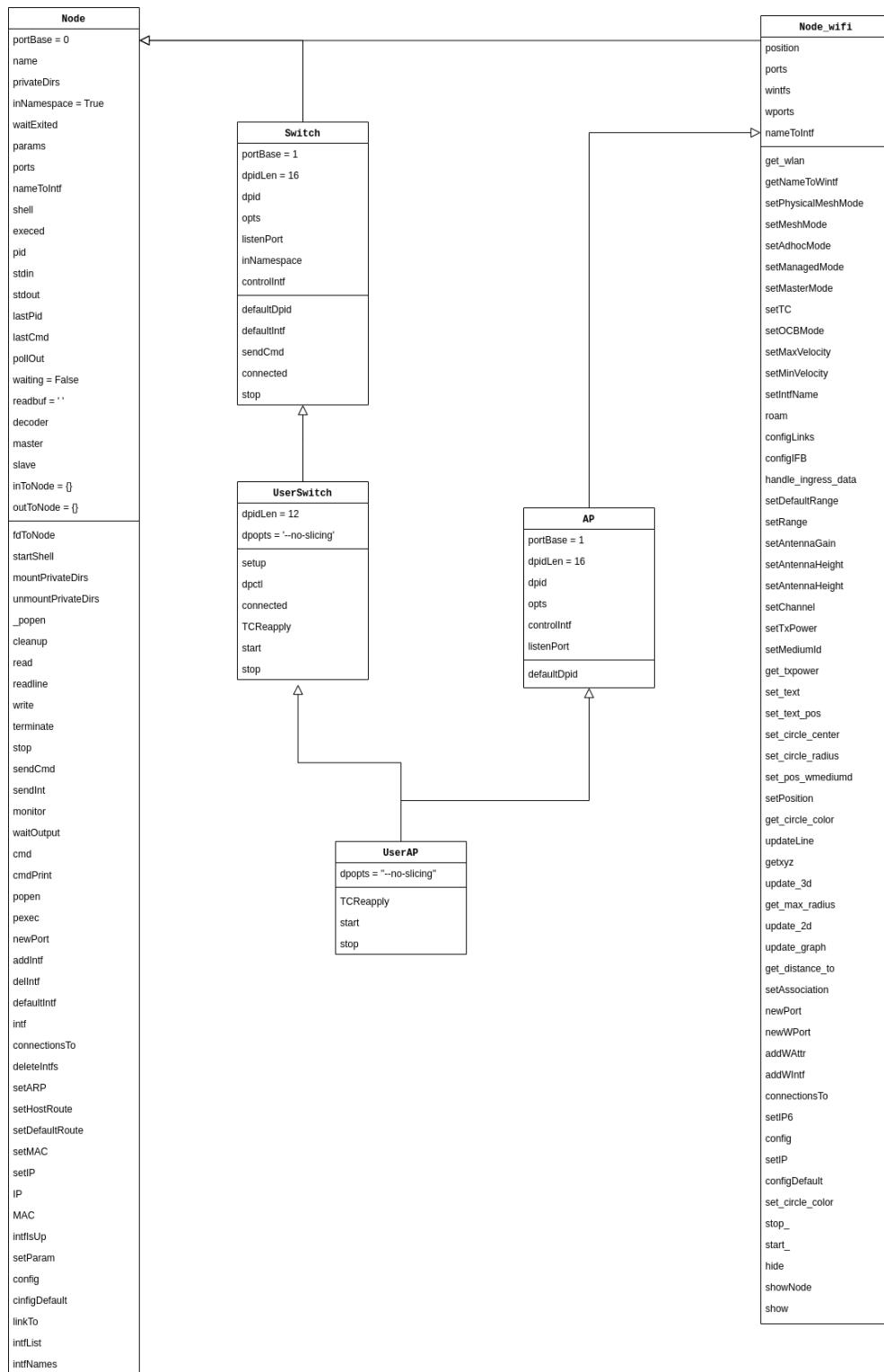
3.6. Análisis de la clase UserAP en Mininet-WiFi

En esta sección, vamos a sumergirnos en un análisis exhaustivo de la clase `UserAP` en Mininet-WiFi, una pieza fundamental que envuelve al BOFUSS. Al observar detenidamente el diagrama UML de clases en la Figura 3.6, nos encontramos con una intrigante jerarquía de clases relacionadas con `UserAP`. En el centro de esta estructura se encuentran las clases primigenias, `Node` y `Node_wifi`, que se llevan la mayor carga lógica al albergar la mayoría de atributos y métodos esenciales. Estas clases primigenias desempeñan un papel crucial al gestionar una serie de operaciones vitales. Entre sus responsabilidades se encuentran la creación de *Network namespaces*, la configuración y creación de interfaces inalámbricas, el manejo del TC para establecer los atributos de los enlaces y mucho más. Son el núcleo de la implementación que permite el funcionamiento armonioso de la infraestructura.

No obstante, es importante destacar que la clase `UserAP`, encargada de encapsular al BOFUSS, también aporta su propia lógica especializada. Su tarea principal radica en la gestión del proceso `HostAPd`, un daemon que trabaja incansablemente para materializar las diversas funcionalidades de punto de acceso. Este componente es fundamental para dotar de vida y dinamismo a la red inalámbrica emulada. Además de su papel esencial en el despliegue del BOFUSS, estas clases tienen una responsabilidad adicional: implementar una interfaz de ejecución que permite adaptar las condiciones del escenario a los parámetros necesarios de la interfaz de línea de comandos del software switch SDN. Esta adaptabilidad se convierte en una ventaja estratégica, ya que proporciona la flexibilidad necesaria para personalizar y ajustar el entorno según las necesidades específicas de cada caso de uso.

Cabe mencionar que Mininet-Wifi redefine atributos que ya se encuentran en Mininet, como por ejemplo la longitud del identificador del *datapath*. Muchos de los *bugs*⁷ encontrados entre los repositorios de las plataformas de emulación y el BOFUSS, se deben a incoherencias en la definición de las interfaces y a redefiniciones de parámetros como se ha podido encontrar. Por ello, para ver a bajo nivel cómo se ejecuta los binarios pertenecientes al BOFUSS, se va a hacer una prueba de concepto lanzando una topología sencilla, y se va a estudiar las trazas de ejecución del mismo. Esto nos será de utilidad para poder comprender qué comandos y llamadas al sistema se llevan a cabo para levantar una instancia de un software switch BOFUSS.

⁷En términos de programación, un “bug” es un error o fallo en el código de un programa que provoca un comportamiento inesperado o incorrecto en la aplicación.

Figura 3.6: Diagrama UML de la clase *UserAP*

La topología que se va a desplegar se puede apreciar en la Figura 3.7. Para lanzar dicha topología se tiene que lanzar un script de Python el cual se puede encontrar en el repositorio del TFM (Sección 2.9). A la par que se ejecuta el script de Python que alberga la topología, se tiene que lanzar un controlador SDN que le permita al software switch manejar correctamente los paquetes que atravesen su *datapath*. A continuación, en el bloque de código 3.3, se puede apreciar qué comandos se tienen que utilizar para desplegar el escenario.

Código 3.3: Puesta en marcha del escenario básico

```

1 # Lanzamos el script que pone en marcha el medio inalámbrico via Mininet-WiFi
2 sudo python3 topo.py
3
4 # Lanzamos el controlador (en otra terminal)
5 ryu-manager ryu.app.simple_switch_13

```

Algún lector podría preguntarse en este punto como va a llevarse a cabo la comunicación entre el controlador SDN y el BOFUSS. Dicha comunicación se producirá a través de la interfaz de red de *loopback* de la Network namespace por defecto, donde el controlador Ryu estará escuchando en el puerto 6633, con una interfaz virtual de tipo *tun* generada por el BOFUSS para llevar a cabo la conexión OpenFlow. Para recolectar información sobre la traza de ejecución del script en Mininet-WiFi se debe poner el nivel de log a *debug*.

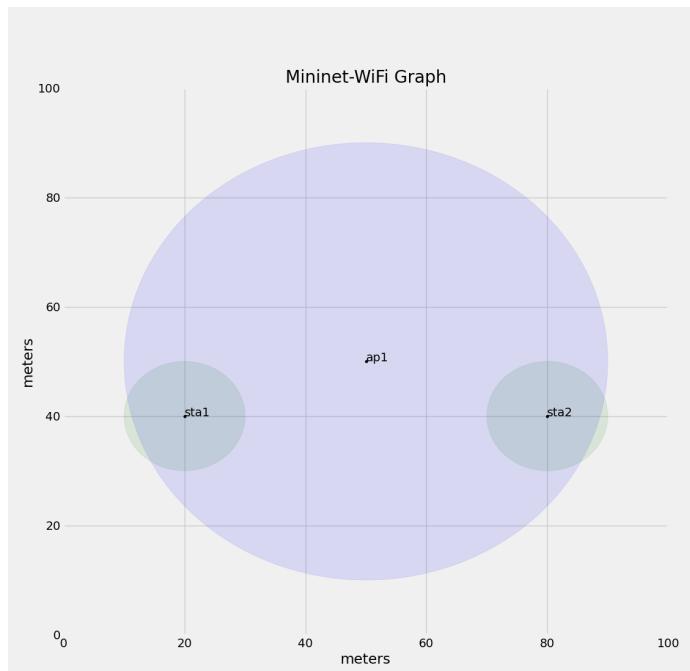


Figura 3.7: Topología básica haciendo uso del UserAP (BOFUSS)

A continuación, en el bloque de código 3.4, se puede apreciar la traza de ejecución de la topología básica. Dicha traza se ha limpiado y se han marcado las partes más importantes. Como se indicó anteriormente, esta traza es muy enriquecedora dado que nos permitirá a posteriori hacer nuestros propios escenarios a medida con topologías inalámbricas a bajo nivel. A lo largo de la traza, se podrán encontrar comentarios en verde que indican que operativas se están llevando a cabo en qué parte.

Código 3.4: Traza de la puesta en marcha del escenario básico

```

1  ### Primero se comprueba las características del sistema sobre donde va a correr
2  *** errRun: ['grep', '-c', 'processor', '/proc/cpuinfo']
3  4
4  0*** Setting resource limits
5  *** Creating nodes
6  *** Add Controller (Ryu) ***
7  *** errRun: ['which', 'mnexec']
8  /usr/bin/mnexec
9  0*** errRun: ['which', 'ifconfig']
10 /usr/sbin/ifconfig
11 0_popen ['mnexec', '-cd', 'env', 'PS1=\x7f', 'bash', '--norc', '--noediting', '-is', 'mininet:←
    ↪ c0'] 359891*** c0 : ('unset HISTFILE; stty -echo; set +m',)
12 unset HISTFILE; stty -echo; set +m
13
14  ### Se comprueba la conectividad con el controlador al puerto por defecto haciéndole un telnet
15 *** c0 : ('echo A | telnet -e A localhost 6633',)
16 Telnet escape character is 'A'.
17 Trying 127.0.0.1...
18 Connected to localhost.
19 Escape character is 'A'.
20
21 telnet> Connection closed.
22 *** Add one UserAP ***
23 *** errRun: ['which', 'mnexec']
24 /usr/bin/mnexec
25 0*** errRun: ['which', 'ip', 'addr']
26 /usr/sbin/ip
27 1_popen ['mnexec', '-cd', 'env', 'PS1=\x7f', 'bash', '--norc', '--noediting', '-is', 'mininet:←
    ↪ ap1'] 359897*** ap1 : ('unset HISTFILE; stty -echo; set +m',)
28 unset HISTFILE; stty -echo; set +m
29
30  ### Se prepara el box del AP1 (El cual correrá el BOFUSS)
31 added intf lo (0) to node ap1
32 *** ap1 : ('ifconfig', 'lo', 'up')
33 *** errRun: ['which', 'ofdatapath']
34 /usr/local/bin/ofdatapath
35 0*** errRun: ['which', 'ofprotocol']
36 /usr/local/bin/ofprotocol
37
38  ### Se prepara los boxes de las estaciones wifi
39 *** Add two WiFi stations ***
40 *** errRun: ['which', 'mnexec']
41 /usr/bin/mnexec

```

```

42 0*** errRun: ['which', 'ip', 'addr']
43 /usr/sbin/ip
44 _popen ['mnexec', '-cdn', 'env', 'PS1=\x7f', 'bash', '--norc', '--noediting', '-is', 'mininet'←
        ↪ :sta1'] 359904*** sta1 : ('unset HISTFILE; stty -echo; set +m',)
45 unset HISTFILE; stty -echo; set +m
46 _popen ['mnexec', '-cdn', 'env', 'PS1=\x7f', 'bash', '--norc', '--noediting', '-is', 'mininet:←
        ↪ sta2'] 359906*** sta2 : ('unset HISTFILE; stty -echo; set +m',)
47 unset HISTFILE; stty -echo; set +m
48
49 ### Se generan los radio taps emulados haciendo uso del modulo del kernel mac80211_hwsim
50 *** Configuring nodes
51 Loading 3 virtual wifi interfaces
52 Created mac80211_hwsim device with ID 0
53 Created mac80211_hwsim device with ID 1
54 Created mac80211_hwsim device with ID 2
55 rfkill unblock 17
56
57 ### Se cambian los nombres por defecto de las interfaces virtuales generadas a nombres
58 ### descriptivos que identifiquen a los nodos
59 *** sta1 : ('ip link set wlan0 down',)
60 *** sta1 : ('ip link set wlan0 name sta1-wlan0',)
61 rfkill unblock 18
62 *** sta2 : ('ip link set wlan1 down',)
63 *** sta2 : ('ip link set wlan1 name sta2-wlan0',)
64 *** ap1 : ('ip link set wlan2 down',)
65 *** ap1 : ('ip link set wlan2 name ap1-wlan1',)
66 *** ap1 : ('ip link set ap1-wlan1 up',)
67
68 ### Se configura la interfaz virtual emulada wireless de la sta1
69 added intf sta1-wlan0 (0) to node sta1
70 *** sta1 : ('ip link set', 'sta1-wlan0', 'down')
71 *** sta1 : ('ip link set', 'sta1-wlan0', 'address', '00:00:00:00:00:02')
72 *** sta1 : ('ip link set', 'sta1-wlan0', 'up')
73 *** sta1 : ('ip addr flush ', 'sta1-wlan0')
74 *** sta1 : ('ip addr add 10.0.0.1/8 brd + dev sta1-wlan0 && ip -6 addr add'←
        ↪ 2001:0:0:0:0:0:1/64 dev sta1-wlan0',)
75 *** sta1 : ('ip -6 addr flush ', 'sta1-wlan0')
76 *** sta1 : ('ip -6 addr add', '2001:0:0:0:0:0:0:1/64', 'dev', 'sta1-wlan0')
77 *** sta1 : ('ip link set lo up',)
78
79 ### Se configura la interfaz virtual emulada wireless de la sta2
80 added intf sta2-wlan0 (0) to node sta2
81 *** sta2 : ('ip link set', 'sta2-wlan0', 'down')
82 *** sta2 : ('ip link set', 'sta2-wlan0', 'address', '00:00:00:00:00:03')
83 *** sta2 : ('ip link set', 'sta2-wlan0', 'up')
84 *** sta2 : ('ip addr flush ', 'sta2-wlan0')
85 *** sta2 : ('ip addr add 10.0.0.2/8 brd + dev sta2-wlan0 && ip -6 addr add'←
        ↪ 2001:0:0:0:0:0:2/64 dev sta2-wlan0',)
86 *** sta2 : ('ip -6 addr flush ', 'sta2-wlan0')
87 *** sta2 : ('ip -6 addr add', '2001:0:0:0:0:0:0:2/64', 'dev', 'sta2-wlan0')
88 *** sta2 : ('ip link set lo up',)
89
90 ### Se configura la interfaz virtual emulada wireless de la ap1 y el proceso
91 ### de hostAPd

```

```

92     added intf api-wlan1 (1) to node ap1
93     *** ap1 : ('ip link set', 'api-wlan1', 'up')
94     *** ap1 : ('ethtool -K', <WirelessLink api-wlan1>, 'gro', 'off')
95     *** ap1 : ('ip link set', 'api-wlan1', 'down')
96     *** ap1 : ('ip link set', 'api-wlan1', 'address', '00:00:00:00:00:01')
97     *** ap1 : ('ip link set', 'api-wlan1', 'up')
98     *** ap1 : ("echo 'interface=ap1-wlan1\ndriver=nl80211\nssid=new-ssid\nwds_sta=1\nhw_mode=g'\
99             ↪ nchannel=1\ncntrl_interface=/var/run/hostapd\ncntrl_interface_group=0' > mn359884_ap1-\
100            ↪ wlan1.apconf",)
101    > > > > > *** ap1 : ('hostapd -B mn359884_ap1-wlan1.apconf ',)
102    ap1-wlan1: interface state UNINITIALIZED->ENABLED
103    ap1-wlan1: AP-ENABLED
104    *** ap1 : ('ip link set', 'ap1-wlan1', 'down')
105    *** ap1 : ('ip link set', 'ap1-wlan1', 'address', '00:00:00:00:00:01')
106    *** ap1 : ('ip link set', 'ap1-wlan1', 'up')
107
108    ### Se configura la potencia y las caracteristicas intrinsecas de los enlaces
109    _popen ['mnexec', '-da', '359897', 'tc', 'qdisc', 'replace', 'dev', 'ap1-wlan1', 'root', '\
110             ↪ handle', '2:', 'netem', 'rate', '54.0000mbit', 'latency', '1.00ms'] 360049*** ap1 : ('\
111             ↪ tc qdisc add dev ap1-wlan1 parent 2:1 handle 10: pfifo limit 1000',)
112    *** sta1 : ('iw dev', 'sta1-wlan0 set txpower fixed 1400')
113    *** sta2 : ('iw dev', 'sta2-wlan0 set txpower fixed 1400')
114    *** ap1 : ('iw dev', 'ap1-wlan1 set txpower fixed 1400')
115
116    *** Add links ***
117
118    added intf sta1-wlan0 (0) to node sta1
119    *** sta1 : ('ip link set', 'sta1-wlan0', 'up')
120    *** sta1 : ('ethtool -K', <WirelessLink sta1-wlan0>, 'gro', 'off')
121    *** executing command: tc qdisc show dev sta1-wlan0
122    *** sta1 : ('tc qdisc show dev sta1-wlan0',)
123    qdisc mq 0: root
124    qdisc fq_codel 0: parent :4 limit 10240p flows 1024 quantum 1514 target 5ms interval 100ms \
125             ↪ memory_limit 32Mb ecn drop_batch 64
126    qdisc fq_codel 0: parent :3 limit 10240p flows 1024 quantum 1514 target 5ms interval 100ms \
127             ↪ memory_limit 32Mb ecn drop_batch 64
128    qdisc fq_codel 0: parent :2 limit 10240p flows 1024 quantum 1514 target 5ms interval 100ms \
129             ↪ memory_limit 32Mb ecn drop_batch 64
130    qdisc fq_codel 0: parent :1 limit 10240p flows 1024 quantum 1514 target 5ms interval 100ms \
131             ↪ memory_limit 32Mb ecn drop_batch 64
132    at map stage w/cmds: ['%s qdisc add dev %s root handle 5:0 htb default 1', '%s class add dev %s\
133             ↪ s parent 5:0 classid 5:1 htb rate 11.000000Mbit burst 15k']
134    *** executing command: tc qdisc add dev sta1-wlan0 root handle 5:0 htb default 1
135    *** sta1 : ('tc qdisc add dev sta1-wlan0 root handle 5:0 htb default 1',)
136    *** executing command: tc class add dev sta1-wlan0 parent 5:0 classid 5:1 htb rate 11.000000\
137             ↪ Mbit burst 15k
138    *** sta1 : ('tc class add dev sta1-wlan0 parent 5:0 classid 5:1 htb rate 11.000000Mbit burst \
139             ↪ 15k',)
140    cmdns: ['%s qdisc add dev %s root handle 5:0 htb default 1', '%s class add dev %s parent 5:0 \
141             ↪ classid 5:1 htb rate 11.000000Mbit burst 15k']
142
143    outputs: ['', '']
144    _popen ['mnexec', '-da', '359904', 'iwconfig', 'sta1-wlan0', 'essid', 'new-ssid', 'ap', '\
145             ↪ 00:00:00:00:00:01'] 360059
146
147    added intf sta2-wlan0 (0) to node sta2
148    *** sta2 : ('ip link set', 'sta2-wlan0', 'up')
149    *** sta2 : ('ethtool -K', <WirelessLink sta2-wlan0>, 'gro', 'off')

```

```

133 *** executing command: tc qdisc show dev sta2-wlan0
134 *** sta2 : ('tc qdisc show dev sta2-wlan0',)
135 qdisc mq 0: root
136 qdisc fq_codel 0: parent :4 limit 10240p flows 1024 quantum 1514 target 5ms interval 100ms ↵
    ↪ memory_limit 32Mb ecn drop_batch 64
137 qdisc fq_codel 0: parent :3 limit 10240p flows 1024 quantum 1514 target 5ms interval 100ms ↵
    ↪ memory_limit 32Mb ecn drop_batch 64
138 qdisc fq_codel 0: parent :2 limit 10240p flows 1024 quantum 1514 target 5ms interval 100ms ↵
    ↪ memory_limit 32Mb ecn drop_batch 64
139 qdisc fq_codel 0: parent :1 limit 10240p flows 1024 quantum 1514 target 5ms interval 100ms ↵
    ↪ memory_limit 32Mb ecn drop_batch 64
140 at map stage w/cmds: ['%s qdisc add dev %s root handle 5:0 htb default 1', '%s class add dev %s ↵
    ↪ parent 5:0 classid 5:1 htb rate 11.000000Mbit burst 15k']
141 *** executing command: tc qdisc add dev sta2-wlan0 root handle 5:0 htb default 1
142 *** sta2 : ('tc qdisc add dev sta2-wlan0 root handle 5:0 htb default 1',)
143 *** executing command: tc class add dev sta2-wlan0 parent 5:0 classid 5:1 htb rate 11.000000Mbit ↵
    ↪ burst 15k
144 *** sta2 : ('tc class add dev sta2-wlan0 parent 5:0 classid 5:1 htb rate 11.000000Mbit burst ↵
    ↪ 15k',)
145 cmd: ['%s qdisc add dev %s root handle 5:0 htb default 1', '%s class add dev %s parent 5:0 ↵
    ↪ classid 5:1 htb rate 11.000000Mbit burst 15k']
146 outputs: ['', '']
147 _popen ['mnexec', '-da', '359906', 'iwconfig', 'sta2-wlan0', 'essid', 'new-ssid', 'ap', ' ↵
    ↪ 00:00:00:00:00:01'] 360065
148 *** Build it ***
149 *** Configuring nodes
150
151 added intf sta1-wlan0 (0) to node sta1
152 *** sta1 : ('ip link set', 'sta1-wlan0', 'up')
153 *** sta1 : ('ethtool -K', <WirelessLink sta1-wlan0>, 'gro', 'off')
154
155 added intf sta2-wlan0 (0) to node sta2
156 *** sta2 : ('ip link set', 'sta2-wlan0', 'up')
157 *** sta2 : ('ethtool -K', <WirelessLink sta2-wlan0>, 'gro', 'off')
158 *** Start the controller ***
159 *** Set controllers ***
160
161 ### AQUÍ se lanza finalmente el BOFUSS
162 *** ap1 : ('ofdatapath -i ap1-wlan1 punix:/tmp/ap1 -d 1000000000001 --no-slicing 1> /tmp/ap1-< ↵
    ↪ ofd.log 2> /tmp/ap1-ofd.log &',)
163 [1] 360070
164 *** ap1 : ('ofprotocol unix:/tmp/ap1 tcp:localhost:6633 --fail=closed --listen=punix:/tmp/ap1-< ↵
    ↪ listen 1> /tmp/ap1-ofp.log 2>/tmp/ap1-ofp.log &',)
165 *** RUN Mininet-Wifis CLI ***
166 *** Starting CLI:
167 *** errRun: ['stty', 'echo', 'sane', 'intr', '^C']

```

De esta traza se quiere destacar, aparte de la gestión que lleva a cabo con las interfaces inalámbricas que se ha ido comentando a lo largo de la traza, es la ejecución de los binarios pertenecientes al software switch BOFUSS, `ofdatapath` y el `ofprotocol`. A continuación en el bloque 3.5, se dejan las líneas extraídas del bloque anterior.

Código 3.5: Puesta en marcha del BOFUSS

```

1 # Binario del datapath
2 ofdatapath -i ap1-wlan1 punix:/tmp/ap1 -d 100000000001 --no-slicing 1> /tmp/ap1-ofd.log 2> /←
   ↪ tmp/ap1-ofd.log
3
4 # Binario del agente de control
5 ofprotocol unix:/tmp/ap1 tcp:localhost:6633 --fail=closed --listen=punix:/tmp/ap1.listen 1> /←
   ↪ tmp/ap1-ofp.log 2>/tmp/ap1-ofp.log

```

Según se ha analizado en la sección anterior 3.5 donde se ha estudiado la interfaz CLI del BOFUSS, podemos llegar a entender cada parámetro que Mininet-WiFi, mediante la clase `UserAP`, ha conseguido traducir del script de la topología básica a la llamada de los dos binarios del software switch. De las dos llamadas a cada binario, se quiere mencionar que Mininet-WiFi por defecto suele mandar los logs del plano de datos y del agente de control al directorio temporal (`/tmp/`) de la distribución linux en cuestión, pero no solo los archivos de logs, también se ubican ahí los descriptores de archivos UNIX para intercomunicar `ofdatapath` y el `ofprotocol`.

3.7. Análisis del entorno de depuración del BOFUSS

En esta sección, exploraremos el proceso de depuración del BOFUSS utilizando Visual Studio Code (VS Code) y los conocimientos adquiridos sobre el funcionamiento de la interfaz de línea de comandos del BOFUSS en Mininet-WiFi (Ver Sección 3.5). El objetivo es comprender en detalle cómo se ejecutan los comandos y qué sucede internamente durante la operación del BOFUSS en un entorno de red inalámbrica emulada. En nuestro escenario, trabajaremos con Mininet-WiFi, que nos proporciona un entorno virtualizado para la emulación de redes inalámbricas gracias al modulo del Kernel `mac80211_hwsim`.

Por ello, trabajaremos en estrecha colaboración con Mininet-WiFi para llevar a cabo la depuración. Sin embargo, este enfoque puede presentar cierta complejidad, por lo que realizaremos una primera aproximación ejecutando el código de una topología sencilla en modo de depuración, lo que nos permitirá observar los comandos que se ejecutan y comprender su funcionamiento. Posteriormente, exploraremos cómo convertir estos comandos en scripts de shell para mayor conveniencia y automatización. Nuestras herramientas de trabajo serán las siguientes:

- **Visual Studio Code (VS Code):** Utilizaremos este editor para escribir y editar el código, así como para realizar la depuración paso a paso. VS Code proporciona una interfaz intuitiva y funciones avanzadas de depuración que nos facilitarán el proceso.

A parte de tener una maravillosa terminal integrada y una interfaz con GDB ya desarrollada.

- **Mininet-WiFi:** Esta herramienta nos permitirá emular redes inalámbricas. Trabajaremos con una topología específica, la cual ya hemos mencionado en la sección anterior, y la ejecutaremos en modo de depuración para comprender mejor su funcionamiento, para así poder extraer las líneas de comandos necesarias para replicar su funcionamiento de forma completamente externa.
- **GDB:** Utilizaremos el depurador GDB para analizar y depurar el código del BOFUSS. GDB nos permitirá examinar el estado del programa en tiempo de ejecución, establecer puntos de interrupción, inspeccionar variables y ejecutar el código paso a paso, lo que nos ayudará a identificar posibles errores y problemas en el BOFUSS.

Por tanto, vamos a resumir qué estrategia vamos a seguir para depurar al switch. Según se puede apreciar en la Figura 3.8, los pasos que se van a seguir para conseguir depurar al BOFUSS son los siguientes. Como se ha indicado, se va a utilizar la misma topología descrita en la Sección 3.6, de la cual se va a obtener la traza de ejecución de la misma. Una vez que se tiene la traza de ejecución de la misma, se desarrollan dos shell script para levantar el escenario y otro para destruirlo.

La idea detrás de esto, es que podamos externalizar el proceso de levantar la topología. Si los scripts son capaces de replicar el funcionamiento de Mininet-WiFi, pasaremos a la siguiente fase, donde se configura el depurador de C que se prefiera, en nuestro caso trabajaremos con GDB. Una vez configurado en VS Code, lanzaremos la el escenario con los scripts previamente programados, y se depurará el funcionamiento del software switch.

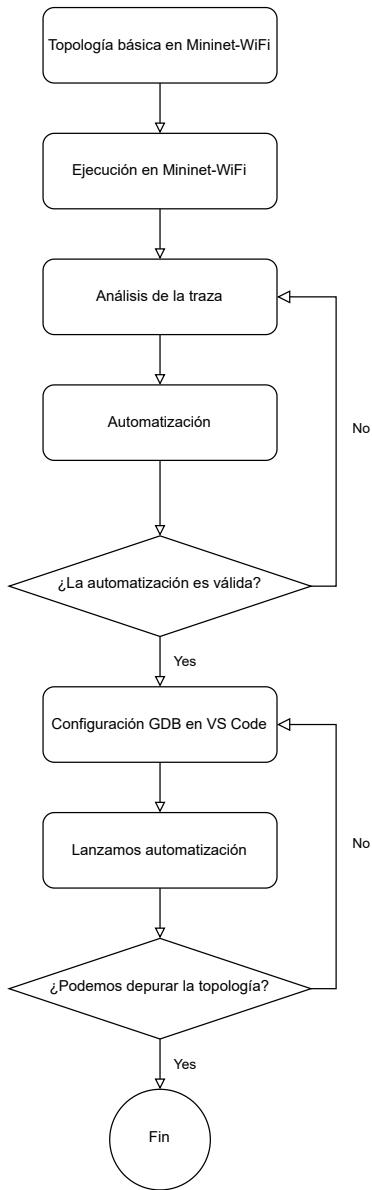


Figura 3.8: Proceso de debug al BOFUSS

3.7.1. Limpieza del escenario

La limpieza del escenario es un proceso muy importante dado que la emulación de todos los escenarios que vayamos lanzando se pueden quedar en nuestro equipo haciendo que se consuman recursos o incluso arrojando un comportamiento no esperado haciendo que las conclusiones sobre los desarrollos bajo test sean incorrectos. Para la limpieza del escenario solo hará falta lanzar el siguiente script que únicamente tiene una línea de código (Ver bloque 3.6).

Código 3.6: Script de limpieza del escenario - clean.sh

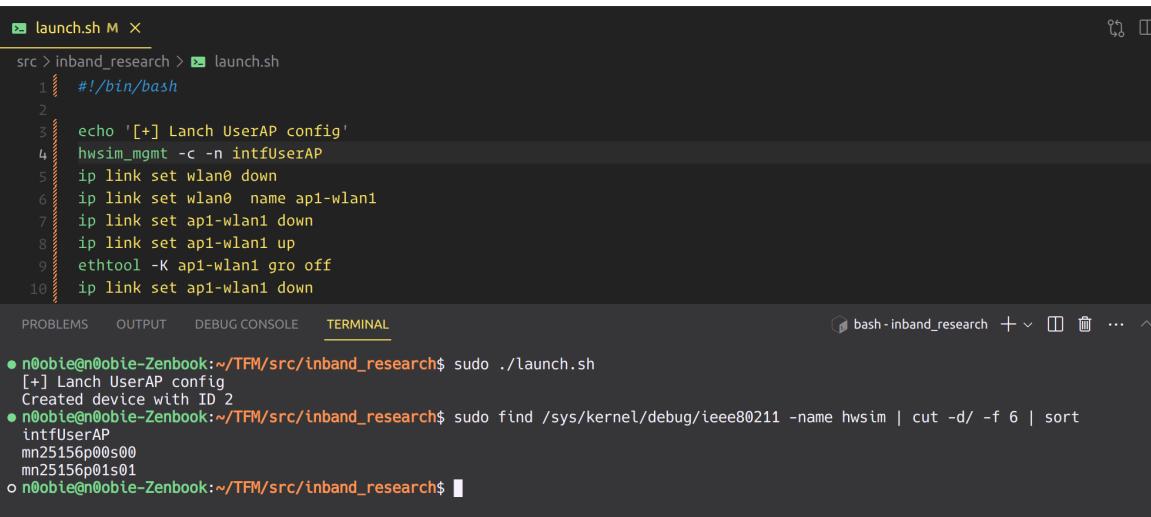
```
1 # Lanzamos el script de limpieza del propio Mininet
2 sudo mn -c
```

La simplicidad del script de limpieza es una de sus principales fortalezas. Aunque pueda parecer básico, este script ha sido probado exhaustivamente en una amplia variedad de configuraciones y topologías, demostrando su eficacia para limpiar de forma completa y agnóstica de la topología todos los componentes relacionados con la interfaz wireless.

Durante las pruebas realizadas, en una de ellas, se han ejecutado manualmente los comandos para levantar de forma individual la interfaz wireless para el punto de acceso AP1. En este proceso, se ha observado que el script de limpieza elimina correctamente todas las configuraciones previamente establecidas con nuestro shell script. ¿Cuál es el secreto detrás de esta efectividad?

Aquí es donde debemos reconocer el trabajo de Ramon Fontes y su contribución en el desarrollo de Mininet-WiFi. Al examinar el contenido⁸ de la ruta `/sys/kernel/debug/ieee80211`, se puede encontrar una lista de todas las interfaces inalámbricas emuladas cargadas en el sistema. Es gracias a esta información que el script de limpieza puede identificar y eliminar de manera precisa todas las configuraciones relacionadas con las interfaces wireless, garantizando una limpieza completa. Como se puede ver en la Figura 3.9, cuando se lanza el escenario topo.py, al listar las interfaces de la misma forma que lo hace Mininet-WiFi somos capaces de listar todas las interfaces inalámbricas emuladas del sistema, tanto las lanzadas desde Mininet-WiFi como las añadidas a mano haciendo uso de un shell script en paralelo.

⁸mininet-wifi/blob/master/mn_wifi/clean.py-L77



```

src > inband_research > launch.sh
src > inband_research > launch.sh
1 #!/bin/bash
2
3 echo '[+] Lanch UserAP config'
4 hwsim_mgmt -c -n intfUserAP
5 ip link set wlan0 down
6 ip link set wlan0 name ap1-wlan1
7 ip link set ap1-wlan1 down
8 ip link set ap1-wlan1 up
9 ethtool -K ap1-wlan1 gro off
10 ip link set ap1-wlan1 down

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
● n@obie@n@obie-Zenbook:~/TFM/src/inband_research$ sudo ./launch.sh
[+] Lanch UserAP config
Created device with ID 2
● n@obie@n@obie-Zenbook:~/TFM/src/inband_research$ sudo find /sys/kernel/debug/ieee80211 -name hwsim | cut -d/ -f 6 | sort
intfUserAP
mn25156p00s00
mn25156p01s01
● n@obie@n@obie-Zenbook:~/TFM/src/inband_research$ 

```

Figura 3.9: Listado de interfaces inalámbricas en `/sys/kernel/debug/ieee80211`

Como se puede ver, como el módulo de limpieza de Mininet-WiFi lee de una ruta desde la cual listan todas las interfaces inalámbricas emuladas, cuando lancemos dicho comando, se listará nuestra capa *phy* emulada, y por ende, será capaz de limpiarla a posteriori.

3.7.2. Puesta en marcha del escenario

Durante el desarrollo del script de puesta en marcha del escenario inalámbrico emulado, se llevó a cabo una exhaustiva investigación para comprender en detalle el funcionamiento interno de la topología básica y la clase `UserAP` en Mininet-WiFi. Esta investigación fue fundamental para identificar los comandos necesarios y garantizar el correcto funcionamiento del script.

Para lograrlo, se analizaron cuidadosamente las trazas de ejecución generadas por el software switch de `UserAP`. A través de este análisis, se pudo aislar y comprender los comandos utilizados en la creación de radios emuladas. Estas trazas proporcionaron una valiosa información sobre los pasos y procesos involucrados en la configuración de las interfaces inalámbricas. En particular, se observó que la creación de radios emuladas se gestiona mediante la herramienta `hwsim_mgmt`. Dentro del código fuente de Mininet-WiFi, este proceso se lleva a cabo en un punto específico y crítico. Este conocimiento fue esencial para extraer los comandos necesarios y adaptarlos al script de lanzamiento del `UserAP`.

El análisis de las trazas y la comprensión del funcionamiento interno de la clase `UserAP` permitieron obtener una visión clara de los pasos necesarios para configurar y establecer las interfaces inalámbricas emuladas en el escenario. Con esta información en mano, fue

posible implementar un script de lanzamiento efectivo que automatiza el proceso y asegura que todas las configuraciones sean aplicadas de manera adecuada (Ver bloque de código 3.7).

Código 3.7: Script de puesta en marcha del escenario - launch.sh

```

1  #!/bin/bash
2
3  # Vars
4  AP_SSID='new-ssid'
5  AP_MAC='00:00:00:00:00:01'
6  STA_2_CONN=('sta1' 'sta2')
7
8  # Create UserAP
9  echo '[+] Lanch UserAP config'
10 hwsim_mgmt -c -n intfUserAP
11 ip link set wlan0 down
12 ip link set wlan0 name ap1-wlan1
13 ip link set ap1-wlan1 down
14 ip link set ap1-wlan1 up
15 ethtool -K ap1-wlan1 gro off
16 ip link set ap1-wlan1 down
17 ip link set ap1-wlan1 address ${AP_MAC}
18 ip link set ap1-wlan1 up
19 iw ap1-wlan1 set txpower fixed 100
20 echo -e "interface=ap1-wlan1\ndriver=n180211\nssid=${AP_SSID}\nnwds_sta=1\nhw_mode=g\nchannel←
    ↪ =1\nctrl_interface=/var/run/hostapd\nctrl_interface_group=0" > mn43736_ap1-wlan1.←
    ↪ apconf
21 hostapd -B mn43736_ap1-wlan1.apconf
22 ip link set ap1-wlan1 down
23 ip link set ap1-wlan1 address ${AP_MAC}
24 ip link set ap1-wlan1 up
25 tc qdisc replace dev ap1-wlan1 root handle 2: netem rate 54.0000mbit latency 1.00ms
26 tc qdisc add dev ap1-wlan1 parent 2:1 handle 10: pfifo limit 1000
27 iw dev ap1-wlan1 set txpower fixed 1400
28 ofdatapath -i ap1-wlan1 punix:/tmp/ap1 -d 100000000001 --no-slicing 1> /tmp/ap1-ofd.log 2> /←
    ↪ tmp/ap1-ofd.log &
29 ofprotocol unix:/tmp/ap1 tcp:localhost:6633 --fail=closed --listen=punix:/tmp/ap1.listen 1> /←
    ↪ tmp/ap1-ofp.log 2>/tmp/ap1-ofp.log &
30
31 # Connect stations to AP
32 for sta in ${STA_2_CONN[@]}
33 do
34     echo "[+] Connecting ${sta} to UserAP"
35     PID_STA=$(ps aux | grep mininet | grep ${sta} | cut -d' ' -f7)
36     echo "[+] ${sta} - Detected pid ${PID_STA}"
37     nsenter --target ${PID_STA} --net iwconfig ${sta}-wlan0 essid ${AP_SSID} ap ${AP_MAC}
38 done

```

Como se puede ver en el bloque de código anterior, primero configuramos lo que viene a ser todos los parámetros propios de la clase `userAP`, creación de interfaces *wireless* emula-

das, configuración de red, además de crear la información requerida con el punto de acceso. Y más adelante lo que se hace es conectar las estaciones WiFi del escenario previamente levantado la red creada por el punto de acceso que se acaba de levantar, accediendo en cada Network namespace de cada estación WiFi.

Se quiere añadir un par de detalles que se cree que pueden ser de utilidad al lector en caso de que quieran replicar la depuración BOFUSS. Como se ha podido ver para la creación de una radio emulada se tiene que hacer uso del modulo del Kernel mac80211_hwsim, sin embargo, si se quiere trabajar con el módulo una vez ya insertado en el Kernel tendremos que utilizar otra herramienta. Dicha herramienta es `hwsim_mgmt`⁹, y a continuación en el bloque de código 3.8 se indican algunos ejemplos de uso de la operativa básica de la herramienta.

Código 3.8: Operativa básica de la herramienta `hwsim_mgmt`

```

1 # Crear una radio emulada
2 sudo hwsim_mgmt -c -n [phy_name]
3
4 # Para eliminarlas, se llama a la misma herramienta, de la siguiente manera
5 sudo hwsim_mgmt -x [phy_name]
```

Es necesario que para trabajar con la herramienta, `hwsim_mgmt`, que el modulo del kernel `mac80211_hwsim` esté cargado, si no lo está, no podremos crear ninguna radio nueva. En la Figura 3.10 se ilustra como podemos comprobar si el módulo está previamente cargado. En este caso, si vamos a lanzar primero el script de la topología básica en primera instancia `topo.py`, el cual ya cargará el modulo, no será necesario tener que cargarlo. Si creamos una interfaz con el modulo ya cargado podemos comprobar que se ha creado un nuevo radio de la siguiente forma (Ver Figura 3.11).

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
● n@obie@n0obie-Zenbook:~/tfm-thesis$ lsmod | grep -e 'mac80211_hwsim'
mac80211_hwsim          94208  0
mac80211           1323008  2 iwlwifi,mac80211_hwsim
cfg80211          1052672  4 iwlwifi,mac80211_hwsim,iwlwifi,mac80211
o n@obie@n0obie-Zenbook:~/tfm-thesis$
```

Figura 3.10: Comprobación de si el módulo `mac80211_hwsim` está cargado

⁹https://github.com/patgrosse/mac80211_hwsim_mgmt

```
● n0obie@n0obie-Zenbook:~/TFM/src/inband_research$ iw dev
phy#12
    Interface wlan0
        ifindex 18
        wdev 0xc000000001
        addr 02:00:00:00:02:00
        type managed
        txpower 20.00 dBm
phy#0  Unnamed/non-netdev interface
        wdev 0x2
        addr c4:bd:e5:72:ac:f0
        type P2P-device
        txpower 0.00 dBm
    Interface wlo1
        ifindex 2
        wdev 0x1
        addr c4:bd:e5:72:ac:ef
        type managed
        channel 11 (2462 MHz), width: 20 MHz, center1: 2462 MHz
        txpower 22.00 dBm
        multicast TXQ:
            qsz-byt qsz-pkt flows drops marks overlmt hashcol tx-bytes      tx-packets
            0       0       0       0       0       0       0       0       0
o n0obie@n0obie-Zenbook:~/TFM/src/inband_research$
```

Figura 3.11: Listado de *phy* inalámbricas usando el comando *iw*

3.7.2.1. Resolución de problemas encontrados

Uno de los problemas más comunes que pueden surgir durante el desarrollo del escenario inalámbrico emulado, es que en algunas ocasiones, nos podemos encontrar con que la interfaz inalámbrica se bloquea y nos muestra un mensaje de advertencia indicando lo siguiente:

Código 3.9: Bloqueo de la interfaz por RF-Kill

```
1  ~$ Operation not possible due to RF-kill
```

Este mensaje indica que la interfaz inalámbrica está bloqueada por una restricción llamada “RF-kill”. Esta restricción puede ser causada por diferentes factores, como un interrupción mal gestionada, una mala configuración del sistema operativo, o para salvaguardar los recursos de la máquina. Pero generalmente son *soft-blocked* por el Kernel, en la mayoría de los casos para auto-protegerse. Para solucionar este problema, debemos ejecutar el siguiente comando en la terminal (Ver bloque de código 3.10).

Código 3.10: desbloqueo de la interfaz por RF-Kill

```
1  rfkill unblock all
```

Este comando desbloqueará todas las interfaces inalámbricas que estén afectadas por la restricción ”RF-kill”. Una vez ejecutado el comando, la interfaz inalámbrica estará disponible y podremos establecerla en el estado UP sin problemas. Si el problema persiste después de ejecutar el comando mencionado, es recomendable verificar otros posibles problemas, como configuraciones incorrectas o conflictos en el sistema.

3.7.3. Configuración de VS Code

Para configurar Visual Studio Code y poder depurar el BOFUSS utilizando GDB, crearemos un archivo JSON con la configuración necesaria. Antes de eso, es importante destacar que hemos logrado lanzar un escenario y ejecutar un UserAP desde un script de shell. Ahora debemos crear un archivo JSON en VS Code que invoque las dos últimas líneas de ofdatapath y ofprotocol para depurar los binarios. Por lo tanto, debemos parametrizar las instrucciones (Líneas del bloque de código 3.5) en el JSON de depuración. A continuación, en el bloque 3.11, se indica el JSON de configuración para la depuración.

Código 3.11: JSON de depuración con GDB del BOFUSS

```

1  {
2      "version": "0.2.0",
3      "configurations": [
4          {
5              "name": "(ap1)ofprotocol",
6              "type": "cppdbg",
7              "request": "launch",
8              "program": "${workspaceFolder}/secchan/ofprotocol",
9              "args": [
10                  "unix:/tmp/ap1",
11                  "tcp:localhost:6653",
12                  "--fail=closed",
13                  "--listen=unix:/tmp/ap1.listen"
14              ],
15              "stopAtEntry": false,
16              "cwd": "${workspaceFolder}",
17              "environment": [],
18              "externalConsole": false,
19              "MIMode": "gdb",
20              "setupCommands": [
21                  {
22                      "text": "target-run",
23                      "description": "Ofprotocol",
24                      "ignoreFailures": true
25                  }
26              ]
27          },
28          {
29              "name": "(ap1)ofdatapath",
30              "type": "cppdbg",
31              "request": "launch",
32              "program": "${workspaceFolder}/udatapath/ofdatapath",
33              "args": [
34                  "-i",
35                  "ap1-wlan1",
36                  "unix:/tmp/ap1",
37                  "-d",
38                  "000000000001",
39                  "--no-slicing"
40              ],

```

```

41         "stopAtEntry": false,
42         "cwd": "${workspaceFolder}",
43         "environment": [],
44         "externalConsole": false,
45         "MIMode": "gdb",
46         "setupCommands": [
47             {
48                 "text": "target-run",
49                 "description": "Ofdatapath",
50                 "ignoreFailures": true
51             }
52         ]
53     },
54     "compounds": [
55         {
56             "name": "(ap1)ofprotocol/(ap1)ofdatapath",
57             "configurations": [
58                 "(ap1)ofprotocol",
59                 "(ap1)ofdatapath"
60             ],
61             "preLaunchTask": "${defaultBuildTask}",
62             "stopAll": true
63         }
64     ]
65 }
66 }
```

Es importante mencionar que GDB no admite la ejecución con privilegios de root directamente. Para solucionar este problema, es necesario realizar un ajuste adicional¹⁰¹¹.

¹⁰<https://github.com/microsoft/vscode-cmake-tools/issues/2463>

¹¹<https://github.com/microsoft/vscode-cpptools/issues/861>

4. Implementación y evaluación del protocolo

En este capítulo, se abordará el desarrollo del protocolo en la plataforma seleccionada, destacando las partes más relevantes del proceso. Además, se realizará una evaluación para analizar su funcionamiento de manera funcional. El objetivo principal de este capítulo es presentar el resultado del desarrollo realizado, mostrando cómo se ha implementado el protocolo en el entorno seleccionado para el proyecto. Se destacarán aquellas etapas del desarrollo que se consideran de mayor importancia debido a su impacto en el rendimiento y la efectividad del protocolo.

Se comenzará describiendo detalladamente la arquitectura y los componentes del protocolo, explicando su funcionamiento en el contexto de la plataforma elegida. Se hará especial hincapié en los aspectos clave que diferencian al protocolo y que lo hacen una solución innovadora en el ámbito del control in-band para entornos de IoT. A continuación, se presentará el proceso de implementación, resaltando los desafíos y decisiones clave que se han enfrentado durante el desarrollo. Se discutirán las técnicas y algoritmos utilizados para lograr la funcionalidad deseada, asegurando una comprensión clara de la implementación realizada. Posteriormente, se llevará a cabo una evaluación del protocolo desarrollado. Esta evaluación permitirá obtener una visión clara y objetiva del rendimiento del protocolo, proporcionando una base sólida para la toma de decisiones y posibles mejoras futuras. Se presentarán los resultados obtenidos y se discutirán las implicaciones y conclusiones derivadas de ellos.

4.1. Entorno de desarrollo y validación

En esta sección se quiere resumir que entorno de trabajo se va a tener, después del análisis realizado en el capítulo anterior. La primera decisión de diseño ha sido utilizar el entorno de emulación Mininet-WiFi para el desarrollo y validación. Recordemos que esta decisión radica en la sobrecarga de trabajo que se tendría por tener que implementar un controlador SDN si se quisiera simular en Cooja.

En cuanto al software switch, se ha optado por utilizar BOFUSS en lugar de OvS. El motivo principal es que BOFUSS trabaja en espacio de usuario, mientras que OvS trabaja en espacio de Kernel. Aunque OvS tiene un rendimiento superior gracias a que parte de su

switch opera en espacio de Kernel, BOFUSS es más accesible para depuración y permite añadir nuevas funcionalidades de manera más sencilla. Además, OvS cuenta con una comunidad de desarrolladores más amplia y una gran cantidad de pruebas, lo que lo hace más sólido y popular. Sin embargo, BOFUSS, aunque cuenta únicamente con un mantenedor y carece de un sistema de pruebas establecido, se considera la opción más adecuada para este proyecto debido a su implementación preliminar de in-band, la cual es requerida en nuestro caso de uso específico.

En cuanto al controlador, se ha evaluado la opción de utilizar ONOS en comparación con Ryu. ONOS destaca por su potencia y rendimiento, y es ampliamente utilizado por los operadores de red. Sin embargo, Ryu presenta una menor carga y es más fácil de depurar y ampliar con nuevas funcionalidades si es necesario. Ryu también se despliega e instala más rápidamente debido a su menor tamaño y tiene una curva de aprendizaje más pequeña en comparación con ONOS. Aunque ONOS cuenta con una aplicación de descubrimiento topológico con una interfaz web llamativa, esta aplicación utiliza el protocolo LLDP, que no permite descubrir la configuración de enlaces inalámbricos, lo cual es fundamental para nuestro proyecto. Implementar un nuevo protocolo de descubrimiento excedería los objetivos temporales y de alcance establecidos.

En resumen, ver Tabla 4.1, se ha elegido utilizar Mininet-WiFi para la emulación de redes inalámbricas, BOFUSS como el software switch SDN debido a su implementación preliminar de in-band, y el controlador Ryu debido a su menor carga, facilidad de depuración y ampliación, y una curva de aprendizaje más pequeña.

Paradigma	Plataforma	Agente SDN	Agente de control SDN
Emulación	Mininet-WiFi (mac80211_hwsim)	BOFUSS	Ryu

Tabla 4.1: Resumen del entorno de desarrollo y validación

4.2. Control in-band en el BOFUSS

En esta sección se quiere dar el contexto necesario sobre la implementación preliminar que había de control in-band, llevada a cabo por Boby N. Constantin [78]. En adelante, nos referiremos a dicha implementación como **in-BOFUSS**, del inglés *in-band Basic OpenFlow Userspace Software Switch*. Dado que se identificó que parte de esta implementación podría ser re-aprovechada en este TFM, se inició un proceso de análisis. En primer lugar, se han detectado todas las modificaciones realizadas anteriormente en la implementación del control in-band llevada , y posteriormente se han clasificado en tres bloques funcionales: (1)

Modificaciones del puerto local del ofprotocol, (2) Implementación de la lógica de control del modo in-band, (3) Implementación del protocolo Amaru.

Sin profundizar en detalles, Amaru es un protocolo que habilita el control in-band en entornos SDN. Su principal enfoque es ofrecer un despliegue automático de alta velocidad y múltiples rutas para mejorar la resiliencia del entorno, al tiempo que minimiza la cantidad de entradas requeridas en las tablas para evitar problemas de escalabilidad [79]. Para lograr estas ventajas, Amaru explora todas las posibles rutas entre cada dispositivo de red y el nodo raíz, que es el nodo directamente conectado al controlador SDN. La distinción entre Amaru y el protocolo IoTorii radica en los identificadores utilizados. En el caso de Amaru, se emplean las AMACs asociadas por puerto del switch, mientras que en IoTorii se utilizan HLMACs asociadas mediante una relación de vecindad entre nodos en rango de cobertura. Cabe mencionar que uno de estos protocolos está diseñado para entornos SDN cableados, mientras que el otro está orientado a entornos IoT.

Una vez se han identificado y se han clasificado todas las modificaciones realizadas sobre el switch BOFUSS, se va a pasar a la siguiente etapa, en la cual se va a ir módulo por módulo analizando y decidiendo si las modificaciones de cada módulo pueden ser aprovechadas en la implementación del protocolo IoTorii. La primera modificación hacía referencia a un error en el tamaño del identificador del puerto local en el bloque de ofprotocol, teniendo que ser aplicado a 16 bits. Durante el desarrollo del *in-BOFUSS*, identificó que la lógica encargada de implementar el funcionamiento in-band del software-switch no se ejecutaba de manera adecuada. Se descubrió que este problema se debía a una discrepancia en el tamaño del identificador del puerto local, lo cual impedía que los datos correspondientes a dicho puerto fueran almacenados correctamente en la estructura utilizada por el módulo ofprotocol para almacenar los datos de los puertos del switch BOFUSS.

Una vez reaprovechado dicho módulo de modificaciones, se ha seguido por el siguiente módulo el cual hace referencia a la implementación del control in-band del *software switch*. En la implementación del *in-BOFUSS*, se detectó que el switch era incapaz de instalar reglas en las *flow tables* de forma automática en aras de mandar el tráfico de control OpenFlow por los puertos indicados. Esto se debía a que los mensajes encargados de instalar las reglas conocidos como *FLOW_MOD*, generados de forma local, estaban siendo marcados como erróneos dado que estaban siendo mal generados, y por ello, el binario *ofdatapath* no podía instalar dichas reglas.

Más específicamente, se observó que la estructura de matches no se generaba de manera correcta, lo cual impide que el módulo *ofdatapath* pueda extraer los matches e implementar

adecuadamente las reglas en su tabla de flujos. El objetivo de las modificaciones es permitir que el módulo ofprotocol sea capaz de crear mensajes **FLOW_MOD** que añadan, modifiquen o eliminen las reglas necesarias para el funcionamiento en modo in-band. Las principales modificaciones implementadas fueron las siguientes:

- Función `make_flow_mod()` del archivo `ofp.c`: Se corrigió la generación de la estructura de matches para el paquete **FLOW_MOD** y se han completado los campos requeridos de los paquetes **FLOW_MOD**. Para asegurar una generación precisa de la estructura de match, se ha implementado la función `create_ofl_match_UAH()`.
- Función `make_add_flow()` del archivo `ofp.c`: Se introdujo la creación de un paquete **FLOW_MOD** de tipo **DROP**, que instala una regla para descartar el flujo especificado en la estructura de match cuando el tamaño de las acciones es igual a 0. En caso contrario, se genera un **FLOW_MOD** normal.
- Función `make_add_simple_flow()` del archivo `ofp.c`: Se modificó para que cree un paquete **FLOW_MOD** que instale una regla para encaminar el tráfico, identificado por los campos de match, hacia un puerto específico. Si el puerto de salida es 0, se crea un **FLOW_MOD** del tipo **DROP**, es decir, una regla para descartar los paquetes correspondientes al tráfico caracterizado por los campos de match.

Realizadas las modificaciones pertinentes en los mecanismos y lograr que los mensajes **FLOW_MOD** instalaran correctamente las reglas, se evidenció un inconveniente en la lógica del bloque ofprotocol encargado de analizar los mensajes **PACKET_IN** antes de enviarlos al controlador. Dicha lógica tiene como finalidad extraer la información necesaria del paquete encapsulado en el mensaje **PACKET_IN**, con el propósito de crear las reglas que permiten configurar el control in-band. Se constató que no se extraía adecuadamente el puerto de entrada del paquete encapsulado, lo que imposibilitaba la instalación de las reglas que dependían de esta información.

A raíz de ello, se efecturaron las modificaciones necesarias para obtener de forma precisa el puerto de entrada. Consecuentemente, se determinó que era imprescindible programar una lógica que permitiera analizar los paquetes encapsulados en los mensajes **PACKET_IN**, y de esta manera, instalar las reglas necesarias para implementar el funcionamiento in-band de forma autónoma. Para alcanzar este propósito, el switch debe ser capaz de detectar el tráfico OpenFlow asociado a otros dispositivos y, en consecuencia, instalar las reglas pertinentes para encaminarlo correctamente.

Ya que en este punto, solo quedan inducir las modificaciones del protocolo IoTorii en la logica existente del control in-band del BOFUSS. Para ello, se tiene que conseguir que el

puerto de control, que va a ser siempre el mismo al tener solo una interfaz, apunte a la dirección MAC real del proximo salto de camino al nodo raíz. Dichas modificaciones se explicarán en la Sección 4.3, y en la Subsección 4.2.1 se explicará la problemática encontrada con la herramienta `dpctl` a la hora de verificar y analizar el funcionamiento del in-BOFUSS.

4.2.1. Puesta a punto de la interfaz de control del Datapath - `dpctl`

En esta subsección se va a estudiar el funcionamiento interno de la herramienta `dpctl`¹, la cual se utiliza para gestionar y controlar el plano de datos de software switch BOFUSS. La motivación de esta sección radica en el desarrollo realizado para adaptar su funcionamiento a raíz de encontrar serios problemas en su interfaz de comandos. Dichos problemas e incompatibilidades llevan registrados en varios *issues*²³⁴ desde el 2018 tanto en Mininet como en el repositorio de BOFUSS, lo que hacía ingestionable la comprobación del correcto funcionamiento del software switch modificado.

La problemática principal que se ha encontrado a la hora de hacer *debug* sobre la implementación del BOFUSS, cuando se deseaba obtener las tablas de los flujos que se deberían haber instalado en el switch, sin embargo, estas no aparecen.

```
mininet-wifi> dpctl dump-flows
*** ap1 -----
*** ap1 : ('dpctl dump-flows unix:/tmp/ap1.listen',)
dpctl: Error connecting to switch dump-flows. (Address family not supported by protocol)
dpctl: Error connecting to switch dump-flows. (Address family not supported by protocol)
mininet-wifi> 
```

Figura 4.1: Error en la interfaz de control del BOFUSS desde Mininet-WiFi

En una primera instancia, procedimos a verificar la existencia y ubicación correcta de los dos sockets UNIX mencionados en el lanzamiento del plano de datos y control del switch (ver Bloque de código 3.5). Se constató que efectivamente se generaron dichos descriptores y que disponían de los permisos adecuados. Como se puede observar en los comandos previos, se especifican dos sockets UNIX. Uno de ellos, denominado `/tmp/ap1`, se destina a la intercomunicación entre el plano de datos y el plano de control, mientras que el segundo, `/tmp/ap1.listen`, se utiliza para la gestión interna, tal como se señala en la documentación pertinente del switch. Indagando más allá, se ha tomado la decisión de llevar a cabo el lanzamiento de la herramienta desde fuera del entorno de Mininet-WiFi. Esta medida

¹<https://github.com/CPqD/ofsoftswitch13/blob/master/utilities/dpctl.c>

²<https://github.com/mininet/mininet/issues/745>

³<https://github.com/CPqD/ofsoftswitch13/issues/288>

⁴<https://github.com/mininet/mininet/issues/628>

se ha adoptado con el fin de descartar la posibilidad de que el problema surgiera a raíz de un error interno en la forma en que el emulador invoca la herramienta de control. En consecuencia, se procedió a ejecutar la herramienta directamente desde la terminal, y tal como se había presupuesto, se vió que se obtiene el mismo comportamiento (Ver Figura 4.2).

```
④ n0obie@n0obie-Zenbook:/tmp$ sudo dpctl dump-flows unix:/tmp/ap1.listens
dpctl: Error connecting to switch dump-flows. (Address family not supported by protocol)
④ n0obie@n0obie-Zenbook:/tmp$ █
```

Figura 4.2: Error en la interfaz de control del BOFUSS

Inicialmente, se asumió que la herramienta en cuestión había sido desarrollada por Eder, basándose en el hecho de que en el repositorio de BOFUSS se encuentran tanto las fuentes⁵ del binario compilado e instalado, como la documentación asociada⁶, es decir, las páginas de manual o *man pages*. Sin embargo, esta suposición resultó ser equivocada. En realidad, la herramienta es heredada de Stanford y data del año 2008. Se llegó a esta conclusión al percatarse de que se instalan las páginas de manual de la versión antigua de la herramienta, y en ellas se referencia a la implementación primigenia.

Consultando la documentación de ambas herramientas, la original mediante `man dpctl`, y la modificada, mediante `dpctl -h`, se pudo apreciar que la sintaxis era completamente diferente. A continuación, en el bloque 4.1 se indican las diferencias más notables en la CLI.

Código 4.1: Diferencias en las herramientas dpctl

```
1 # [Original] Herramienta dpctl
2 sudo dpctl CMD [SW]
3
4 # [Modificada] Herramienta dpctl
5 sudo dpctl [SW] CDM
```

Es evidente que la sintaxis ha experimentado cambios significativos. Se observa claramente cómo se han intercambiado los parámetros de comando y de identificación del switch en cuestión. Sin embargo, los cambios no se limitan solo a la reorganización de los parámetros, sino que también se han modificado los comandos disponibles en la herramienta.

Un ejemplo ilustrativo es el comando `dump-flows`, que es ampliamente reconocido y utili-

⁵<https://github.com/CPqD/ofsoftswitch13/blob/master/utilities/dpctl.c>

⁶<https://github.com/CPqD/ofsoftswitch13/blob/master/utilities/dpctl.8.in>

zado en la documentación de Mininet para recopilar información sobre los flujos instalados en la lógica interna del switch. Si examinamos el código fuente de la herramienta original, podemos confirmar que existe una función específica para proporcionar dicha funcionalidad. La función en cuestión se encuentra en el siguiente enlace:

- [dpctl-Original](#)
- [dpctl-Modificada](#)

Sin embargo, al revisar el código fuente de la versión modificada por Eder, no encontraremos una función con características similares que permita obtener una vista completa de la información de los flujos. A pesar de realizar una búsqueda exhaustiva en el código fuente modificado, no se hallará una función idéntica.

En esta situación, se presentan dos opciones: la primera consiste en utilizar la nueva interfaz de comandos proporcionada por la herramienta desarrollada por Eder. La segunda opción implica la instalación y compilación de la versión anterior de la herramienta, con el riesgo potencial de encontrar problemas o inconvenientes en el proceso.

4.2.2. Parser para la herramienta dpctl

La primera alternativa consiste en adaptarse a los cambios inducidos en la herramienta. Por lo tanto, vamos a explorar cómo extraer los flujos del switch. Después de ejecutar el comando de ayuda, `dpctl -h`, se ha determinado que la forma de realizar dicha extracción es la siguiente (Ver bloque 4.2).

Código 4.2: Extracción de flujos con la nueva versión de dpctl

```
1 sudo dpctl unix:/tmp/api.listens stats-flow
```

El inconveniente de extraer los flujos de esta manera es que la información obtenida resulta algo confusa, como se muestra en la Figura 4.3.

```
arpapp@arpapp-david:~/tfm_test/TFM/src/userAP_research$ sudo dpctl unix:/tmp/api.listens stats-flow
SENDING (xid=0x0FF00F0):
stat_req(type="flow", flags="0x0", table="all", oport="any", ogrp="any", cookie=0x0, mask=0x0, match=oam{all match})

RECEIVED (xid=0x0FF00F0):
stat_reply(type="flow", flags="0x0", stats=[{table="0", match="oam{in_port='1', eth_dst='00:00:00:00:00:02', eth_src='00:00:00:00:00:03'", dur_s=145668, dur_ns=49000000, prio=1, idle_to=0, hard_to=0, cookie=0x0", pkt_cnt=12, byte_cnt=948, insts=[apply(acts=[out(port='1')])]}, {table="0", match="oam{in_port='1', eth_dst='00:00:00:00:00:03', eth_src='00:00:00:00:00:02'", dur_s=145668, dur_ns=46000000, prio=1, idle_to=0, hard_to=0, cookie=0x0", pkt_cnt=11, byte_cnt=888, insts=[apply(acts=[out(port='1')])]}, {table="0", match="oam{all match}", dur_s=146810, dur_ns=943000000, prio=0, idle_to=0, hard_to=0, cookie=0x0", pkt_cnt=1562, byte_cnt=164147, insts=[apply(acts=[out(port='ctrl', mle=n=65535)])]}])
```

Figura 4.3: Información ofuscada de la nueva herramienta dpctl

En la figura, se visualiza el comando que se envía al socket UNIX `unix:/tmp/api.listens`,

junto con los datos recibidos. Sin embargo, como se mencionó previamente, la información se muestra de manera poco clara o comprensible. Por lo tanto, se requerirá realizar un proceso de análisis y parseo para obtener una visualización más legible de los datos.

No obstante, es relevante destacar que esta necesidad no ha sido planteada por primera vez. Un usuario de BOFUSS ya había abordado esta cuestión en su momento y realizó una petición en forma de *pull request* en el repositorio, agregando un script en Python para el análisis de la información recibida desde el switch. Dicho script, creado por un desarrollador israelí⁷, fue escrito en una versión anterior de Python (Python 2.7). Dado que el script podría mejorarse en varios aspectos dado que el JSON resultante no está correctamente formado, y las expresiones regulares no funcionaban en todos los casos. Después de implementar el parser, la salida obtenida del mismo comando se puede apreciar en la Figura 4.4.

```
n@obie@n@obie-Zenbook:~/TFM/src/userAP_research$ sudo python3 parse_stats_flow.py unix:/tmp/ap1.listens
----- Table 0 -----
[+] Match: oxm{'in_port': '1', 'eth_dst': '00:00:00:00:00:02', 'eth_src': '00:00:00:00:00:03'}
[+] Duration: 4
[+] Prio: 1
[+] Pkt cnt: 3
[+] Byte cnt: 256
[+] Insts: ['apply', {'acts': ['out', {'port': '1'}]}]
----- Table 0 -----
[+] Match: oxm{'in_port': '1', 'eth_dst': '00:00:00:00:00:03', 'eth_src': '00:00:00:00:00:02'}
[+] Duration: 4
[+] Prio: 1
[+] Pkt cnt: 2
[+] Byte cnt: 196
[+] Insts: ['apply', {'acts': ['out', {'port': '1'}]}]
----- Table 0 -----
[+] Match: oxm{all match}
[+] Duration: 60
[+] Prio: 0
[+] Pkt cnt: 47
[+] Byte cnt: 4946
[+] Insts: ['apply', {'acts': ['out', {'port': 'ctrl', 'mflen': '65535'}]}]
n@obie@n@obie-Zenbook:~/TFM/src/userAP_research$
```

Figura 4.4: Resultado del parser la nueva herramienta dpctl

4.2.3. Rollback a la herramienta dpctl

La segunda opción consiste en adaptar la herramienta antigua. Para lograr esto, se debe acceder al repositorio de la herramienta anterior llamada “dpctl” y compilarla sobre la herramienta nueva desarrollada por Eder. A continuación, se proporciona el enlace⁸ al re-

⁷<https://github.com/simhond>

⁸<https://github.com/mininet/openflow/tree/master>

positorio de la herramienta antigua. Los pasos a seguir son los siguientes, considerando una instalación limpia de Mininet, haciendo referencia al escenario utilizado⁹.

Código 4.3: Instalación de las dependencias de la nueva versión de dpctl

```

1 # Tenemos que clonar el repositorio de Openflow indicado anteriormente
2 git clone https://github.com/mininet/openflow.git
3
4 # Hacemos un install de las deps
5 sudo apt install git autotools-dev pkg-config libc6-dev

```

Una vez que se ha clonado el repositorio de OpenFlow de Standford con la versión 1.0 de OpenFlow, se deben instalar las dependencias generales necesarias para compilar el repositorio desde el código fuente. Cabe mencionar que sería más apropiado construir exclusivamente la herramienta y no todo el repositorio, incluyendo el controlador y la implementación 1.0 de OpenFlow. Sin embargo, para realizar una prueba conceptual, será suficiente. Una vez que se tienen las dependencias necesarias instaladas, podemos proceder hacer la contrucción de la herramienta. A continuación, en el bloque de Código 4.4, se indican los pasos para hacer un build desde el source:

Código 4.4: Construcción de la nueva versión de dpctl

```

1 # Entramos
2 cd openflow
3
4 # Bootteamos y configuramos
5 # (a.k.a pre-checking de que tenemos todas las herramientas para hacer el make)
6 ./boot.sh
7 ./configure
8
9 # Build :)
10 make
11
12 # E instalamos para que el binario en path de dpctl sea este ultimo
13 sudo make install

```

Una vez se ha instalado la herramienta, cuando se ejecute la ayuda del binario `dpctl -h`, se tendría que apreciar el contenido de la Figura 4.5. Para hacer una prueba de concepto, se ha levantado una topología de ejemplo con Mininet, con `userSwitches`, y se ha comprobado como ahora la herramienta de control, si es capaz de extraer los flujos sin ningún tipo de problema (Ver Figura 4.5).

⁹<https://github.com/davidcawork/TFM/tree/main/src/scenarios/mininet>

```
vagrant@mininetTest:~$ dpctl -h
dpctl: command not found
vagrant@mininetTest:~$ dpctl -h
dpctl: OpenFlow switch management utility
usage: dpctl [OPTIONS] COMMAND [ARG...]

For local datapaths only:
  adddp nl:DP_ID           add a new local datapath DP_ID
  deldp nl:DP_ID           delete local datapath DP_ID
  addif nl:DP_ID IFACE...   add each IFACE as a port on DP_ID
  delif nl:DP_ID IFACE...   delete each IFACE from DP_ID
  get-idx OF_DEV            get datapath index for OF_DEV

For local datapaths and remote switches:
  show SWITCH              show basic information
  status SWITCH [KEY]       report statistics (about KEY)
  show-protostat SWITCH    report protocol statistics
  dump-desc SWITCH          print switch description
  dump-tables SWITCH        print table stats
  mod-port SWITCH IFACE ACT modify port behavior
  dump-ports SWITCH [PORT]  print port statistics
  desc SWITCH STRING        set switch description
  dump-flows SWITCH         print all flow entries
  dump-flows SWITCH FLOW    print matching FLOWS
  dump-aggregate SWITCH     print aggregate flow statistics
  dump-aggregate SWITCH FLOW print aggregate stats for FLOWS
  add-flow SWITCH FLOW      add flow described by FLOW
  add-flows SWITCH FILE     add flows from FILE
  mod-flows SWITCH FLOW     modify actions of matching FLOWS
  del-flows SWITCH [FLOW]    delete matching FLOWS
  monitor SWITCH            print packets received from SWITCH
  execute SWITCH CMD [ARG...] execute CMD with ARGS on SWITCH
Queue Ops: Q: queue-id; P: port-id; BW: perthousand bandwidth
  add-queue SWITCH P Q [BW]  add queue (with min bandwidth)
  mod-queue SWITCH P Q BW   modify queue min bandwidth
  del-queue SWITCH P Q      delete queue
  dump-queue SWITCH [P [Q]]  show queue info

For local datapaths, remote switches, and controllers:
  probe VCONN                probe whether VCONN is up
  ping VCONN [N]               latency of N-byte echos
  benchmark VCONN N COUNT     bandwidth of COUNT N-byte echos
where each SWITCH is an active OpenFlow connection method.

Active OpenFlow connection methods:
  nl:DP_IDX                  local datapath DP_IDX
  tcp:HOST[:PORT]             PORT (default: 6633) on remote TCP HOST
  unix:FILE                   Unix domain socket named FILE
  fd:N                       File descriptor N
```

Figura 4.5: Rollback a la nueva herramienta dpctl

```
vagrant@mininetTest:~$ sudo mn --switch user
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1
*** Starting CLI:
mininet> dptcl dump-flows
*** s1 -----
stats_reply (xid=0x2ffc4215): flags=none type=1(flow)
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1.17 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.353 ms
^C
--- 10.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1006ms
rtt min/avg/max/mdev = 0.353/0.762/1.171/0.409 ms
mininet> dptcl dump-flows
*** s1 -----
stats_reply (xid=0xa6af96de): flags=none type=1(flow)
    cookie=0, duration_sec=4s, duration_nsec=159000000s, table_id=0, priority=65535, n_packets=1, n_bytes=60, idle_timeout=60, hard_timeout=0, arp,in_port=2,dl_vlan=0xffff,dl_vlan_pcp=0x00,dl_src=42:4f:b0:f5:92:a1,dl_dst=c6:0b:2d:63:85:0c,nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0x00,nw_proto=2,tp_src=0,tp_dst=0,actions=output:1
    cookie=0, duration_sec=4s, duration_nsec=158000000s, table_id=0, priority=65535, n_packets=2, n_bytes=196, idle_timeout=60, hard_timeout=0, icmp,in_port=1,dl_vlan=0xffff,dl_vlan_pcp=0x00,dl_src=c6:0b:2d:63:85:0c,dl_dst=42:4f:b0:f5:92:a1,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0x00,icmp_type=8,icmp_code=0,actions=output:2
    cookie=0, duration_sec=4s, duration_nsec=158000000s, table_id=0, priority=65535, n_packets=2, n_bytes=196, idle_timeout=60, hard_timeout=0, icmp,in_port=2,dl_vlan=0xffff,dl_vlan_pcp=0x00,dl_src=42:4f:b0:f5:92:a1,dl_dst=c6:0b:2d:63:85:0c,nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0x00,icmp_type=0,icmp_code=0,actions=output:1
mininet> 
```

Figura 4.6: Funcionamiento de la herramienta antigua sobre Mininet

4.3. Implementación del protocolo IoTorii

En esta sección se va a explicar la implementación realizada del protocolo IoTorii en el *software switch* BOFUSS. Dado que el trabajo de implementación ha sido bastante extenso, dado que había que discernir que parte de la implementación del control in-band podía ser aprovechada, se ha decidido generar un documento anexo [80] aparte de la memoria, donde se explique a bajo nivel las modificaciones realizadas. Todo el código se puede encontrar indicado en la Sección 2.9.

En la Figura 4.7, se puede apreciar la secuencia lógica que se ha llevado a cabo para la implementación del protocolo IoTorii en el software switch. En adelante se podrá encontrar el término **win-BOFUSS**, el cual hará referencia a la implementación del protocolo IoTorii en la implementación de in-band del BOFUSS, conocida como **in-BOFUSS**, por lo que para

añadirle la condición de *wireless* se ha apodado como **win-B0FUSS**, del inglés *wireless in-band Basic OpenFlow Userspace Software Switch*.

La implementación de IoTorii está basada en una publicación del grupo de investigación NetIS de la Universidad de Alcalá [71]. El funcionamiento del protocolo está descrito en la Sección 3.1, y con él conseguiremos crear múltiples caminos desde cualquier nodo de la topología al nodo raíz. La idea de utilizar este protocolo según se comentó es para tener caminos de respaldo en cada nodo hacia el nodo raíz, pudiendo comutar de uno a otro cuando sea necesario, bien sea porque un equipo ha fallado o porque al ser un entorno inalámbrico donde la movilidad es intrínseca el *next-hop* al nodo raíz ya no se encuentra en rango.

En la Figura 4.8, se indica el diagrama de flujo con la implementación del protocolo en el BOFUSS. En primer lugar, al poner el switch en el modo de control in-band, se inicia la lógica de IoTorii. Una vez que el protocolo IoTorii ha sido inicializado, el nodo raíz comienza la exploración, propagando los paquetes utilizados para generar los caminos que posteriormente serán almacenados en las tablas IoTorii de los nodos. Dichos paquetes, de tipo *SetHLMAC*, solo serán entregados a los nodos que cumplan la condición de vecinos, es decir, aquellos nodos que hayan sido notificados mediante un mensaje de tipo *Hello*. Cuando una ruta de la tabla HLMAC no se ve renovada, es decir, el nodo anexo al siguiente salto de esa ruta no ha enviado un mensaje de tipo *Hello*, esta ruta se invalidará y se saltará a la siguiente dentro de la tabla HLMAC. Cuando esto ocurre el puerto local será el mismo, dado que la interfaz será la misma, pero el *next-hop* no lo será dado que cambiará, habrá que actualizar la MAC destino en la conexión TCP que se realiza através de un mensaje netlink al stack de red del Kernel.

En un entorno Linux, es posible utilizar Netlink para modificar la dirección MAC de destino asociada a una IP específica en la tabla ARP. Netlink es una interfaz de comunicación entre el espacio de usuario y el espacio de Kernel que permite realizar diversas operaciones relacionadas con la configuración de red. Lo primero que se tiene que llevar a cabo en esta operación es abrir una conexión Netlink para comunicarse con el Kernel. Esto implica crear un socket Netlink y enlazarlo a un tipo de mensaje Netlink en específico. Acto seguido, se tiene que preparar el mensaje Netlink de tipo **RTM_NEIGH** (mensaje utilizado para manipular la caché ARP), y establecer los atributos del mensaje, configurando los atributos del mensaje para indicar la IP y la nueva dirección MAC destino a fijar. Enviar dicho mensaje y gestionar la confirmación del Kernel. De esta forma podremos actualizar de una ruta HLMAC a otra, pero lamentablemente no se ha conseguido hacer la operación de forma automática sin cerrar el socket por lo que será necesario volver a abrir el socket TCP con el controlador.

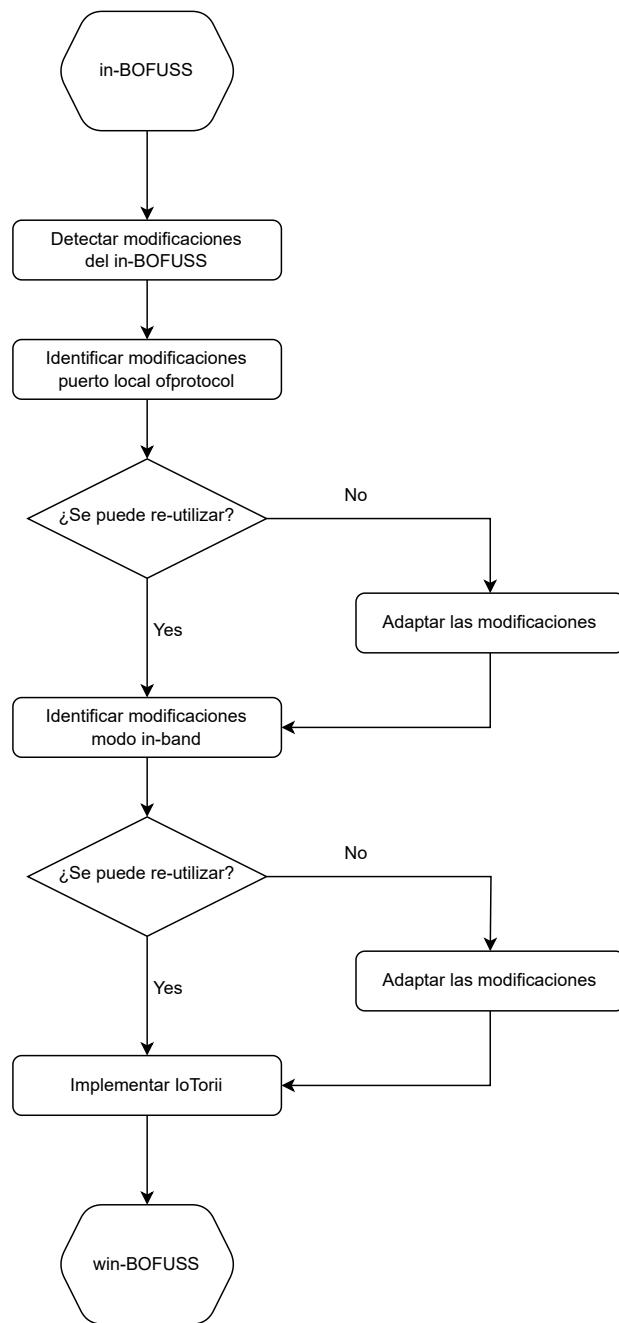


Figura 4.7: Diagrama de flujo para la implementación del protocolo IoTorii en el software switch BOFUSS

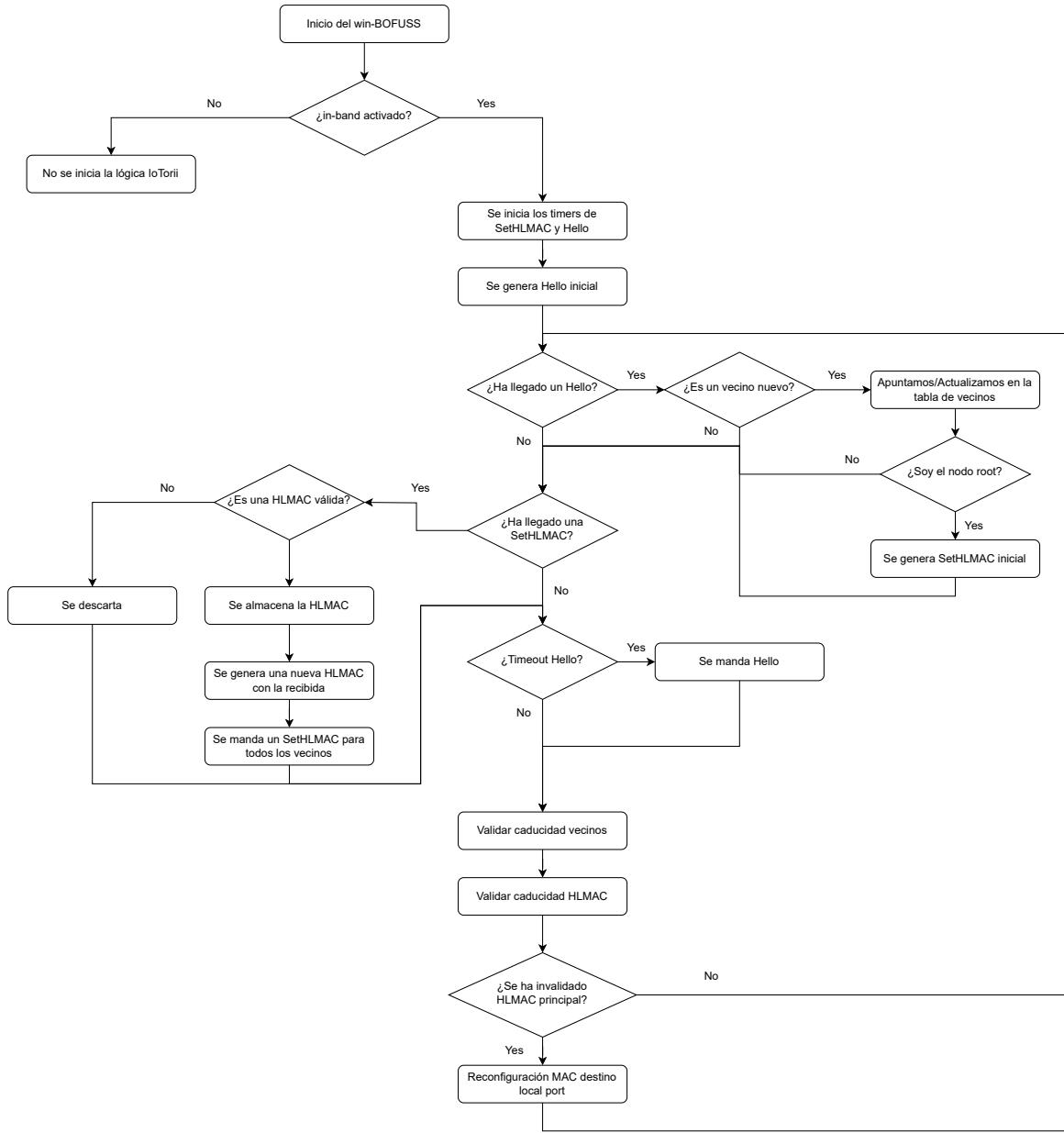


Figura 4.8: Diagrama de flujo de la operativa del protocolo IoTorii en el software switch BOFUSS

La importancia de establecer un temporizador para los mensajes *Hello* y definir la caducidad de los vecinos en un protocolo radica en la mejora de su resiliencia y capacidad para adaptarse a cambios en la topología de la red. Estos mecanismos son fundamentales para mantener la conectividad y la estabilidad del protocolo en entornos dinámicos.

El temporizador de los mensajes *Hello* determina la frecuencia con la que los nodos del protocolo intercambian información de vecindad. Estos mensajes *Hello* son vitales para conocer y mantener actualizada la lista de vecinos en la red. Al establecer un temporizador adecuado, se asegura que los nodos comparten regularmente información sobre su disponibilidad, estado y conexiones. Esto permite identificar rápidamente cambios en la topología de la red, como la caída de un enlace o la aparición de nuevos nodos. La caducidad de los vecinos define el tiempo que un nodo considera que un vecino sigue siendo alcanzable y activo. Cuando un vecino no responde a los mensajes *Hello* dentro de este período, se considera que ha caducado y se elimina de la lista de vecinos. Establecer una caducidad adecuada es esencial para detectar y reaccionar rápidamente ante la pérdida de conectividad con un vecino. Al eliminar los vecinos caducados, se evita enviar tráfico a nodos inalcanzables y se garantiza que los recursos de red se utilicen de manera eficiente.

La resiliencia de un protocolo depende en gran medida de su capacidad para recuperarse rápidamente de fallos en la red y adaptarse a cambios en la topología. Al establecer un temporizador de mensajes *Hello* y una caducidad de vecinos adecuados, el protocolo puede detectar y responder de manera oportuna a eventos como enlaces caídos, nodos inalcanzables o nuevos vecinos. Esto permite que el protocolo reconstruya rápidamente su tabla de vecinos y ajuste sus rutas en consecuencia, minimizando el impacto de las fallos y optimizando la utilización de los recursos de red.

A continuación, se van a indicar algunas de las modificaciones más importantes a la hora de implementar la lógica del protocolo IoTorii:

- La estructura `reg_HLMAC`, ha sido creada para representar una entrada de la tabla de HLMACs, también conocida como tabla de rutas de IoTorii. Cada entrada tiene almacenada la HLMAC, un flag de si está activa o no.
- La estructura `table_HLMAC`, ha sido creada para representar la tabla de HLMACs, que viene a ser una lista enlazada simple de estructuras `reg_HLMAC`.
- La estructura `reg_nb`, ha sido creada para representar una entrada de la tabla de vecinos. Cada entrada tiene almacenada la MAC real del vecino, un flag de si está activa o no, y el sufijo único asignado.

- La estructura `table_nb`, ha sido creada para representar la tabla de vecinos, que viene a ser también una lista enlazada simple de estructuras `reg_nb`.
- Se ha adaptado la función `table_AMACS_add_AMAC()` con las nuevas estructuras de datos, para que gestione el guardado de las HLMACs de IoTorii en vez de las AMACs. La función se ha renombrado a `table_HLMACS_add_HLMAC()`.
- Se ha modificado la función `dp_ports_output_amaru()`, la cual se encargaba de crear y enviar un paquete de Amaru por todas las interfaces del switch, para que ahora, dado que solo va a haber una interfaz genere los mensajes *SetHLMAC* por la interfaz por los N vecinos que tenga el nodo en cuestión. La función se ha renombrado a `dp_ports_output_iotorii()`.
- Se han adaptado las funciones `disable_invalid_amacs_UAH()` y `enable_valid_amacs_UAH()`, las cuales se utilizaban para conmutar el *flag* de active de las rutas, para que puedan trabajar con las nuevas estructuras de datos HLMAC. Las funciones se han renombrado a `disable_invalid_hlmacs_UAH()` y `enable_valid_hlmacs_UAH()` respectivamente.
- Se ha modificado la función `configure_new_local_port_amaru_UAH()`, la cual se encargaba de buscar en la tabla de rutas AMACS, para que busque en la nueva tabla HLMAC y que aparte gestione el traspaso de una ruta a otra, modificando vía Netlink la nueva MAC destino del siguiente salto. La función se ha renombrado a `configure_new_local_port_iotorii_UAH()`.
- Se ha modificado la función `send_amaru_new_localport_packet_UAH()`, para que ajuste las reglas de las tablas de flujos del ofdatapath mediante una generación de un mensaje de tipo `PACKET_IN` que se envía al ofprotocol.
- Se ha modificado la función `dp_ports_run()`, la cual de forma periódica verificará el estado del puerto, y además gestionará las funciones `disable_invalid_amacs_UAH()` y `enable_valid_amacs_UAH()` verificando la caducidad de las entradas. En caso de caducar una ruta activa se llamará a la función, `install_new_localport_rules_UAH()`, y a la función `configure_new_local_port_iotorii_UAH()` para gestionar la instalación de una ruta alternativa.
- Se han creado una lógica de control de timer para la generación de los mensajes *Hello*s. Toda la lógica está incluida en la función `timer_UAH()`.

4.3.1. Despliegue de la implementación en una RPi

Durante el proceso de desarrollo, se ha realizado un intento exhaustivo para desplegar la implementación completa del protocolo IoTorii en una Raspberry Pi (RPi). Sin embargo, hasta el momento, este intento no ha alcanzado el éxito debido a una problemática relacionada con la traducción de símbolos que se realiza en la librería de `of1lib`. Se ha identificado que esta traducción debe ser corregida para lograr un despliegue funcional en dicha plataforma, pero se escapa de los objetivos del TFM. Por otro lado, se ha observado que el despliegue del BOFUSS, una vez se ha corregido el funcionamiento interno de la librería en cuestión, debería ser inmediato, ya que el entorno Linux es el mismo, y solo se requiere adaptarse correctamente a la arquitectura. Es relevante mencionar que se ha identificado un *issue* específico que refleja esta misma problemática de despliegue sobre una arquitectura ARM, la misma que se ha visto a la hora de desplegar la implementación del protocolo IoTorii.

- Dicho *issue* puede consultarse en el siguiente enlace: <https://github.com/CPqD/ofsoftswitch13/issues/196>

4.4. Validación

En esta sección, se presenta una validación funcional, donde se han llevado a cabo pruebas sobre una topología básica para entender el correcto funcionamiento de la implementación realizada sobre el BOFUSS. Cabe destacar que las pruebas desarrolladas en este capítulo no tienen como objetivo cuantificar el rendimiento del switch BOFUSS por diversas razones. En primer lugar, dicho enfoque no ha sido establecido como objetivo principal del proyecto. En segundo lugar, el switch BOFUSS se encuentra más orientado hacia el prototipado y desarrollo de nuevas funcionalidades que a lograr un rendimiento excepcional en el procesamiento de los paquetes. En consecuencia, el enfoque de las pruebas se centra en verificar el funcionamiento adecuado de las implementaciones realizadas, sin ahondar en la medición precisa del rendimiento del *software switch*.

A continuación, en la Figura 4.9, se puede apreciar el entorno de validación que se va a utilizar. En esencia, es el mismo entorno que el que se utiliza en Mininet-WiFi, pero en este caso se va a ejecutar mediante un shell script para tener más control y conciencia de que instrucciones se van a ejecutar para desplegar el escenario. Como se puede ver en la figura, todo el entorno de validación corre sobre una única máquina que porta el kernel de Linux, el cual nos proveerá del módulo `mac80211_hwsim` de emulación del entorno inalámbrico `ieee80211` para generar las topologías inalámbricas.

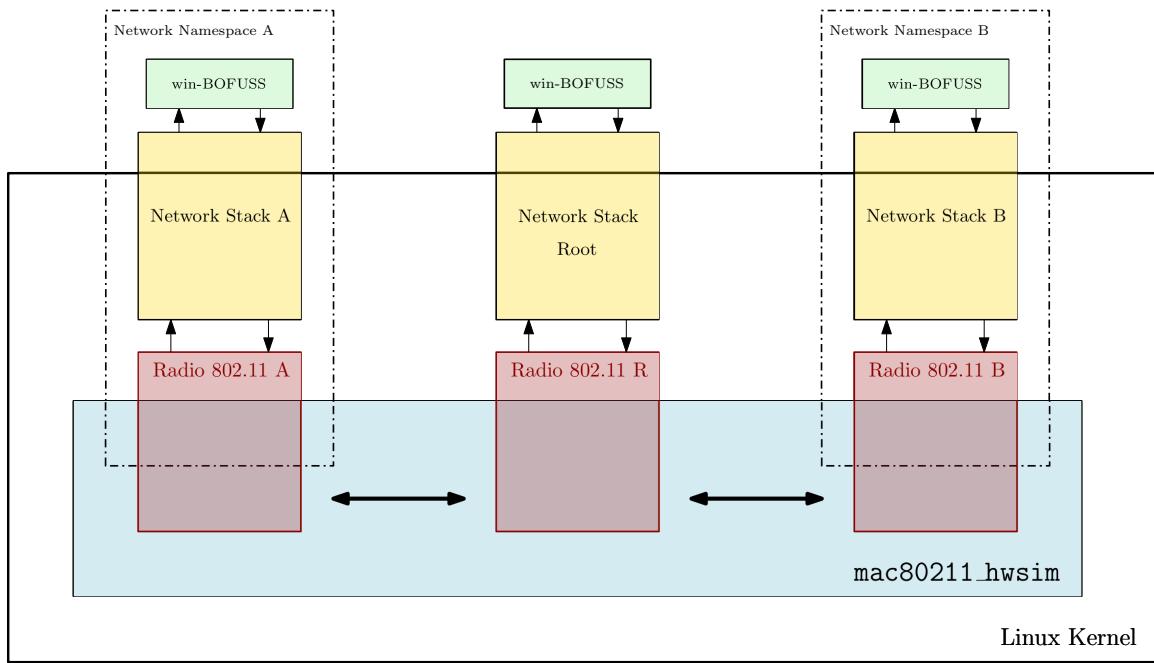


Figura 4.9: Entorno de validación haciendo uso del módulo del kernel `mac80211_hwsim`

Cada nodo de la topología se encapsulará en una Network namespace, la cual contendrá una de las radios WiFi emuladas por el módulo del kernel `mac80211_hwsim`. De esta forma se consigue que el tráfico no atraviese de un nodo al nodo raíz directamente, y siga la ruta preestablecida de la topología física emulada. Las radios internamente se verán entre ellas dentro del módulo del kernel, donde se intercambiarán los mensajes radio, los cuales serán delegados al procesado del *stack* de red que corresponda, para finalmente, ser entregado al programa de espacio de usuario `win-BOFUSS`¹⁰.

De esta forma, se podrán replicar escenarios inalámbricos con relativa rapidez, y de una forma sencilla, dado que al utilizar los mismos recursos que Mininet y Mininet-WiFi, se pueden utilizar las herramientas auxiliares a estos para interactuar con el escenario lanzado con el shell script. Se quiere mencionar, que aunque el controlador no aparezca en la Figura 4.9, este puede correr en cualquiera de las Network namespace que haya en la topología. En nuestro caso, correrá en la Network namespace por defecto, la *root*, donde también correrá el nodo raíz.

¹⁰Técnicamente son dos, el ofdatapath y el ofprotocol

4.4.1. Comprobación funcional

En esta subsección se va a realizar una comprobación funcional de la implementación llevada a cabo sobre el BOFUSS. Para ello, se va a hacer uso de una topología relativamente sencilla, pero que tiene la conectividad suficiente para ver la operativa real del protocolo IoTorii. La topología está compuesta de 6 nodos, y se ha marcado al nodo A como nodo raíz de la topología, es decir será el nodo que tenga acceso al controlador SDN. La topología se puede apreciar en la Figura 4.10, habiendo seis nodos identificados por [A,B,C,D,E,F], los cuales tienen las siguientes direcciones MAC respectivamente, [:01,:02,:03,:04,:05,:06].

Para poner en marcha el escenario solo se tendrá que lanzar el shell script (Ver sección 2.9), y para limpiarlo, como se indicaba anteriormente, se hará uso de la utilidad de Mininet para limpiar los recursos reservados para la emulación de la topología. A continuación, se indica en el bloque de Código 4.5 como trabajar con el escenario.

Código 4.5: Puesta en marcha y limpieza del escenario

```
1 # Levantar escenario básico
2 sudo ./topo_basic.sh
3
4 # Limpiamos
5 sudo mn -c
```

La comprobación funcional consistirá en el estudio y análisis de las tablas de vecinos y HLMAC de cada nodo de la topología. En primera instancia se debatirá como deberá funcionar el protocolo de forma teórica, más adelante se comprobará accediendo a los logs de cada `win-BOFUSS` el contenido de las tablas de vecinos y de HLMAC para comprobar si el funcionamiento esperado y real coinciden. En esta validación se habría querido poner alguna captura de Wireshark, pero, al no haber programado un disector para el protocolo IoTorii, no será posible apreciar el contenido de los mensajes intercambiados entre los nodos de la topología, solo se verá un campo que indique data con una cadena hexadecimal.

Según se ha indicado anteriormente, al trabajar con los mismos recursos que Mininet-WiFi, podremos utilizar herramientas auxiliares que incorporan. Si ya se ha lanzado la topología, y se inicia el núcleo de Mininet-WiFi, pasándole las coordenadas de cada nodo se puede comprobar si el escenario se ha levantado correctamente dibujando la topología. A continuación, en la Figura 4.11, se puede apreciar la topología básica levantada a través de Mininet-WiFi.

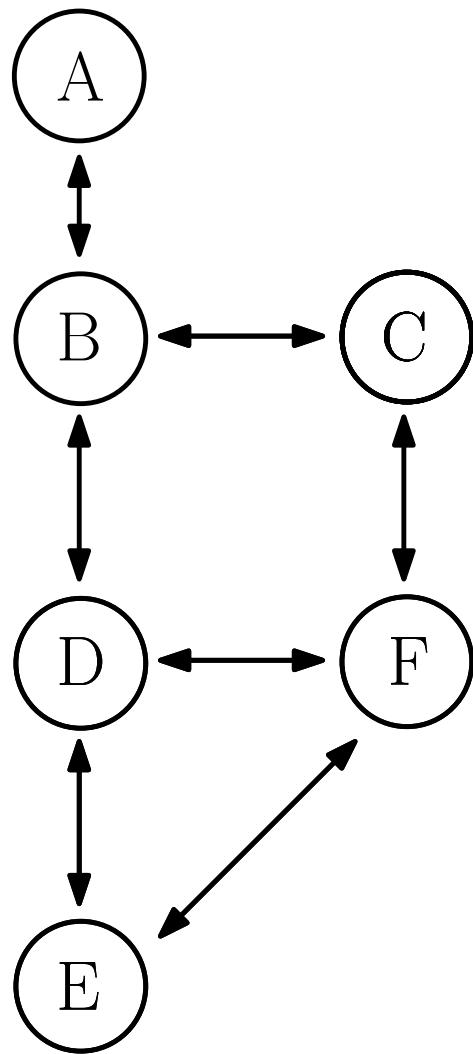


Figura 4.10: Topología básica a evaluar el funcionamiento de IoTorii

Otro problema que se ha encontrado a la hora de emular, es el problema mencionado anteriormente en la memoria en la Sección 3.7.2.1. Cuando se ha intentado emular topologías más grandes, parece que el módulo del kernel no hace un buen uso de los recursos reservados dado que las interfaces generadas son bloqueadas por el kernel vía *software*. Por ello, se ha querido demostrar el funcionamiento en una topología sencilla, que funcione de forma sólida y sea fácil de apreciar el funcionamiento del protocolo.

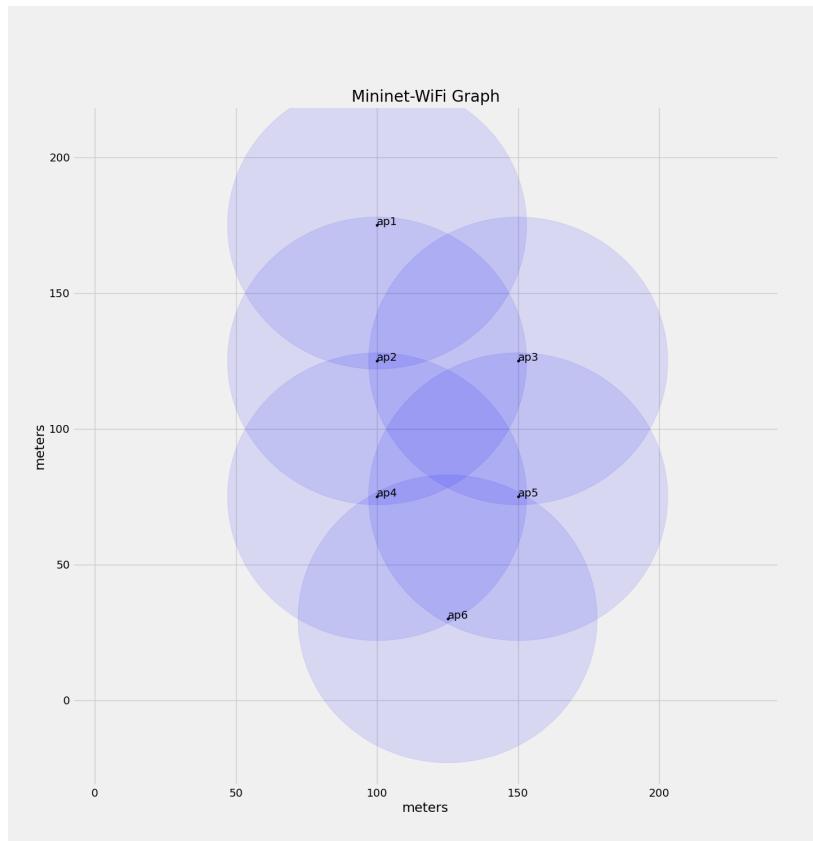


Figura 4.11: Topología básica representada con el motor de Mininet-WiFi

4.4.1.1. Tablas de vecinos

En este punto se va a estudiar las tablas de vecinos para la topología anteriormente presentada, así como la operativa básica de anuncio de vecindad mediante el intercambio de mensajes *Hello*. Recordemos que la condición de vecinos se da siempre y cuando dos nodos se encuentren en rango de señal entre ellos y se hayan notificado respectivamente. Cuando se notifican entre ellos, cada nodo apunta en una tabla los vecinos que tiene, guardando su dirección MAC real junto a un sufijo único. Dicho sufijo puede variar en función de la implementación, en este caso es un contador al uso.

Si nos detenemos a observar la Figura 4.12, podremos apreciar cómo todos los nodos presentes en la topología generan mensajes del tipo *Hello* únicamente con aquellos nodos que se encuentran dentro de su rango de comunicación. Es importante destacar que la cantidad de mensajes *Hello* generados será directamente proporcional al tamaño de la topología, es decir, a la cantidad de nodos N presentes en ella.

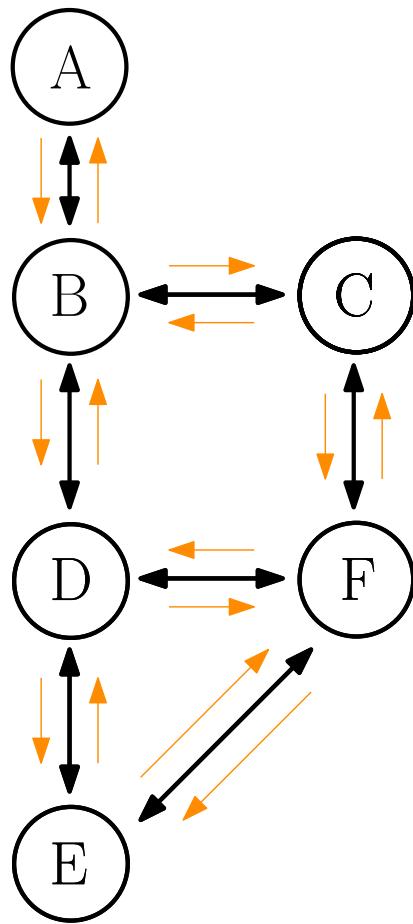


Figura 4.12: Intercambio de mensajes de tipo *Hello* en la topología básica

Esta relación entre la cantidad de mensajes *Hello* y el número de nodos se debe a que cada nodo necesita establecer y mantener un vínculo de conectividad con los demás nodos dentro de su rango. A medida que aumenta el número de nodos en la topología, la complejidad de los mensajes *Hello* también aumenta, ya que cada nodo debe comunicarse con todos los demás nodos que están a su alcance.

De esta manera, la generación de mensajes *Hello* se convierte en un mecanismo esencial para mantener una visión actualizada de la topología de la red, permitiendo a los nodos conocer y establecer conexiones con sus vecinos cercanos. Así, la cantidad y el contenido de los mensajes *Hello* desempeñan un papel crítico en la configuración y el funcionamiento adecuado del protocolo en una topología determinada.

Observando la topología, ver Figura 4.13, se puede apreciar como el único vecino que tendrá el nodo raíz será el nodo B, por lo que después de notificarse, el nodo A apuntará al nodo B

en su tabla de vecinos, teniendo en cuenta su dirección MAC real, y le asignará un sufijo único que en este caso es un contador que se va incrementando según se descubren vecinos nuevos.

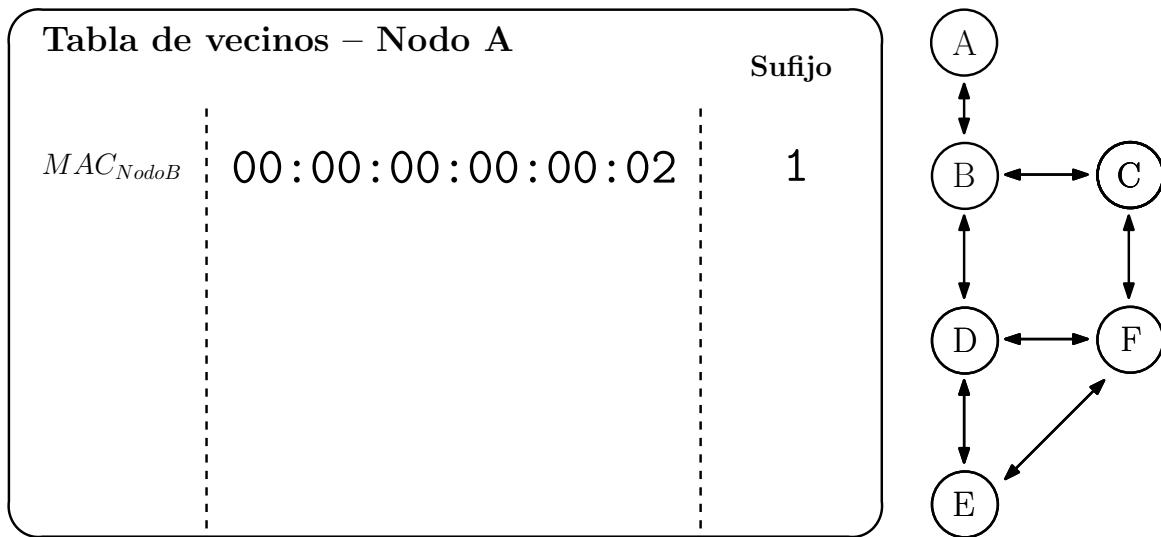


Figura 4.13: Tabla de vecinos para el nodo A

Si nos fijamos por ejemplo en la tabla de vecinos del nodo B, ver Figura 4.14, se puede ver como en este caso este nodo tiene como vecinos a los nodos [A, C, D], por lo que tendrá que generar tres sufijos únicos, [1, 2, 3], uno por cada uno de ellos, para después almacenar la MAC real junto a dicho sufijo único.

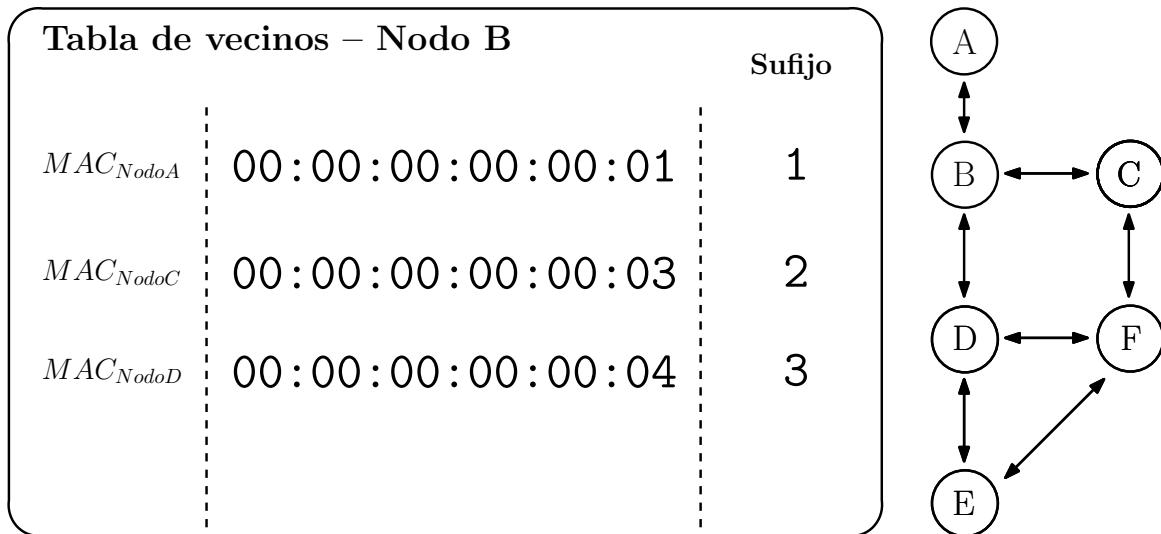


Figura 4.14: Tabla de vecinos para el nodo B

En cuanto al nodo C, ver Figura 4.15, se puede ver en la topología que solo tendrá como vecinos a los nodos [B,F], por lo que solo hará uso de dos sufijos únicos.

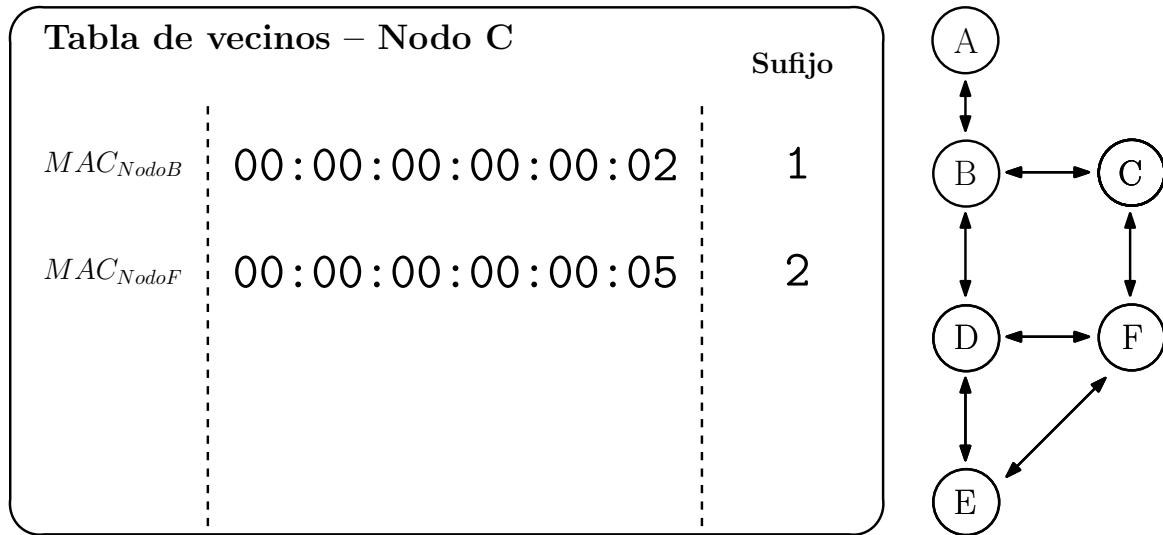


Figura 4.15: Tabla de vecinos para el nodo C

En cuanto a los nodos D, F y E, ver Figuras 4.16, 4.18 y 4.17, se tienen ambos como vecinos y rellenarán sus tablas de vecinos de forma análoga. Mencionar que el nodo D, al igual que el nodo B, es un nodo hiperconectivo de la topología por lo que tendrá bastantes rutas alternativas para alcanzar al nodo root.

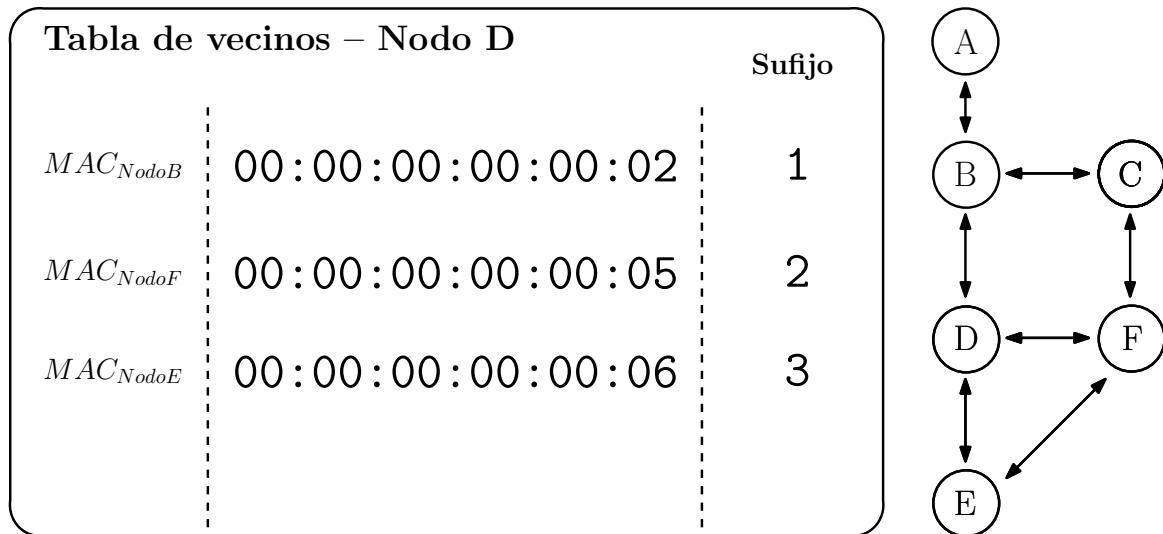


Figura 4.16: Tabla de vecinos para el nodo D

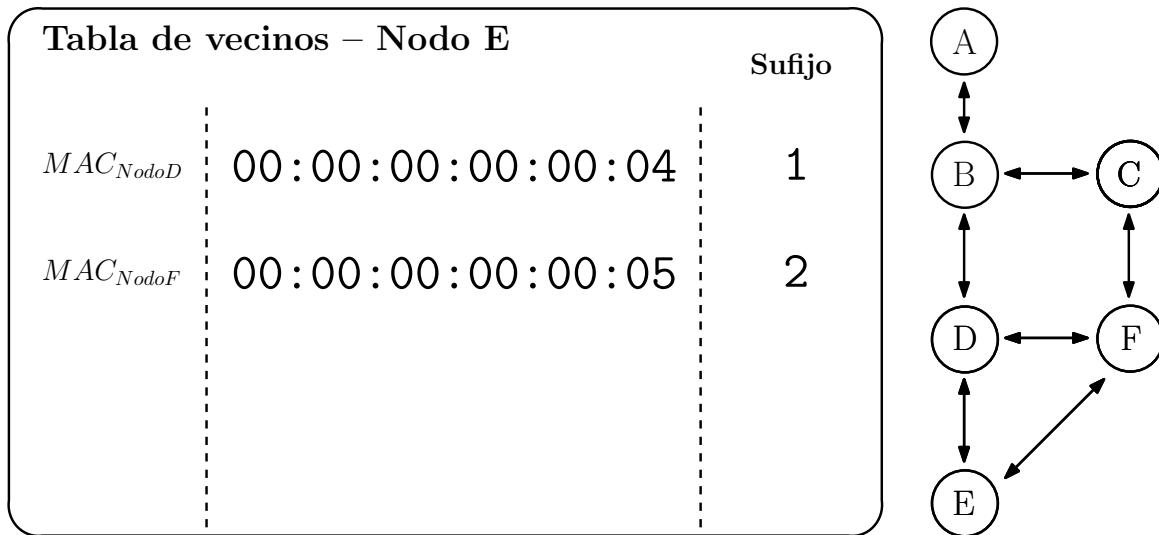


Figura 4.17: Tabla de vecinos para el nodo E

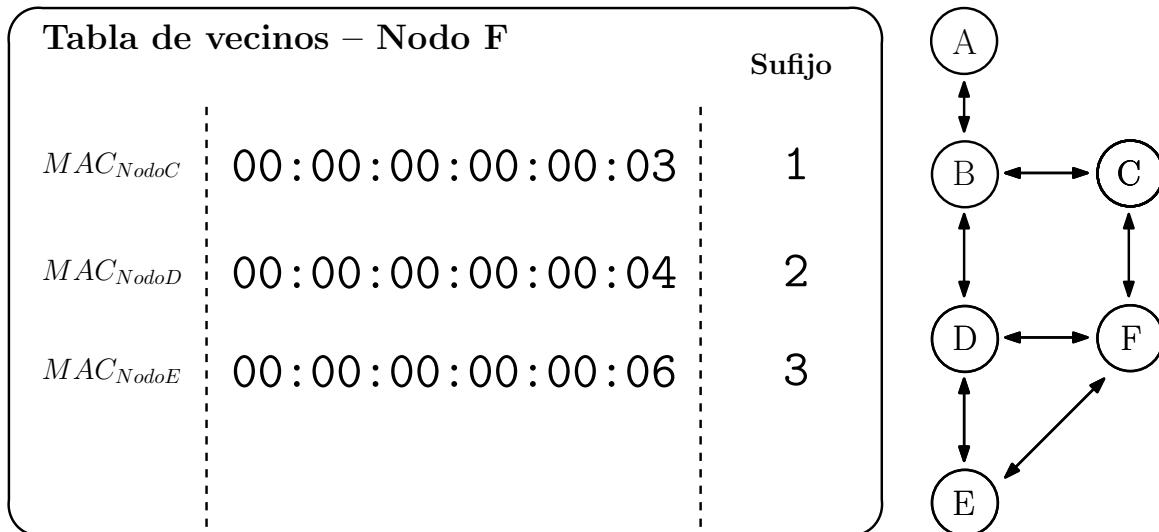


Figura 4.18: Tabla de vecinos para el nodo F

Hemos examinado el comportamiento teórico del proceso de notificación de vecindad entre los nodos de la topología. Hemos explorado cómo los mensajes *Hello* permiten a los nodos establecer y mantener conexiones con sus vecinos cercanos, creando una visión actualizada de la estructura de la red.

Sin embargo, la comprobación teórica por sí sola no basta para garantizar la solidez y efectividad de la implementación. Es por ello que ahora nos adentraremos en la fase de contrastar este comportamiento teórico con el funcionamiento real de la misma topología,

la cual hemos emulado con las condiciones reales de una red inalámbrica. Esto nos permitirá evaluar el desempeño y la estabilidad de la implementación en un entorno pseudo-real. Analizaremos el comportamiento de los nodos en tiempo real, observando cómo interactúan entre sí y si la generación y recepción de mensajes *Hello* se desarrolla de acuerdo con lo esperado. Mediante este enfoque, buscaremos obtener conclusiones sólidas y fiables sobre la capacidad del protocolo para adaptarse a cambios en la topología, como la aparición o desaparición de nodos y enlaces.

Para ello, se ha levantado la topología según se ha explicado en el bloque de Código 4.5, y se han abierto los seis ficheros de log asociados a las tablas de vecinos de cada nodo de la topología (`/tmp/ap_X_nb.log`). Como se puede apreciar en la Figura 4.19, todas las tablas de vecinos coinciden con las tablas que se han ido analizando poco a poco de forma teórica, por lo que se asume que esta parte de la implementación se ha realizado satisfactoriamente.

```
arpPath@arpPath-david:~ 104x17
arpPath@arpPath-david:~ 104x16
arpPath@arpPath-david:~ 104x15
arpPath@arpPath-david:~ 104x16
arpPath@arpPath-david:~ 104x16
```

Cada 2.0s: head /tmp/ap_A_nb.log

1.00-00:00-00:00:02

2.00-00:00-00:00:03

3.00-00:00-00:00:04

Cada 2.0s: head /tmp/ap_B_nb.log

1.00-00:00-00:00:01

2.00-00:00-00:00:03

3.00-00:00-00:00:04

Cada 2.0s: head /tmp/ap_D_nb.log

1.00-00:00-00:00:02

2.00-00:00-00:00:05

3.00-00:00-00:00:06

Cada 2.0s: head /tmp/ap_E_nb.log

1.00-00:00-00:00:04

2.00-00:00-00:00:05

Cada 2.0s: head /tmp/ap_F_nb.log

1.00-00:00-00:00:03

2.00-00:00-00:00:04

3.00-00:00-00:00:06

Figura 4.19: Tabla de vecinos extraídas de los logs de los *software switches*

4.4.1.2. Tablas HLMAC

En este punto se van a estudiar las tablas de rutas conocidas como HLMAC, así como la operativa básica de anuncio de rutas al nodo raíz mediante el intercambio de mensajes *SetHLMAC*. Recordemos que para que un nodo genere un mensaje genere un mensaje de *SetHLMAC*, antes ha tenido que recibir una HLMAC, y tener vecinos para intercambiarla. Cuando se notifica una ruta a un vecino, va la HLMAC recibida más un sufijo único del vecino. De esta forma la HLMAC aumenta en un nivel más.

A continuación, en la Figura 4.20, se ha intentado ilustrar todo el proceso de difusión de estos mensajes *SetHLMAC* a lo largo de la topología. Se han incluido cuatro *snapshots* del proceso de difusión, siguiendo un orden cronológico de izquierda a derecha y de arriba abajo. Es importante señalar que aunque en esta figura se hayan representado así el intercambio de mensajes, no tiene por qué realizarse en ese mismo orden. Esto es así porque en la realidad hay que tener en cuenta latencias y tiempos de procesamiento.

Volviendo a la Figura 4.20, se puede apreciar como en primera instancia la primera oleada de difusión empieza por el nodo A, generando la HLMAC con valor 1, la cual será difundida a todos los vecinos de A, que en este caso es el nodo B. Si nos fijamos en la tabla de vecinos del nodo A, podemos ver como el nodo B tiene el sufijo único con valor 1. Por tanto, la HLMAC que le llegará al nodo B será 1.1. Este proceso se lleva a cabo de forma análoga por toda la topología, el nodo B, una vez que le llega la nueva HLMAC con valor 1.1 procederá a difundirla a todos sus vecinos, que en este caso son el nodo C y D respectivamente. El nodo B comprobará su tabla de vecinos en busca de los nodos C y D, y generará las HLMAC 1.1.3 y 1.1.2.

Se quiere mencionar que hay algunas HLMAC que no se difundían para evitar bucles en la topología. La condición de ser una HLMAC válida es la de no ser hija de una ya almacenada en la tabla de rutas de un nodo en cuestión. Por ejemplo, si tenemos la ruta 1.1.2, todas las rutas del tipo 1.1.2.X.Y serán descartadas.

Otro detalle que se quiere mencionar de la implementación, es el criterio que se ha seguido para elegir una ruta al nodo raíz frente a otras. En este caso se marca como HLMAC activa aquella que llegue antes al nodo. Se pueden establecer muchos criterios para tomar esta decisión, desarrollando una función objetivo y evaluar cada ruta en base a unos criterios, pero en este caso, se va a coger la primera ruta que llegue que se entenderá que es aquella ruta más rápida en alcanzar al *root*.

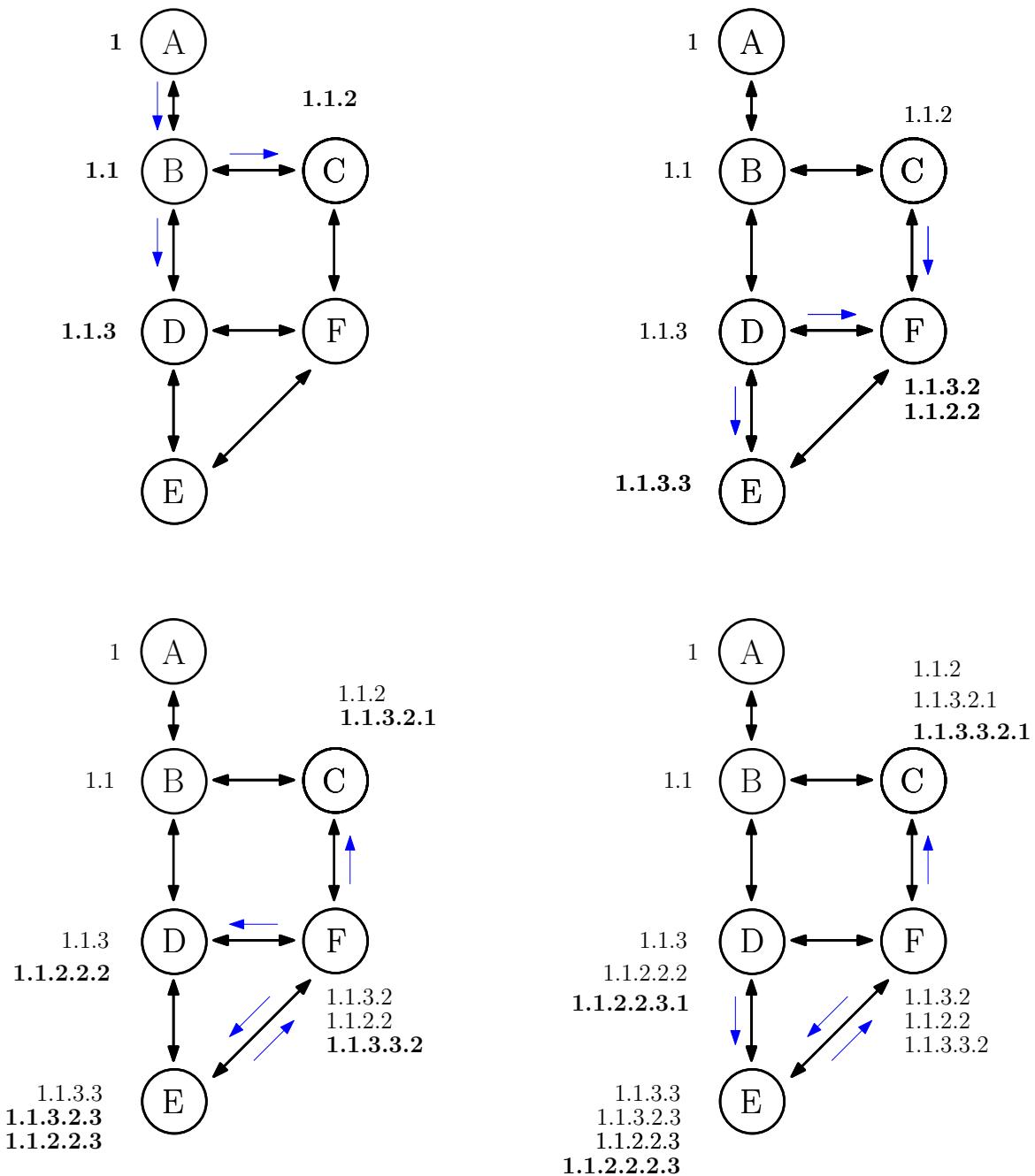


Figura 4.20: Proceso de difusión de mensajes de tipo *SetHLMAC* en la topología básica

Si nos fijamos en la comprobación en real del escenario emulado, todas las tablas de rutas HLMAC coinciden por lo que se da por satisfactoria la comprobación funcional del protocolo. La comprobación en real se ha llevado a cabo consultando los ficheros de logs generados por el `win-BOFUSS` en el path `/tmp/ap_X_hlmac.log`. El formato de dichos ficheros de log es el siguiente, `flag | HLMAC` . Por lo que se puede apreciar aquellas HLMAC que llegan antes al nodo serán marcadas como rutas activas al nodo raíz. Una vez difundidas las etiquetas, y marcada al menos una por nodo, se puede apreciar como a partir de la topología física inalámbrica se puede obtener la topología lógica.

En la Figura 4.22, podemos observar la topología física que ha sido estudiada en la sección anterior, compuesta por un total de 6 nodos. Tras difundir las etiquetas, hemos aplicado el criterio basado en la ruta más rápida, lo que ha permitido que cada nodo de la topología obtenga su respectiva HLMAC, como se muestra en la figura. Una vez que todos los nodos han adquirido sus HLMAC es probable que algunos enlaces de la topología queden en desuso, ya que no existirá ninguna etiqueta que haga uso de dichos enlaces. Esto se debe a que el proceso de determinar las HLMAC permite a los nodos seleccionar rutas óptimas hacia el nodo raíz, lo que podría resultar en que algunos enlaces no sean necesarios. Es importante resaltar que este proceso de selección de rutas y asignación de HLMAC tiene un impacto significativo en la eficiencia y utilización de los enlaces en la topología. Aquellos enlaces que no sean utilizados quedan en desuso, lo que podría liberar recursos y mejorar la capacidad de la red para adaptarse a cambios en la topología.

Si consideramos todas las $HLMAC_{active}$ de los nodos en la topología, podemos definir una topología lógica que contiene todos los nodos de la topología física pero solo un subconjunto de los enlaces presentes en dicha topología física. Este proceso de establecer la topología lógica puede ser formulado utilizando un grafo $G = (\mathcal{N}, \mathcal{L})$, donde \mathcal{N} es un conjunto de N nodos y $\mathcal{L} \subseteq \{\{i, j\} \mid i, j \in \mathcal{N} \text{ and } i \neq j\}$ es un conjunto de enlaces, cada uno compuesto por un par desordenado de nodos (es decir, cada enlace conecta dos nodos distintos).

Al obtener la topología lógica, se crea un subgrafo $G' = (\mathcal{N}', \mathcal{L}')$ en el que se cumple que $\mathcal{N}' \subseteq \mathcal{N}$ y $\mathcal{L}' \subseteq \mathcal{L}$. Es decir, la topología lógica contiene un subconjunto de los nodos y un subconjunto de los enlaces de la topología física. Sin embargo, se debe destacar que en este caso, los nodos son idénticos en ambas topologías, lo que se expresa mediante la relación de igualdad para los nodos. Esto significa que todos los nodos en la topología lógica \mathcal{N}' son iguales a los nodos en la topología física \mathcal{N} . Es decir, si $a_i \in \mathcal{N}$ y $b_i \in \mathcal{N}'$, donde $0 \leq i \leq N$, se cumple que para cada nodo a_i y su correspondiente nodo en la topología lógica b_i , se tiene que $a_i = b_i$.

<code>arpba@arpba-path-david:~ 104x18</code>	<code>arpba@arpba-path-david:~ 104x18</code>	<code>arpba@arpba-path-david:~ 104x18</code>
<code>loda 2.0s: head /tmp/ap_A_hmac.log</code>	<code>Cada 2.0s: head /tmp/ap_B_hmac.log</code>	<code>arpba@arpba-path-david:~ 104x18</code>
<code>1 1</code>	<code>1 1.1</code>	
<code>loda 2.0s: head /tmp/ap_C_hmac.log</code>	<code>Cada 2.0s: head /tmp/ap_D_hmac.log</code>	<code>arpba@arpba-path-david:~ 104x15</code>
<code>1 1.1.2</code>	<code>1 1.1.3</code>	<code>arpba@arpba-path-david:~ 104x15</code>
<code>9 1.1.3.2.1</code>	<code>0 1.1.2.2.2</code>	<code>arpba@arpba-path-david:~ 104x15</code>
<code>9 1.1.3.2.1</code>	<code>0 1.1.2.2.3.1</code>	<code>arpba@arpba-path-david:~ 104x15</code>
<code>loda 2.0s: head /tmp/ap_E_hmac.log</code>	<code>Cada 2.0s: head /tmp/ap_F_hmac.log</code>	<code>arpba@arpba-path-david:~ 104x15</code>
<code>1 1.1.3.3</code>	<code>1 1.1.3.2</code>	<code>arpba@arpba-path-david:~ 104x15</code>
<code>9 1.1.3.2.3</code>	<code>0 1.1.2.2</code>	<code>arpba@arpba-path-david:~ 104x15</code>
<code>9 1.1.2.2.3</code>	<code>0 1.1.3.3.2</code>	<code>arpba@arpba-path-david:~ 104x15</code>
<code>9 1.1.2.2.3</code>		<code>arpba@arpba-path-david:~ 104x15</code>

Figura 4.21: Tabla de HIIMAC extraídas de los logs de los software switches

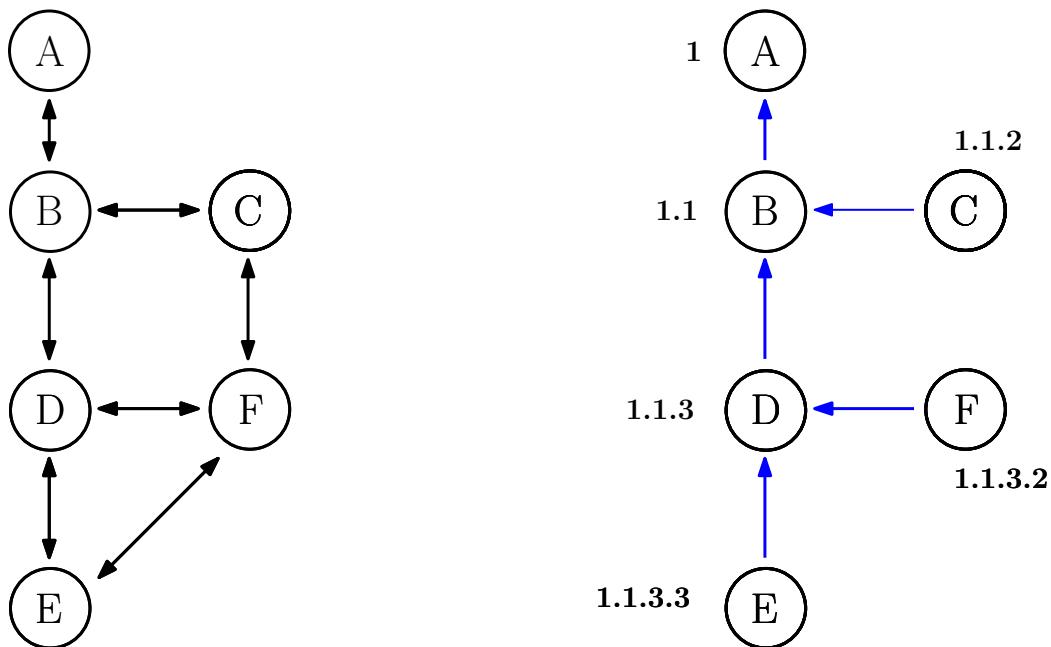


Figura 4.22: Establecimiento de la topología lógica

4.4.1.3. Conexiones in-band

En este punto se quiere demostrar que se están llevando a cabo las conexiones in-band a través del nodo A de la topología indicada anteriormente hacia el controlador Ryu que estará corriendo en la Network Namespace *root*. Para ello, haciendo uso del escenario anteriormente expuesto (Ver Figura 4.9), solo se tendrá en la Network Namespace por defecto al nodo A, y el resto de nodos se conectarán entre ellos a través del módulo del kernel `mac80211_hwsim` al estar aislados en Network Namespaces.

Una vez lanzada la topología podemos comprobar que corren en Network Namespaces diferente de la siguiente forma. Primero necesitamos obtener todos los identificadores de proceso de todos los nodos de la topología. Eso lo haremos ejecutando un `ps aux` y filtramos por nombres de nodo. En la Figura 4.23 se indica cómo realizar esta comprobación.

```
[root@n0obie-Zenbook:~$ sudo ps aux | grep -e 'ap[0-9]'
```

User	PPID	UID	GID	State	Start Time	Time	Command
root	10478	0.0	0.0	12584	3864 pts/8	Ss+ 14:40 0:00 bash --norc --noediting -is mininet:ap1	
root	10483	0.0	0.0	12584	3848 pts/9	Ss+ 14:40 0:00 bash --norc --noediting -is mininet:ap2	
root	10487	0.0	0.0	12584	3848 pts/10	Ss+ 14:40 0:00 bash --norc --noediting -is mininet:ap3	
root	10491	0.0	0.0	12584	3892 pts/11	Ss+ 14:40 0:00 bash --norc --noediting -is mininet:ap4	
root	10495	0.0	0.0	12584	3866 pts/12	Ss+ 14:40 0:00 bash --norc --noediting -is mininet:ap5	
root	10499	0.0	0.0	12584	3908 pts/13	Ss+ 14:40 0:00 bash --norc --noediting -is mininet:ap6	

Figura 4.23: Obtención de los PID de todos los nodos de la topología

Una vez se tienen los PID de todos los nodos de la topología se podrá consultar dentro

de la información de proceso, en que Namespaces se está ejecutando dicho proceso. En este caso, solo nos interesa la información relativa al identificador de Namespace de Red. A continuación, en la Figura 4.24 se demuestra como el nodo A (*ap1*) se está ejecutando en una Network Namespace diferente a la Network Namespace en la que corre el nodo B (*ap2*).

```
n0obie@n0obie-Zenbook:~$ sudo ls -la /proc/10478/ns/net | grep -e '\[[0-9]+\+\]'  
lrxwxrwxrwx 1 root root 0 jul 20 14:40 /proc/10478/ns/net -> net:[4026531840]  
n0obie@n0obie-Zenbook:~$ sudo ls -la /proc/10483/ns/net | grep -e '\[[0-9]+\+\]'  
lrxwxrwxrwx 1 root root 0 jul 20 14:40 /proc/10483/ns/net -> net:[4026532998]  
n0obie@n0obie-Zenbook:~$
```

Figura 4.24: Comprobación de identificadores de Network Namespaces de los nodos A y B de la topología

Para llevar a cabo la prueba se plantea iniciar la aplicación de descubrimiento topológico de Ryu a la par que se levanta la topología. Si bien es cierto que dicha aplicación trabaja con el protocolo LLDP y no reconocerá la relación de enlaces, por ser un medio inalámbrico, el hecho de que los detecte significará que todos los nodos de la topología están descubiertos por el controlador, es decir, que será posible que todos los nodos tengan una conexión OpenFlow con él. Para lanzar Ryu con la aplicación de descubrimiento topológico se tiene que ejecutar el siguiente comando (Ver bloque de Código 4.6).

Código 4.6: Lanzamos el controlador de Ryu con la App de topo.discovery

```
1 # Requiere permisos de root  
2 ryu-manager ryu.app.gui_topology.gui_topology  
3  
4  
5 # Nos abrirá una interfaz web en el puerto 8080 de localhost --> http://localhost:8080  
6 # En dicha interfaz se verán los nodos encontrados por el controlador
```

Si activamos el núcleo de Mininet-WiFi y le pasamos las coordenadas como antes, seremos capaces de pintar la topología, y además, comparar con el descubrimiento topológico realizado por el controlador Ryu. Si nos fijamos en la Figura 4.25 podemos afirmar que ha descubierto a todos los nodos de la topología, por lo que será posible llevar a cabo una conexión in-band.

Si se quiere ir un paso más, podemos abrir esta vez Wireshark y consultar en la interfaz de *loopback* de la Network Namespace por defecto si realmente están llegando las conexiones OpenFlow al controlador Ryu. Si nos fijamos en la Figura 4.26, se puede ver como los 6 nodos de la topología son capaces de conectarse con el controlador. Se puede ver como hay

conexión bi-direccional, al haber 6 mensajes de tipo *Hello* en un sentido 6 en el otro, si nos fijamos después de los mensajes de tipo *Hello* empieza a ver una negociación OpenFlow al uso, preguntando por las *features*. Por tanto, se demuestra que las conexiones in-band se pueden llevar a cabo satisfactoriamente.

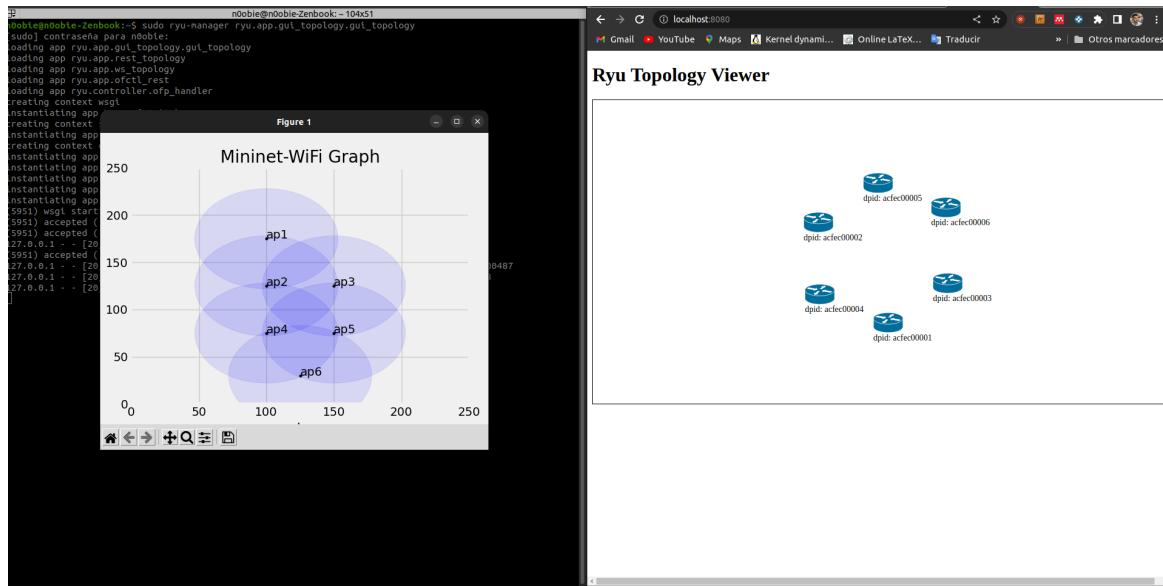


Figura 4.25: Descubrimiento topológico llevado a cabo por Ryu

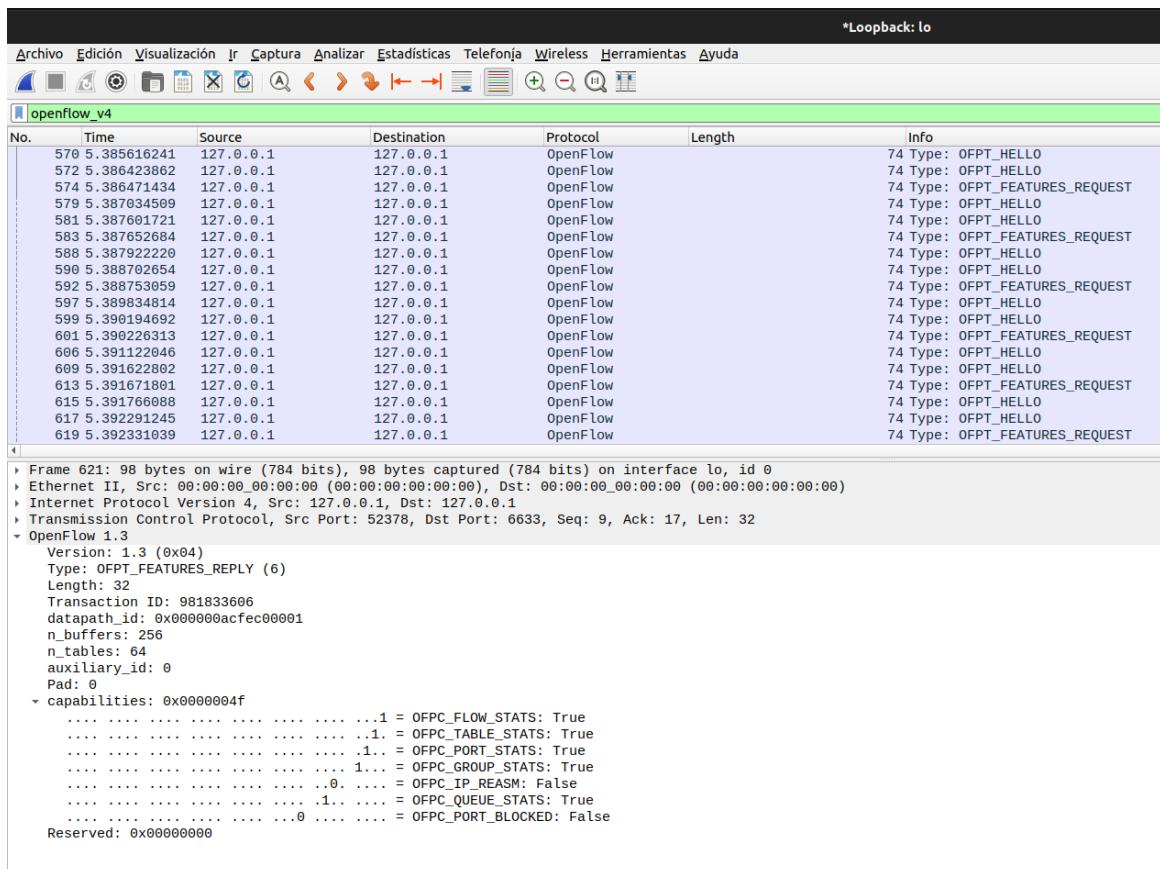


Figura 4.26: Conexiones in-band con el controlador Ryu que corre en la interfaz de *loopback* de la Network Namespace por defecto

5. Conclusiones y trabajo futuro

En este capítulo, se concluirá la memoria presentando las conclusiones del trabajo realizado y evaluando tanto el trabajo en sí como los resultados obtenidos durante el desarrollo. Además, se explorarán las posibles vías de trabajo futuro que surgieron a medida que se llevaba a cabo el TFM. Se presentarán estas vías y se cerrará la memoria con una visión general y reflexiva sobre el proyecto realizado.

5.1. Conclusiones del TFM

En este TFM, se ha llevado a cabo el diseño e implementación de un protocolo de control escalable para redes IoT en entornos SDN de la nueva generación de redes móviles, 6G. El protocolo se ha basado en un enfoque de control in-band, estableciendo la conectividad entre los nodos de la red y el controlador a través del plano de datos para transmitir información de control.

En la primera fase del proyecto, se realizó un exhaustivo estudio y documentación de las principales herramientas y tecnologías relacionadas, sentando así una sólida base teórica antes de abordar el diseño y prototipado del protocolo de control in-band. Posteriormente, se analizaron las necesidades y características de las distintas tecnologías para seleccionar las herramientas más adecuadas para la implementación del protocolo de control. Se llevó a cabo un estudio de estas herramientas con el objetivo de lograr una implementación optimizada en la medida de lo posible. El proyecto se completó mediante la validación del protocolo desarrollado a través de la emulación, lo que permitió realizar pruebas de funcionamiento del protocolo implementado en el entorno BOFUSS sobre interfaces inalámbricas emuladas. De este modo, se pudo comprobar el correcto desempeño del protocolo en diferentes casos de uso. Esta validación resulta fundamental para asegurar que el protocolo cumple con los requisitos de escalabilidad y control en entornos IoT y SDN. De forma adicional, se ha estudiado como validar el protocolo en un entorno real, no siendo posible, dado que hay problemas en la traducción de símbolos a la arquitectura ARM.

Durante todo el proceso de ejecución del proyecto, se generaron diversas contribuciones no asociadas directamente a los objetivos del proyecto, pero sí a la comunidad SDN. Se

aportó claridad al documentar el funcionamiento interno de herramientas como “dpctl” y su interfaz de comandos, lo que permitió resolver problemas relacionados con ellas. Se exploró a fondo el funcionamiento de Mininet-WiFi, documentando su jerarquía de clases y los módulos que utiliza en el kernel de Linux para emular el entorno de radio, lo que facilitó la identificación de posibles fuentes de errores debido a desvíos en las interfaces inalámbricas emuladas. También se profundizó en el funcionamiento interno de BOFUSS, solucionando problemas de compilación para las últimas versiones de distribuciones Linux y actualizando documentación incorrecta sobre su funcionamiento.

Además, se crearon escenarios replicables mediante Vagrant, que incluyen todas las herramientas relacionadas con entornos SDN (Mininet, Mininet-WiFi, Ryu, ONOS, BOFUSS) en la última versión de Ubuntu 22.04. Esto ha permitido mantener actualizadas estas herramientas y proyectos que habían sido abandonados. Por ejemplo, el proyecto BOFUSS fue retomado después de casi cuatro años sin recibir ninguna contribución (*commit*), gracias a la colaboración entre el autor y el autor de este TFM, quien asumió un rol de mantenedor en el proyecto.

En conclusión, este TFM ha logrado desarrollar un mecanismo de control in-band para dispositivos IoT en entornos 6G. Además, se ha generado una cadena de valor al aportar conocimiento y mejoras a herramientas y proyectos previamente abandonados, lo que ha contribuido a revitalizarlos.

Personalmente, este proyecto ha sido un reto, por su complejidad a nivel técnico, y por lo transversal que es. Se ha estudiado las bondades y las tecnologías habilitantes del 6G, el entorno radio, las nuevas configuraciones de *arrays* de antenas, las arquitecturas propuestas en la nueva generación. Asimismo, se ha ido trabajando herramienta por herramienta del mundo SDN, se ha programado desde alto nivel (Python) a bajo nivel (BOFUSS), se ha trabajado desde el espacio de usuario a espacio de kernel, incluso se ha tenido que diseñar una plataforma de emulación inalámbrica. Por todo ello, por el carácter polímata y transversal del proyecto, creo que este TFM ilustra bastante bien las competencias y habilidades que un Ingeniero de Telecomunicación debería tener, que es por ello, por lo que somos reconocidos tanto en la industria, como en la sociedad.

5.2. Líneas de trabajo futuro

Tras concluir y revisar el alcance conseguido de los objetivos propuestos para la ejecución del proyecto, podemos apuntar a próximas líneas de trabajo futuro basadas en los siguientes puntos:

- Realizar un despliegue en real utilizando Raspberry Pi (RPi) que tienen una interfaz WiFi nativa para evaluar el funcionamiento del win-BOFUSS con varias RPis. Esto permitirá verificar si el rendimiento es similar al observado durante el proyecto.
- Realizar la integración del switch BOFUSS en la jerarquía de clases de Mininet-WiFi. Esto facilitará la realización de pruebas y evaluaciones más consistentes al combinar las funcionalidades de ambos proyectos.
- Añadir seguridad mediante el uso de TLS en las conexiones OpenFlow in-band desde el UserAP hacia el controlador. Esto garantizará la confidencialidad y la integridad de las comunicaciones, protegiendo la información transmitida entre el UserAP y el controlador.
- Realizar simulaciones de pérdida utilizando el protocolo desarrollado y evaluar cómo afecta al funcionamiento intrínseco de la operativa programada. Esto ayudará a comprender el impacto de la pérdida de paquetes en el rendimiento del sistema y permitirá optimizar el protocolo en función de los resultados obtenidos.
- Modificar la interfaz del BOFUSS con la herramienta de gestión “dpctl” para que implemente una interfaz estándar como JSON.
- Arreglar el trazado de símbolos a la arquitectura ARM. Este punto tiene que ver con el primero a líneas a futuro, porque si no se porta correctamente a ARM según se ha explicado, será imposible desplegarlo en una RPi, o en dispositivos ARM que porten Openwrt, por ejemplo.

Estas líneas de trabajo futuro proporcionan oportunidades para mejorar y ampliar el proyecto, explorando nuevas funcionalidades, mejorando la seguridad y evaluando su rendimiento en diferentes escenarios y condiciones. Como se ha indicado en las conclusiones, aparte de apuntar a las posibles líneas de trabajo a futuro, también hay que mencionar la realidad inmediata, que después de estar trabajando con el *software switch* BOFUSS, conseguir hacerlo funcionar en las últimas versiones se tomará el relevo a Eder como mantenedor del proyecto en una nueva organización agnóstica.

Bibliografía

- [1] S. Balaji, K. Nathani, and R. Santhakumar, “IoT Technology, Applications and Challenges: A Contemporary Survey,” *Wireless Personal Communications*, vol. 108, pp. 363–388, 9 2019. [Online]. Available: <https://link.springer.com/article/10.1007/s11277-019-06407-w>
- [2] S. Li, L. D. Xu, and S. Zhao, “5G Internet of Things: A survey,” *Journal of Industrial Information Integration*, vol. 10, pp. 1–9, 6 2018.
- [3] IoT-Analytics, “Number of connected IoT devices growing 18% globally.” [Online]. Available: <https://iot-analytics.com/number-connected-iot-devices/>
- [4] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, D. Niyato, O. Dobre, and H. V. Poor, “6G Internet of Things: A Comprehensive Survey,” *IEEE Internet of Things Journal*, vol. 9, pp. 359–383, 1 2022.
- [5] J. Liang, L. Li, and C. Zhao, “A transfer learning approach for compressed sensing in 6G-IoT,” *IEEE Internet of Things Journal*, vol. 8, no. 20, pp. 15 276–15 283, 2021.
- [6] “Europe scales up 6G research investments Shaping Europe’s digital future.” [Online]. Available: <https://digital-strategy.ec.europa.eu/en/news/europe-scales-6g-research-investments-and-selects-35-new-projects-worth-eu250-million>
- [7] “The Smart Networks and Services Joint Undertaking | Shaping Europe’s digital future.” [Online]. Available: <https://digital-strategy.ec.europa.eu/en/policies/smart-networks-and-services-joint-undertaking>
- [8] “Europe launches the second phase of its 6G Research and Innovation Programme - Shaping Europe’s digital future.” [Online]. Available: <https://digital-strategy.ec.europa.eu/en/news/europe-launches-second-phase-its-6g-research-and-innovation-programme>
- [9] M. A. Uusitalo, M. Ericson, B. Richerzhagen, E. U. Soykan, P. Rugeland, G. Fettweis, D. Sabella, G. Wikström, M. Boldi, M.-H. Hamon, H. D. Schotten, V. Ziegler, E. C. Strinati, M. Latva-aho, P. Serrano, Y. Zou, G. Carrozzo, J. Martrat, G. Stea, P. Demestichas, A. Pärssinen, and T. Svensson, “Hexa-X The European 6G flagship project,”

- in *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, 2021, pp. 580–585.
- [10] M. Katz, M. Matinmikko-Blue, and M. Latva-Aho, “6Genesis Flagship Program: Building the Bridges Towards 6G-Enabled Wireless Smart Society and Ecosystem,” *Proceedings - 2018 10th IEEE Latin-American Conference on Communications, LATINCOM 2018*, 1 2019.
 - [11] “ADROIT6G: Distributed Artificial Intelligence-Driven Open & Programmable Architecture for 6G Networks.” [Online]. Available: <https://adroit6g.eu/>
 - [12] “6GTandem: Unlock new potential of wireless network.” [Online]. Available: <https://horizon-6gtandem.eu/>
 - [13] “FCC Opens Spectrum Horizons for New Services & Technologies - Federal Communications Commission.” [Online]. Available: <https://www.fcc.gov/document/fcc-opens-spectrum-horizons-new-services-technologies>
 - [14] “South Korea to launch 6G pilot project in 2026: Report.” [Online]. Available: <https://www.rcrwireless.com/20200810/asia-pacific/south-korea-launch-6g-pilot-project-2026-report>
 - [15] M. A. Uusitalo, P. Rugeland, M. R. Boldi, E. C. Strinati, P. Demestichas, M. Ericson, G. P. Fettweis, M. C. Filippou, A. Gati, M. H. Hamon, M. Hoffmann, M. Latva-Aho, A. Parssinen, B. Richerzhagen, H. Schotten, T. Svensson, G. Wikstrom, H. Wymeersch, V. Ziegler, and Y. Zou, “6G Vision, Value, Use Cases and Technologies from European 6G Flagship Project Hexa-X,” *IEEE Access*, vol. 9, pp. 160 004–160 020, 2021.
 - [16] 5G PPP Architecture Working Group, “The 6G Architecture Landscape European perspective,” 12 2022. [Online]. Available: <https://5g-ppp.eu/6g-architecture-landscape-new-white-paper-for-public-consultation/>
 - [17] Hexa-X, “Targets and requirements for 6G - initial E2E architecture,” 2022. [Online]. Available: https://hexa-x.eu/wp-content/uploads/2022/03/Hexa-X_D1.3.pdf
 - [18] D. Carrascal Acebron *et al.*, “Diseño y estudio de dispositivos IoT integrados en entornos SDN,” 2020.
 - [19] D. Carrascal, E. Rojas, J. M. Arco, D. Lopez-Pajares, J. Alvarez-Horcajo, and J. A. Carral, “A Comprehensive Survey of In-Band Control in SDN: Challenges and Opportunities,” *Electronics*, vol. 12, no. 6, p. 1265, 2023.
-

- [20] A. Jalili, H. Nazari, S. Namvarasl, and M. Keshtgari, “A comprehensive analysis on control plane deployment in SDN: In-band versus out-of-band solutions,” in *2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI)*. IEEE, 2017, pp. 1025–1031.
- [21] C. Paoloni, “one6G white paper, 6G technology overview: November 2022,” 2022.
- [22] I. FG-NET, “Network 2030: A blueprint of technology, applications and market drivers towards the year 2030 and Beyond,” *Geneva: ITU, 2019 (2019-05-07)[2020-04-07]. https://wwwitu.int/en/ITU-T/focusgroups/net2030/Documents/White_Paper.pdf, 2019.*
- [23] M. Giordani, M. Polese, M. Mezzavilla, S. Rangan, and M. Zorzi, “Toward 6G networks: Use cases and technologies,” *IEEE Communications Magazine*, vol. 58, no. 3, pp. 55–61, 2020.
- [24] M. Latva-aho, K. Leppänen, F. Clazzer, and A. Munari, “Key drivers and research challenges for 6G ubiquitous wireless intelligence,” 2020.
- [25] T. Kürner and A. Hirata, “On the impact of the results of WRC 2019 on THz communications,” in *2020 Third International Workshop on Mobile Terahertz Systems (IWMTS)*. IEEE, 2020, pp. 1–3.
- [26] I. . L. S. Committee *et al.*, “IEEE standard for high data rate wireless multi-media networks—amendment 2: 100 Gb/s wireless switched point-to-point physical layer,” *IEEE Std*, vol. 802, pp. 1–55, 2017.
- [27] V. Petrov, T. Kurner, and I. Hosako, “IEEE 802.15. 3d: First standardization efforts for sub-terahertz band communications toward 6G,” *IEEE Communications Magazine*, vol. 58, no. 11, pp. 28–33, 2020.
- [28] L. G. Ordóñez, D. P. Palomar, and J. R. Fonollosa, “Fundamental diversity, multiple-plexing, and array gain tradeoff under different MIMO channel models,” in *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2011, pp. 3252–3255.
- [29] E. Björnson, E. G. Larsson, and T. L. Marzetta, “Massive MIMO: Ten myths and one critical question,” *IEEE Communications Magazine*, vol. 54, no. 2, pp. 114–123, 2016.
- [30] J. Jeon, G. Lee, A. A. Ibrahim, J. Yuan, G. Xu, J. Cho, E. Onggosanusi, Y. Kim, J. Lee, and J. C. Zhang, “MIMO evolution toward 6G: Modular massive MIMO in low-frequency bands,” *IEEE Communications Magazine*, vol. 59, no. 11, pp. 52–58, 2021.

- [31] J. Pan, L. Cai, S. Yan, and X. S. Shen, “Network for AI and AI for network: Challenges and opportunities for learning-oriented networks,” *IEEE Network*, vol. 35, no. 6, pp. 270–277, 2021.
 - [32] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.
 - [33] J. Mills, J. Hu, and G. Min, “Communication-efficient federated learning for wireless edge intelligence in IoT,” *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5986–5994, 2019.
 - [34] N. Hu, Z. Tian, X. Du, and M. Guizani, “An energy-efficient in-network computing paradigm for 6G,” *IEEE Transactions on Green Communications and Networking*, vol. 5, no. 4, pp. 1722–1733, 2021.
 - [35] A. Sapiro, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis, “In-network computation is a dumb idea whose time has come,” in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, 2017, pp. 150–156.
 - [36] J. Woodruff, M. Ramanujam, and N. Zilberman, “P4dns: In-network dns,” in *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, 2019, pp. 1–6.
 - [37] Z. Qin, S. Deng, X. Yan, L. Lu, M. Zhao, Y. Xi, J. Wu, T. Sun, and N. Shi, “6G Data Plane: A Novel Architecture Enabling Data Collaboration with Arbitrary Topology,” *Mobile Networks and Applications*, pp. 1–12, 2023.
 - [38] V. Gazis, M. Goertz, M. Huber, A. Leonardi, K. Mathioudakis, A. Wiesmaier, and F. Zeiger, “Short paper: Iot: Challenges, projects, architectures,” in *2015 18th International Conference on Intelligence in Next Generation Networks*, 2015, pp. 145–147.
 - [39] M. U. Farooq, M. Waseem, S. Mazhar, A. Khairi, and T. Kamal, “A review on internet of things (iot),” *International Journal of Computer Applications*, vol. 113, no. 1, pp. 1–7, 2015.
 - [40] T. D. Nadeau and K. Gray, *SDN: Software Defined Networks: an authoritative review of network programmability technologies.* ” O'Reilly Media, Inc.”, 2013.
 - [41] D. Levy and N. McKeown, “Overhaul may bring better, faster internet to 100 million homes,” *Stanford University News*, 2003.
 - [42] “Onf overview,” <https://www.opennetworking.org/mission/>, accessed: 2023-06-10.
-

- [43] E. L. Fernandes, “Software Switch 1.3: An experimenter-friendly OpenFlow implementation,” Ph.D. dissertation, PhD thesis. Universidade Estadual de Campinas, 2015.
- [44] L. Zhu, M. M. Karim, K. Sharif, C. Xu, F. Li, X. Du, and M. Guizani, “Sdn controllers: A comprehensive analysis and performance evaluation study,” *ACM Computing Surveys (CSUR)*, vol. 53, no. 6, pp. 1–40, 2020.
- [45] F. Tomonori, “Introduction to ryu sdn framework,” *Open Networking Summit*, pp. 1–14, 2013.
- [46] Nippon Telegraph and Telephone Corporation, “Ryu application API - Read the Docs,” 2014. [Online]. Available: https://ryu.readthedocs.io/en/latest/ryu_app_api.html
- [47] Isaku Yamahata, “Ryu SDN framework,” 2013. [Online]. Available: <https://www.slideshare.net/yamahata/ryu-sdnframeworkupload>
- [48] Elena Olkhovskaya, Ayaka Koshibe, “ONOS : An Overview,” 2016. [Online]. Available: <https://wiki.onosproject.org/display/ONOS/ONOS%2B%2BAn+Overview>
- [49] Carmelo Cascone, “ONOS - Tutorials,” 2019. [Online]. Available: <https://wiki.onosproject.org/display/ONOS/Tutorials>
- [50] Peterson, Cascone, O’Connor, Vachuska, and Davie, “Software-Defined Networks: A Systems Approach,” 2022. [Online]. Available: <https://sdn.systemsapproach.org/onos.html>
- [51] O’Connor, Vachuska, and Davie, “Software-Defined Networks: Switch-Level Schematic,” 2022. [Online]. Available: <https://sdn.systemsapproach.org/switch.html#switch-level-schematic>
- [52] Open vSwitch - Docs, “What Is Open vSwitch?” 2016. [Online]. Available: <https://docs.openvswitch.org/en/latest/intro/what-is-ovs/>
- [53] A. Mendiola, J. Astorga, E. Jacob, and M. Higuero, “A survey on the contributions of software-defined networking to traffic engineering,” *IEEE Communications Surveys & Tutorials*, vol. 19, no. 2, pp. 918–953, 2016.
- [54] E. L. Fernandes, E. Rojas, J. Alvarez-Horcajo, Z. L. Kis, D. Sanvito, N. Bonelli, C. Cascone, and C. E. Rothenberg, “The road to BOFUSS: The basic OpenFlow userspace software switch,” *Journal of Network and Computer Applications*, vol. 165, p. 102685, 2020.
- [55] Maxim Krasnyansky, “Universal TUN/TAP device driver,” 2000. [Online]. Available: <https://www.kernel.org/doc/html/v5.8/networking/tuntap.html>
-

- [56] Ikluft, “diagram of the TUN and TAP drivers in the OSI network stack,” 2019. [Online]. Available: <https://commons.wikimedia.org/wiki/File:Tun-tap-osilayers-diagram.png>
- [57] M. Kerrisk, *veth - Virtual Ethernet Device*, The Linux man-pages project, June 2020.
- [58] docker community, “Docker Docs - Networking overview,” 2023. [Online]. Available: <https://docs.docker.com/network/>
- [59] bert hubert, “tc - show / manipulate traffic control settings,” 2001. [Online]. Available: <https://man7.org/linux/man-pages/man8/tc.8.html>
- [60] “Traffic control - debian,” <https://wiki.debian.org/TrafficControl>, accessed: 2023-06-30.
- [61] M. Kerrisk, *Namespaces (7) - overview of Linux namespaces*, The Linux man-pages project, May 2020.
- [62] Michael Kerrisk, *Network namespaces - overview of Linux network namespaces*, The Linux man-pages project, June 2020.
- [63] A. Kato, M. Takai, and S. Ishihara, “Design and implementation of a wireless network tap device for ieee 802.11 wireless network emulation,” in *2017 Tenth International Conference on Mobile Computing and Ubiquitous Network (ICMU)*, 2017, pp. 1–6.
- [64] M. Vipin and S. Srikanth, “Analysis of open source drivers for ieee 802.11 wlans,” in *2010 International Conference on Wireless Communication and Sensor Computing (ICWCSC)*, 2010, pp. 1–5.
- [65] B. Lantz, B. Heller, and N. McKeown, “A network in a laptop: rapid prototyping for software-defined networks,” in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, pp. 1–6.
- [66] B. Heller, “Reproducible network research with high-fidelity emulation,” Ph.D. dissertation, Stanford University, 2013.
- [67] R. R. Fontes, S. Afzal, S. H. B. Brito, M. A. S. Santos, and C. E. Rothenberg, “Mininet-wifi: Emulating software-defined wireless networks,” in *2015 11th International Conference on Network and Service Management (CNSM)*, 2015, pp. 384–389.
- [68] A. Dunkels, B. Gronvall, and T. Voigt, “Contiki - a lightweight and flexible operating system for tiny networked sensors,” in *29th Annual IEEE International Conference on Local Computer Networks*, 2004, pp. 455–462.
-

- [69] A. Kurniawan, *Practical Contiki-NG: Programming for Wireless Sensor Networks*. Apress, 2018.
- [70] Adnk, “Contiki operating system running an IPv6 routing protocol on 41 nodes in the Cooja Contiki network simulator,” 2012. [Online]. Available: <https://commons.wikimedia.org/wiki/File:Contiki-ipv6-rpl-cooja-simulation.png>
- [71] E. Rojas, H. Hosseini, C. Gomez, D. Carrascal, and J. R. Cotrim, “Outperforming RPL with scalable routing based on meaningful MAC addressing,” *Ad Hoc Networks*, vol. 114, p. 102433, 2021.
- [72] E. Rojas, J. Alvarez-Horcajo, I. Martinez-Yelmo, J. M. Arco, and J. A. Carral, “GA3: scalable, distributed address assignment for dynamic data center networks,” *Annals of Telecommunications*, vol. 72, pp. 693–702, 2017.
- [73] “IEEE Standard for Local and Metropolitan Area Networks:Overview and Architecture—Amendment 2: Local Medium Access Control (MAC) Address Usage,” *IEEE Std 802c-2017 (Amendment to IEEE Std 802-2014 as amended by IEEE Std 802d-2017)*, pp. 1–26, 2017.
- [74] Pablo Collado, David Carrascal, “Mininet Internals,” 2020. [Online]. Available: <https://github.com/GAR-Project/project#mininet-internals->
- [75] M. Baddeley, R. Nejabati, G. Oikonomou, M. Sooriyabandara, and D. Simeonidou, “Evolving SDN for low-power IoT networks,” in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE, 2018, pp. 71–79.
- [76] A. Anadiotis, L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, “SD-WISE: A Software-Defined WIreless SEnsor network,” *Computer Networks*, vol. 159, pp. 84 – 95, 2019. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S1389128618312192>
- [77] I. Martinez-Yelmo, J. Alvarez-Horcajo, J. A. Carral, and D. Lopez-Pajares, “eHDDP: Enhanced Hybrid Domain Discovery Protocol for network topologies with both wired/wireless and SDN/non-SDN devices,” *Computer Networks*, vol. 191, p. 107983, 2021.
- [78] B. N. Constantin *et al.*, “Desarrollo de una solución de encaminamiento para tráfico de control in-band en entornos SDN,” 2020.
- [79] D. Lopez-Pajares, J. Alvarez-Horcajo, E. Rojas, A. Asadujjaman, and I. Martinez-Yelmo, “Amaru: Plug&play resilient in-band control for SDN,” *IEEE Access*, vol. 7, pp. 123 202–123 218, 2019.

-
- [80] D. Carrascal, “in-band research,” <https://hackmd.io/@davidcawork/HJDrBYmLn>, 2023.

Lista de Acrónimos y Abreviaturas

5G	Quinta Generación de redes móviles.
6G	Sexta Generación de redes móviles.
AI	<i>Artificial Intelligence.</i>
BOFUSS	Basic OpenFlow User-space Software Switch.
BPF	Berkeley Packet Filter.
eBPF	extended Berkeley Packet Filter.
eMBB	<i>enhanced Mobile BroadBand.</i>
FCC	<i>Federal Communications Commission.</i>
FIFO	<i>First In, First Out.</i>
FL	<i>Federated Learning.</i>
HLMAC	Hierarchical Local MAC.
IEEE	Institute of Electrical and Electronics Engineers.
IoT	Internet of Things.
ITU	<i>International Telecommunication Union.</i>
LLDP	Link Layer Discovery Protocol.
M2M	Machine to Machine.
MIMO	<i>Multiple-Input and Multiple-Output.</i>
ML	<i>Machine Learning.</i>
mMTC	<i>massive Machine-Type Communication.</i>
NBI	<i>Northbound Interface.</i>
NFV	<i>Network Function Virtualization.</i>
ONF	Open Networking Foundation.
ONOS	Open Network Operating System.
OvS	Open vSwitch.
P4	Programming Protocol-independent Packet Processors.
QoS	Calidad de servicio.
RPL	<i>Routing Protocol for Low Power and Lossy Networks.</i>
SBI	<i>Southbound Interface.</i>

SDN	Software-Defined Networking.
SNMP	Simple Network Management Protocol.
TC	Traffic Control.
TFM	Trabajo Fin de Máster.
UAH	Universidad de Alcalá.
URLLC	<i>Ultra-Reliable and Low-Latency Communication.</i>
Veth	Virtual Ethernet Device.
XDP	eXpress Data Path.

A. Anexo I - Pliego de condiciones

En este anexo se podrán encontrar las condiciones materiales de las distintas máquinas donde se ha llevado a cabo el desarrollo y evaluación del proyecto. De forma adicional, se han indicado las limitaciones *hardware* así como las especificaciones *software* para poder replicar el proyecto de manera íntegra en otro sistema.

A.1. Condiciones materiales y equipos

A continuación, se presentan todas las máquinas empleadas en el desarrollo del proyecto, indicando únicamente aquellas características relevantes para la ejecución del TFM. De esta forma, se quiere garantizar que en caso de que se replique las pruebas y validaciones del proyecto se obtengan los mismos resultados, siendo el entorno completamente replicable.

A.1.1. Especificaciones Máquina A

- Procesador: i7-8700K (12) @ 4.700GHz
- Memoria: 15674MiB
- Gráfica: Intel UHD Graphics 630
- Sistema operativo: Ubuntu 20.04.6 LTS x86_64

```
arppath@arppath-desktop:~/TFM$ neofetch
  _.-+oOSSSSsoo+-.
  `:+SSSSSSSSSSSSSSSSSS+-+
  +-SSSSSSSSSSSSSSSSSSySSSS+-+
  .OSSSSSSSSSSSSSSSSdMMMNySSSSo.
  /SSSSSSSSSSSSSSSSSSdMMMNySSSSo.
  +SSSSSSSSSSSSSSSSSShydMMMMMMNdddySSSSSS+-+
  /SSSSSSSSSSSSSSSSSShNMMMyhyyvhNMMNhSSSSSSSS/
  .SSSSSSSSdMMNNhSSSSSSSShNMMMdSSSSSSSS.
  +SSSSSSSShNMMNySSSSSSSSSSyNMMMySSSSSSSS.
  OSSyNMMMNyMMhSSSSSSSSSShMMNhSSSSSSSS.
  OSSyNMMMNyMMhSSSSSSSSSShMMNhSSSSSSSS.
  +SSSSSSSShhyNMMNySSSSSSSSSSyNMMMySSSSSS+-+
  .SSSSSSSSdMMNNhSSSSSSSShNMMNhSSSSSSSS.
  /SSSSSSSShNMMMyhyyvhNMMNhSSSSSSSS/
  +SSSSSSSSSSSSSShydMMMMMMNdddySSSSSS+-+
  /SSSSSSSSSSSSSSSSdMMMNySSSSo.
  .OSSSSSSSSSSSSSSSSdMMMNySSSSo.
  +-SSSSSSSSSSSSSSSSySSSS+-+
  :+SSSSSSSSSSSSSSSS+-+
  .-/+oOSSSSsoo+-.

arppath@arppath-desktop:~/TFM$
```

Figura A.1: Especificaciones de la máquina A

A.1.2. Especificaciones Máquina B

- Procesador: Intel i5-7500 (4) @ 3.800GHz
 - Memoria: 31984MiB
 - Gráfica: Intel HD Graphics 630
 - Sistema operativo: Ubuntu 22.04.2 LTS x86_64

Figura A.2: Especificaciones de la máquina B

A.1.3. Especificaciones Máquina C

- Procesador: Intel(R) Core(TM) 12th Gen i7-1260P (16) CPU @ 4.70Ghz
 - Memoria: 15674MiB
 - Gráfica: Intel Alder Lake-P
 - Sistema operativo: Ubuntu 22.04.2 LTS x86_64

Figura A.3: Especificaciones de la máquina C

B. Anexo II - Presupuesto

En este anexo se quiere hacer una aproximación del presupuesto que tendría el proyecto llevado a cabo. Para ello, debemos hacer recuento de cuantas horas efectivas se han dedicado al proyecto, dado que, si bien es cierto que el TFM se ha extendido el doble del tiempo que se había estimado, no se ha dedicado el doble de semanas para la realización del mismo, dado que se ha ido compaginando con otros proyectos en paralelo. También hay que tener en cuenta los medios que se han utilizado para el desarrollo, clasificándolos en elementos *hardware* y en elementos *software*.

B.1. Duración del proyecto

Según se ha comentado anteriormente, con el propósito de obtener el número de horas **efectivas** dedicadas al proyecto, se va a realizar un diagrama de Gantt. De esta forma se pretende aclarar cuantas semanas se han dedicado a cada etapa del proyecto, y de este modo, poder llegar a estimar un número de horas efectivas de trabajo.

Se quiere aclarar, que la duración del proyecto se ha extendido en el tiempo casi un año más de lo previsto, sin embargo, esta extensión temporal no ha sido completamente efectiva sobre el proyecto, dado que se ha ido compaginando con otros proyectos en paralelo. Se quiere de esta forma obtener, aproximadamente, una estimación temporal en semanas de cuanto ha llevado cada etapa del TFM.

Como se puede ver en la figura B.1, la etapa crítica de este proyecto ha sido el aprendizaje del funcionamiento interno del BOFUSS y el desarrollo de las modificaciones sobre el mismo. Si bien es cierto que la curva de aprendizaje del BOFUSS es complicada, una de las grandes causas en la duración de dichas etapas ha sido la paralelización de este trabajo con otros proyectos, gastando bastante tiempo en los cambios de contexto entre ellos.

Número de horas totales	Horas efectivas	Horas efectivas por semana
2000h	≈ 1200 - 1000h	≈ 20h

Tabla B.1: Promedio de horas de trabajo

B.2. Costes del proyecto

Para realizar el cálculo de los costes del proyecto, se llevará a cabo una diferenciación previa en términos de *hardware*, *software* y mano de obra. Esta metodología permitirá un desglose detallado de los costes, lo que brindará una mayor claridad en cuanto a la cuantía total del proyecto. En primer lugar, se considerarán los costes relacionados con el *hardware*. Esto implica evaluar los gastos asociados a la adquisición de equipos, dispositivos y componentes físicos necesarios para el desarrollo y funcionamiento del proyecto. En segundo lugar, se analizarán los costes de *software*. Esto involucra evaluar los gastos relacionados con las licencias de software, el desarrollo de aplicaciones personalizadas, la adquisición de paquetes de software especializados y los costes de mantenimiento y actualizaciones de los programas utilizados en el proyecto. Finalmente, se tendrán en cuenta los costes de mano de obra. Esto incluirá los gastos relacionados con los recursos humanos involucrados en el proyecto, como los salarios de los empleados. Es importante destacar que al desglosar los costes del proyecto de esta manera, se proporcionará una visión más completa y detallada de los recursos financieros requeridos en cada área. Esto permitirá una mejor planificación, seguimiento y control de los costes de cara a futuro en el desarrollo de un proyecto de mismas características.

Producto (IVA incluido)	Valor (€)
Ordenador portátil Asus zenbook	1560,00
Servidor A	2100,00
Servidor B	1700,00
Pantalla Lenovo L27i	130,00
Pantalla Benq 21"	80,00
RPi 4 (2 uds)	250,00
Periféricos	300,00
Infraestructura de Red (APs y Router Asus)	250,00

Tabla B.2: Presupuesto desglosado del Hardware para el TFM

Las licencias de software generalmente se venden por años, o por meses. Por ello, se ha calculado el precio equivalente asociado a la duración del TFM.

Producto (IVA incluido)	Valor (€)
Microsoft Office	300,00
Adobe Suite	500,00
Overleaf	40,00
G suite	120,00

Tabla B.3: Presupuesto desglosado del Software para el TFM

Para determinar los costes de mano de obra, se han utilizado como referencia los honorarios de un ingeniero senior, los cuales ascienden a 30€ por hora. Estos honorarios se aplicarán en función de la cantidad de horas efectivas dedicadas al proyecto. En cuanto a los costes relacionados con el *hardware* y el *software*, se han agrupado como un único elemento dentro del presupuesto. Se ha considerado el valor total resultante del desglose de los productos indicados en el análisis. Al agregar estos costes de *hardware* y *software* al presupuesto, se obtiene el valor total necesario para cubrir estos componentes esenciales del proyecto. Es importante destacar que estos cálculos se basan en las referencias proporcionadas y están sujetos a ajustes según las necesidades específicas del proyecto y las tarifas y precios aplicables en cada caso particular. El desglose y la inclusión de estos costes en el presupuesto garantizan una consideración adecuada de los recursos necesarios para el éxito y la ejecución del proyecto.

Descripción (IVA incluido)	Unidades	Coste unitario (€)	Coste total (€)
Material Hardware	1	6370,00	6370,00
Material Software	1	960,00	960,00
Mano de obra	1000	30,00	30000,00
Costes fijos (Luz, Internet)	1	650,00	650,00
TOTAL			37.980,00 €

Tabla B.4: Presupuesto total con IVA

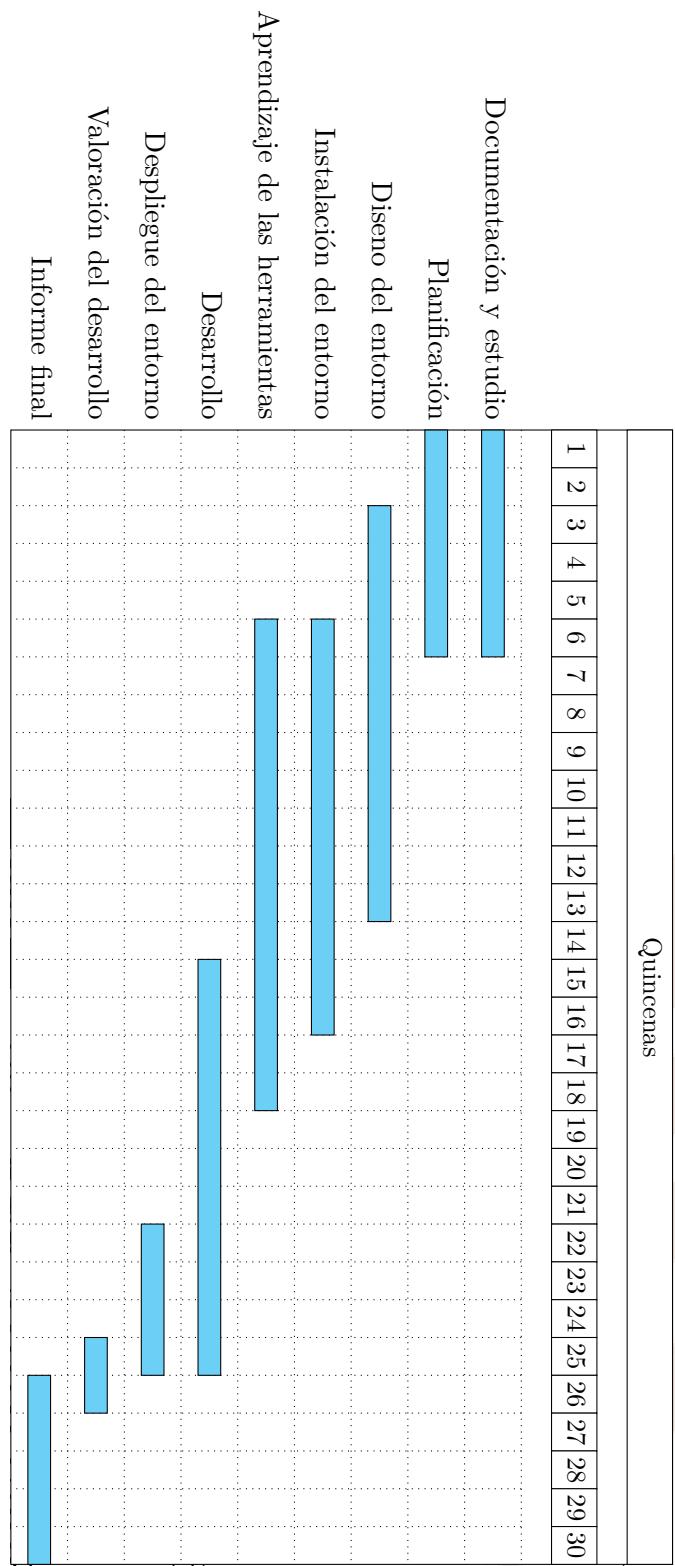


Figura B.1: Diagrama de Gantt del proyecto

C. Anexo III - Manuales de usuario e instalación

En este anexo se proporcionarán los manuales de usuario e instalación de las herramientas necesarias para el desarrollo y la verificación del funcionamiento del TFM. Además, se explicará el funcionamiento de los scripts de instalación generados, facilitando así el acceso a cualquier persona interesada en replicar los diferentes casos de uso. Los manuales de usuario detallarán los pasos necesarios para utilizar cada herramienta, incluyendo instrucciones de instalación, configuración y operación. Se proporcionarán ejemplos prácticos y se explicarán los conceptos clave relacionados con el uso de estas herramientas.

Asimismo, se incluirá información sobre los scripts de instalación desarrollados, que permiten una instalación rápida y sencilla de las herramientas y componentes necesarios para reproducir los casos de uso del TFM. Estos scripts automatizan gran parte del proceso de instalación y configuración, lo que facilita a los usuarios la puesta en marcha de los entornos necesarios para el desarrollo y la evaluación.

C.1. Herramienta iproute2

En el contexto del desarrollo y validación del protocolo desarrollado, la herramienta iproute2 desempeñará un papel fundamental en la configuración de las interfaces inalámbricas emuladas en el kernel, así como en la consulta del estado de estas interfaces y la verificación de la ubicación del usuario dentro de los *Network namespaces*. Por tanto, se considera relevante incluir esta sección, ya que iproute2 se convertirá en una de las herramientas clave para la gestión de los *Network namespaces* y la verificación de los casos de uso en la validación del protocolo desarrollado.

iproute2 es una suite de herramientas de línea de comandos que proporciona funcionalidades avanzadas de gestión de redes en sistemas Linux. Su amplio conjunto de comandos y opciones permite realizar diversas tareas relacionadas con la configuración y administración de interfaces de red, enrutamiento, gestión de tablas de enrutamiento, configuración de políticas de tráfico y mucho más. En el contexto de este proyecto, se utilizará iproute2

para realizar acciones específicas, como configurar las interfaces inalámbricas emuladas en el kernel, verificar el estado de las interfaces y determinar en qué *Network namespace* se encuentra el usuario. La inclusión de esta sección permitirá a los lectores familiarizarse con las funcionalidades básicas de iproute2 que serán relevantes para el desarrollo y verificación de los casos de uso. A lo largo de esta sección, se presentarán los comandos más utilizados y se proporcionarán ejemplos prácticos de su aplicación en el contexto del proyecto. Además, se recomendará al lector que consulte la documentación oficial de iproute2 para obtener una comprensión más completa de las capacidades y opciones avanzadas de esta herramienta.

C.1.1. ¿Qué es iproute2?

Iproute2 es un conjunto de herramientas de utilidad para la gestión de redes en sistemas Linux. Se ha convertido en una solución ampliamente adoptada y está presente en la mayoría de las distribuciones actuales. El proyecto es mantenido por Alexey Kuznetsov y Stephen Hemminger, quienes han sido los principales desarrolladores del software. Además, iproute2 es un proyecto de código abierto en el cual contribuyen activamente cientos de personas a través de su repositorio en GitHub¹.

La versión más reciente de iproute2 es la v6.4.0, la cual se utilizará en el entorno de Ubuntu 18.04 para este proyecto. Este paquete de herramientas está diseñado para reemplazar a las utilidades contenidas en el paquete **net-tools**, como **ifconfig**, **route**, **netstat**, **arp**, entre otras. En la Tabla C.1 se presenta una comparativa de las herramientas equivalentes de net-tools y las correspondientes en iproute2. La amplia adopción de iproute2 se debe a su flexibilidad y capacidad para ofrecer una amplia gama de funcionalidades para la gestión de redes en sistemas Linux. Las herramientas incluidas en iproute2 permiten configurar interfaces de red, manipular tablas de enrutamiento, establecer políticas de tráfico, supervisar el estado de las interfaces y mucho más. Su diseño modular y su enfoque en la eficiencia y el rendimiento hacen de iproute2 una opción preferida para administrar y diagnosticar redes en sistemas Linux.

net-tools	iproute2
arp	ip neigh
ifconfig	ip link
ifconfig -a	ip addr
iptunnel	ip tunnel
route	ip route

Tabla C.1: Herramientas de Iproute2 frente a las de net-tools

¹<https://github.com/shemminger/iproute2>

C.1.2. ¿Por qué necesitamos iproute2?

La justificación de utilizar el paquete de herramientas de red iproute2 para gestionar las interfaces veth, las interfaces inalámbricas emuladas del módulo del kernel `mac80211_hwsim`, las network namespaces y la gestión con interfaces se basa en varios aspectos fundamentales:

- Funcionalidad integral: iproute2 proporciona un conjunto completo de herramientas que permiten la gestión y configuración avanzada de las interfaces de red. Esto incluye la creación, eliminación, configuración de direcciones IP, asignación de rutas, configuración de parámetros y mucho más. Al utilizar iproute2, se tiene acceso a todas estas capacidades en un solo paquete, lo que facilita la administración eficiente de las interfaces requeridas en el proyecto.
- Flexibilidad y compatibilidad: iproute2 es una solución ampliamente adoptada en sistemas Linux y está disponible en la mayoría de las distribuciones actuales. Al utilizar iproute2, nos aseguramos de tener una herramienta compatible y probada que nos permita gestionar las interfaces veth y las interfaces inalámbricas emuladas del módulo `mac80211_hwsim`. Además, iproute2 es compatible con las network namespaces, lo que nos facilita la gestión y operativa de estos entornos de red aislados.
- Capacidad para la configuración avanzada: iproute2 proporciona opciones de configuración avanzadas que permiten un control preciso sobre las interfaces de red. Por ejemplo, podemos utilizar iproute2 para establecer políticas de tráfico, aplicar reglas de enrutamiento específicas, configurar parámetros de calidad de servicio (QoS) y más. Esto es especialmente importante en el contexto de este proyecto, donde se requiere una gestión detallada y precisa de las interfaces veth y las interfaces inalámbricas emuladas.

C.1.3. Compilación e instalación de iproute2

El proceso de compilación e instalación de la herramienta iproute2 es prácticamente análogo tanto en Ubuntu 20.04 como en Ubuntu 22.04, con una única diferencia que se mencionará más adelante. Antes de proceder con la compilación, es necesario realizar una configuración previa instalando los paquetes necesarios. Estos paquetes proporcionarán las dependencias requeridas para el proceso de compilación y garantizarán un entorno adecuado para la instalación de iproute2. Entre los paquetes necesarios se encuentran:

- `bison`: una herramienta generadora de analizadores sintácticos de propósito general.
-

- **flex**: una herramienta para generar programas que reconocen patrones léxicos en el texto.
- **libmnl-dev**: una librería de espacio de usuario orientada a los desarrolladores de Netlink. Netlink es una interfaz entre el espacio de usuario y el espacio del kernel vía sockets.
- **libdb5.3-dev**: un paquete de desarrollo que contiene los archivos de cabecera y librerías estáticas necesarias para la base de datos de Berkley (*Key/Value*).

Es importante asegurarse de tener instalado el paquete `wget` para poder descargar la herramienta iproute2. En caso de no tenerlo, se debe instalar previamente. Una vez que los paquetes necesarios estén instalados, se puede proceder con el proceso de compilación e instalación de iproute2.

Código C.1: Instalación de las dependencias de Iproute2

```
1 sudo apt-get install bison flex libmnl-dev libdb5.3-dev
```

- En segundo lugar, se debe descargar el comprimido de la herramienta iproute2. Al haber varios paquetes, se descargará aquel cuya versión sea con la que se quiere trabajar. Puedes descargarlo desde el siguiente enlace: kernel.org.

Código C.2: Obtención del source de Iproute2

```
1 wget -c http://ftp.iij.ad.jp/pub/linux/kernel/linux/utils/net/iproute2/iproute2-4.15.0.tar.gz
```

- En tercer lugar, se debe descomprimir el comprimido de la herramienta. A continuación, se procederá a configurar, compilar e instalar la herramienta.

Código C.3: Compilación e instalación de Iproute2

```
1 # Descomprimir y entrar al directorio
2 tar -xvfz $(tar).tar.gz && cd $tar
3
4 # Configurar la herramienta
5 ./configura
6
7 # Compilar e instalar, para añadir el nuevo binario al PATH
8 sudo make
9 sudo make install
```

C.1.4. Comandos útiles con iproute2

A continuación, se presentan los comandos más frecuentemente utilizados con la herramienta iproute2. Estos comandos han sido ampliamente empleados a lo largo del desarrollo del proyecto y en el proceso de verificación de los distintos casos de uso. Por tanto, se considera que esta sección resultará de gran utilidad para los lectores que no estén familiarizados con esta herramienta y deseen aprender cómo utilizarla de manera efectiva.

Código C.4: Comandos útiles con iproute2

```
1 # Listar interfaces y visualizar las direcciones asignadas
2 ip addr show
3
4 # Asignar o eliminar una dirección a una interfaz
5 ip addr add {IP} dev {interfaz}
6 ip addr del {IP} dev {interfaz}
7
8 # Activar o desactivar una interfaz
9 ip link set {interfaz} up/down
10
11 # Listar las rutas de red
12 ip route list
13
14 # Obtener la ruta para una dirección IP específica
15 ip route get {IP}
16
17 # Listar los Network namespaces con sus nombres asignados
18 ip netns list
19
20 # Si lees esto, te debo un besito :)
```

Estos comandos proporcionan funcionalidades clave para la administración y configuración de interfaces de red, rutas de red y namespaces de red. Al utilizarlos de manera adecuada, los administradores de redes pueden llevar a cabo tareas importantes, como la asignación de direcciones IP, la habilitación o deshabilitación de interfaces, el enrutamiento de paquetes y la gestión de entornos de red aislados. Por lo tanto, comprender y dominar estos comandos resulta fundamental para cualquier profesional o entusiasta de las redes que desee tener un control preciso sobre su infraestructura de red.

C.2. Herramienta tcpdump

La inclusión de esta sección se motiva por proporcionar un punto de referencia para aquellos usuarios que no están familiarizados con tcpdump. A lo largo de todas las secciones del proyecto, se utilizará esta herramienta para verificar el correcto funcionamiento de los casos de uso. El objetivo de esta sección es ofrecer una guía práctica sobre cómo utilizar tcpdump, especialmente dirigida a aquellos usuarios que no tienen experiencia previa con la herramienta. Se proporcionarán instrucciones detalladas sobre cómo instalar y ejecutar tcpdump, así como ejemplos de comandos comunes para capturar y analizar el tráfico de red.

Al comprender cómo utilizar tcpdump, los usuarios podrán verificar de manera efectiva si los casos de uso implementados en el proyecto están funcionando según lo esperado. Esta herramienta les permitirá examinar el tráfico de red en tiempo real y obtener información relevante para la evaluación y solución de problemas.

C.2.1. ¿Qué es tcpdump?

Tcpdump es un analizador de tráfico diseñado para inspeccionar los paquetes entrantes y salientes de una interfaz. La peculiaridad de esta herramienta es que funciona mediante línea de comandos y tiene soporte en la mayoría de los sistemas UNIX², como Linux, macOS y OpenWrt. La herramienta está escrita en lenguaje C, lo que le confiere un alto rendimiento, y utiliza libpcap³ para la captura de paquetes.

Fue desarrollada en 1988 por trabajadores de los laboratorios de Berkeley. En la actualidad, cuenta con una gran comunidad de desarrolladores respaldándola en su repositorio oficial⁴, donde se lanzan periódicamente nuevas actualizaciones (última versión v4.99.4).

C.2.2. ¿Por qué necesitamos tcpdump?

En la actualidad, es común desarrollar en entornos distribuidos, utilizando contenedores o máquinas virtuales para delimitar el entorno de desarrollo. Por esta razón, se trabaja de forma remota, conectándose a la máquina o contenedor mediante el protocolo SSH⁵.

Esto conlleva numerosas ventajas, pero también puede presentar complicaciones. Por ejemplo, si alguien no sabe cómo configurar un *X Server* para ejecutar aplicaciones gráficas de forma remota, no podría utilizar herramientas como Wireshark. En este sentido, tcpdump

²Unix es un sistema operativo desarrollado en 1969 por un grupo de empleados de los laboratorios Bell

³<https://github.com/the-tcpdump-group/libpcap>

⁴<https://github.com/the-tcpdump-group/tcpdump>

⁵<https://www.ssh.com/ssh/>

resulta especialmente útil, ya que no requiere ninguna configuración adicional para ejecutarse de forma remota. Además, su rápido inicio, en comparación con otros “sniffers” como Wireshark, ha convertido a tcpdump en una herramienta fundamental para verificar casos de uso.

C.2.3. Instalación de tcpdump

Como se mencionó anteriormente, tcpdump cuenta con un amplio soporte en sistemas UNIX, por lo que suele estar preinstalado en la mayoría de las distribuciones Linux. Sin embargo, en caso de no tenerlo instalado, es posible instalarlo fácilmente siguiendo los pasos que se muestran a continuación (Ver bloque C.5). Estos comandos actualizarán los repositorios de paquetes y descargarán e instalarán tcpdump en el sistema. Una vez completada la instalación, tcpdump estará listo para su uso.

Código C.5: Instalación del paquete Tcpdump

```
1 sudo apt install -y tcpdump
```

C.2.4. Comandos útiles con tcpdump

A continuación, se presentan algunos comandos comunes que se han utilizado durante el proceso de verificación de los casos de uso con la herramienta tcpdump. Estos comandos han sido seleccionados debido a su relevancia y utilidad, y se considera que pueden ser de gran ayuda para aquellos lectores que no estén familiarizados con esta herramienta. No obstante, se recomienda encarecidamente que el lector consulte la página de manual de tcpdump⁶ para obtener información más detallada sobre el uso básico de tcpdump y explorar todas sus capacidades. Estos son solo algunos ejemplos de los comandos más utilizados con tcpdump. Como se mencionó anteriormente, se recomienda encarecidamente al lector que consulte la página de manual de tcpdump para obtener una comprensión más completa de sus capacidades y explorar otros filtros y opciones disponibles.

Código C.6: Comandos útiles con Tcpdump

```
1 # Indicar sobre que Interfaz se quiere sniffear
2 tcpdump -i {Interfaz}
3
4 # Podemos almacenar la traza a un archivo para su posterior análisis
5 tcpdump -w fichero.pcap -i {Interfaz}
6
7 # También podemos ver una traza desde un archivo
8 tcpdump -r fichero.pcap
9
```

⁶<https://linux.die.net/man/8/tcpdump>

```
10 # Podemos filtrar por puerto
11 tcpdump -i {Interfaz} port {Puerto}
12
13 # Podemos filtrar por dirección IP destino/origen
14 tcpdump -i {Interfaz} dst/src {IP}
15
16 # Podemos filtrar por protocolo
17 tcpdump -i {Interfaz} {protocolo}
18
19 # Listar interfaces disponibles para escuchar de ellas
20 tcpdump -D
21
22 # Limitar el número de paquetes a sniffear
23 tcpdump -i {Interfaz} -c {Número de paquetes}
```

```
n0obie@n0obie-VirtualBox:~$ sudo tcpdump -i enp0s3 -c1
[sudo] contraseña para n0obie:
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp0s3, link-type EN10MB (Ethernet), capture size 262144 bytes
20:25:55.733209 ARP, Request who-has LAPTOP-7E128QAP.home tell liveboxfibra, length 46
1 packet captured
5 packets received by filter
0 packets dropped by kernel
n0obie@n0obie-VirtualBox:~$ █
```

Figura C.1: Interfaz de tipo CLI de Tcpdump

C.3. Herramienta Mininet

La motivación de añadir esta sección es la de poder indicar al lector como se ha instalado la herramienta de Mininet en las distintas máquinas descritas en el pliego de condiciones (Anexo A). Todo el proceso de instalación se ha llevado acabo en una máquina Linux, con las especificaciones indicadas en la siguiente tabla (Tabla C.2).

Distribución Linux	Cores	Memoria	Disco
Ubuntu 22.04 LTS	2	4096 MiB	40 GiB

Tabla C.2: Especificaciones máquina de instalación Mininet

Para la instalación necesitaremos de conectividad a Internet y de la herramienta `git` para clonar el repositorio de `Mininet`. En caso de no tener la herramienta de `git`, podremos instalarla ejecutando el comando del bloque de código C.7.

Código C.7: Instalación de la herramienta git

```
1 # El parametro -y se indica para confirmar la instalación de la herramienta
2 sudo apt install -y git
```

Una vez disponemos de la herramienta `git` para clonar el repositorio de `Mininet`, vamos a clonarlo, y lanzar el script de instalación que incorpora junto al source para instalar la herramienta. A continuación, en el bloque de código C.8 se indica los comandos ejecutados para instalar la herramienta de `Mininet`.

Código C.8: Instalación de la herramienta Mininet

```
1 # Clonamos el repositorio de Mininet
2 git clone https://github.com/davidcawork/mininet.git
3
4 # Accedemos al directorio
5 cd mininet
6
7 # Lanzamos el script de instalación (Openflow 1.3 - Ryu - Wireshark dissector)
8 sudo util/install.sh -3fmnyv
```

C.4. Herramienta Mininet-WiFi

La motivación de añadir esta sección es la de poder indicar al lector como se ha instalado la herramienta de Mininet en las distintas máquinas descritas en el pliego de condiciones (Anexo A). Todo el proceso de instalación se ha llevado acabo en una máquina Linux, con las especificaciones indicadas en la siguiente tabla (Tabla C.3).

Distribución Linux	Cores	Memoria	Disco
Ubuntu 22.04 LTS	2	4096 MiB	40 GiB

Tabla C.3: Especificaciones máquina de instalación Mininet-WiFi

Para la instalación necesitaremos de conectividad a Internet y de la herramienta `git` para clonar el repositorio de Mininet-WiFi. En caso de no tener la herramienta de `git`, podremos instalarla ejecutando el comando del bloque de código C.7. Una vez disponemos de la herramienta `git` para clonar el repositorio de Mininet-WiFi, vamos a clonarlo, y lanzar el script de instalación que incorpora junto al source para instalar la herramienta. A continuación, en el bloque de código C.9 se indica los comandos ejecutados para instalar la herramienta de Mininet-WiFi.

Código C.9: Instalación de la herramienta Mininet-WiFi

```

1 # Clonamos el repositorio de Mininet
2 git clone https://github.com/davidcawork/mininet.git
3
4 # Accedemos al directorio
5 cd mininet-wifi
6
7 # Lanzamos el script de instalación (Openflow 1.3 - Ryu - Wireshark dissector)
8 sudo ./util/install.sh -3Wlfnv
9
10 # Para esta versión que instala vagrant de ubuntu el kernel que trae
11 # no lleva el modulo mac80211_hwsim por tanto hay que añadirlo, si tu version tiene
12 # compilado el modulo del kernel mac80211_hwsim, no hará falta hacer esto
13 sudo apt-get install -y linux-modules-extra-5.15.0-69-generic

```

C.5. Herramienta Ryu

La motivación de añadir esta sección es la de poder indicar al lector como se ha instalado la herramienta de Ryu en las distintas máquinas descritas en el pliego de condiciones (Anexo A). Todo el proceso de instalación se ha llevado acabo en una máquina Linux, con las especificaciones indicadas en la siguiente tabla (Tabla C.4).

Distribución Linux	Cores	Memoria	Disco
Ubuntu 22.04 LTS	2	4096 MiB	40 GiB

Tabla C.4: Especificaciones máquina de instalación de Ryu

Para la instalación necesitaremos de conectividad a Internet y de la herramienta `git` para clonar el repositorio de Ryu. En caso de no tener la herramienta de `git`, podremos instalarla ejecutando el comando del bloque de código C.7. Una vez disponemos de la herramienta `git` para clonar el repositorio de Ryu, vamos a clonarlo, y lanzar el script de instalación que incorpora junto al source para instalar la herramienta. A continuación, en el bloque de código C.10 se indica los comandos ejecutados para instalar la herramienta de Ryu.

Código C.10: Instalación de la herramienta Ryu

```

1  #!/bin/bash
2
3  echo "[+] Installing Ryu..."
4
5  INSTALL='sudo DEBIAN_FRONTEND=noninteractive apt-get -y -q install'
6  BUILD_DIR=${HOME}
7
8  # Update
9  sudo apt-get update
10
11 # Install Ryu dependencies
12 $INSTALL autoconf automake git g++ libtool python3 make gcc python3-pip python3-dev
13   ↪ libffi-dev libssl-dev libxml2-dev libxslt1-dev zlib1g-dev
14
15 # Fetch RYU
16 cd $BUILD_DIR/
17 git clone https://github.com/davidcawork/ryu.git ryu
18 cd ryu
19
20 # Install ryu
21 sudo pip3 install -r tools/pip-requires -r tools/optional-requires \
22   -r tools/test-requires
23 sudo python3 setup.py install

```

C.6. Herramienta ONOS

La motivación de añadir esta sección es la de poder indicar al lector como se ha instalado la herramienta de ONOS en las distintas máquinas descritas en el pliego de condiciones (Anexo A). Todo el proceso de instalación se ha llevado acabo en una máquina Linux, con las especificaciones indicadas en la siguiente tabla (Tabla C.5).

Distribución Linux	Cores	Memoria	Disco
Ubuntu 22.04 LTS	2	4096 MiB	40 GiB

Tabla C.5: Especificaciones máquina de instalación de ONOS

Para la instalación necesitaremos de conectividad a Internet y de la herramienta `git` para clonar el repositorio de ONOS y las herramientas asociadas. En caso de no tener la herramienta de `git`, podremos instalarla ejecutando el comando del bloque de código C.7. Una vez disponemos de la herramienta `git` para clonar el repositorio de ONOS, vamos a clonarlo, y lanzar el script de instalación que incorpora junto al source para instalar la herramienta. A continuación, en el bloque de código C.11 se indica los comandos ejecutados para instalar la herramienta de ONOS.

Código C.11: Instalación de la herramienta ONOS

```

1  #!/bin/bash
2
3  sudo apt update -y
4  sudo apt install -y git g++ unzip zip openjdk-11-jdk bzip2
5
6  wget -c https://github.com/bazelbuild/bazel/releases/download/6.0.0-pre.20220421.3/←
    ↪ bazel-6.0.0-pre.20220421.3-installer-linux-x86_64.sh
7  chmod +x bazel-6.0.0-pre.20220421.3-installer-linux-x86_64.sh
8  ./bazel-6.0.0-pre.20220421.3-installer-linux-x86_64.sh --user
9  export PATH=$PATH:$HOME/bin
10
11
12 git clone https://gerrit.onosproject.org/onos -b onos-2.5
13 echo "#ONOS" | tee -a ~/.bashrc
14 echo ". ~/onos/tools/dev/bash_profile" | tee -a ~/.bashrc
15 source .bashrc
16 cd onos && bazel build onos

```


Universidad de Alcalá
Escuela Politécnica Superior



ESCUELA POLITECNICA
SUPERIOR

