

Universidad de Alcalá

Escuela Politécnica Superior

Máster Universitario en Ingeniería de Telecomunicación

Trabajo Fin de Máster

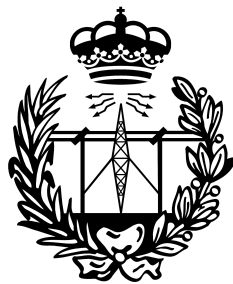
Diseño e implementación de protocolo
de control escalable en redes IoT para
entornos 6G

ESCUELA POLITECNICA
SUPERIOR

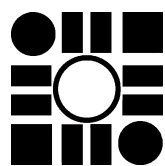
Autor: David Carrascal Acebron

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 mi, y aun 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 mi. Y que 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, Bobby, 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 alentan 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 Final 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, the sixth generation of mobile technologies (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 por analizar las necesidades y características de las distintas tecnologías que se emplearán en ejecución del objetivos predefinidos y así discernir aquellas herramientas necesarias para la implementación del protocolo control. Una vez seleccionadas las herramientas, se estudiarán a fondo para realizar una implementación lo optimizada en la medida de lo posible. Este proyecto concluirá 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 for 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.

In order to fulfil the project, we have started by analysing the requirements and characteristics of the different technologies that will be used in the execution of the predefined objectives and thus be able to 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 will conclude 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 que puerto se dirige”

Arthur Schopenhauer.

Índice general

Resumen	v
Abstract	vii
1. Diseño del protocolo de control In-Band	1
1.1. Protocolo In-Band	1
1.1.1. Protocolo IoTorii	2
1.1.1.1. Operativa del protocolo IoTorii	2
1.1.1.2. Configuración del protocolo IoTorii	4
1.2. Plataforma de desarrollo y validación	4
1.3. Agente SDN	7
1.4. Agente de control SDN	9
1.5. Análisis funcional de la interfaz del BOFUSS	10
1.5.1. Binario <code>ofprotocol</code>	11
1.5.2. Binario <code>ofdatapath</code>	12
1.6. Análisis de la clase <code>UserAP</code> en Mininet-WiFi	14
1.7. Análisis del entorno de depuración del BOFUSS	22
1.7.1. Limpieza del escenario	23
1.7.2. Puesta en marcha del escenario	24
1.7.2.1. Troubleshooting	26
1.7.3. Configuración de VS Code	27
Bibliografía	31
Lista de Acrónimos y Abreviaturas	33
A. Anexo I - Pliego de condiciones	35
A.1. Condiciones materiales y equipos	35
A.1.1. Especificaciones Máquina A	35
A.1.2. Especificaciones Máquina B	36
A.1.3. Especificaciones Máquina C	37

Índice de figuras

1.1. Operativa del protocolo de IoTorii [1]	3
1.2. Mensajes de control en IoTorii [1]	5
1.3. Entorno de emulación real de una cabina de avión a escala 1:1 [3]	6
1.4. Ejecución del binario <code>ofdatapath</code> en modo standalone	13
1.5. Comprobación del puerto de escucha del binario <code>ofdatapath</code>	14
1.6. Diagrama UML de la clase <code>UserAP</code>	16
1.7. Topología básica haciendo uso del <code>UserAP</code> (Basic OpenFlow User-space Software Switch (BOFUSS))	17
1.8. Proceso de debug al BOFUSS	29
1.9. Listado de interfaces inalámbricas en <code>/sys/kernel/debug/ieee80211</code>	30
1.10. Comprobación de si el módulo <code>mac80211_hwsim</code> está cargado	30
1.11. Listado de <i>phy</i> inalámbricas usando el comando <code>iw</code>	30
A.1. Especificaciones de la máquina A	35
A.2. Especificaciones de la máquina B	36
A.3. Especificaciones de la máquina C	37

Índice de tablas

1.1. Comparativa del OvS con el BOFUSS	8
1.2. Comparativa del controlador ONOS con el controlador Ryu	10

Índice de Códigos

1.1. Interfaz CLI del binario ofprotocol	11
1.2. Interfaz CLI del binario ofdatapath	12
1.3. Puesta en marcha del escenario básico	15
1.4. Traza de la puesta en marcha del escenario básico	17
1.5. Puesta en marcha del BOFUSS	21
1.6. Script de limpieza del escenario - clean.sh	23
1.7. Script de puesta en marcha del escenario - launch.sh	24
1.8. Operativa básica de la herramienta hwsim_mgmt	26
1.9. Bloqueo de la interfaz por RF-Kill	26
1.10. desbloqueo de la interfaz por RF-Kill	27
1.11. JSON de depuración con GDB del BOFUSS	27

1. 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. En este capítulo, se realizará un exhaustivo análisis de soluciones anteriores basadas en el enfoque In-Band, donde se explorarán diferentes propuestas y se evaluarán 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.

1.1. 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 cuidadosa evaluación de las opciones disponibles, se ha decidido utilizar el protocolo IoTorii [1], 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 definidas por software (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.

1.1.1.1. 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.

1.1.1.1.1. 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 [2], el cual trataba de buscar el mismo etiquetado jerárquico pero en redes cableadas de Data Centers. El principio de estas etiquetas está basado en una revisión del estándar *ieee802* que se hizo para dar cabida al uso de direcciones MAC mejoradas para el reenvío de paquetes en red.

El proceso de la difusión de dichas etiquetas se puede ver en la figura 1.1. El primer paso es establecer en la topología quien va actuar como nodo *root*, 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. La asignación de etiquetas empezará desde el *root* haciendo uso de un mensaje de tipo *SetHLMAC*, el cual consiste en mandar la HLMAC del remitente más un sufijo que se asociará al vecino. Pero, ¿A qué consideramos vecino? La condición de vecino se da cuando dos nodos se encuentran en rango de cobertura, y se han notificado entre ellos de su presencia mediante un mensaje de tipo *Hello*. Dicho mensaje es una trama vacía donde solo anuncian su dirección MAC real.

Un simil en la vida real podría verse como cuando una nueva familia llega al vecindario y va casa por casa presentandose, “¡Hola somos nuevos aquí 😊! Acabamos de llegar al vecindario, nos hemos mudado a la casa de enfrente”. Sin embargo, no te sueles presentar con todas las personas del vecindario, solo con los vecinos más cercanos, por que no te interesa que todos sepan de ti, solo aquellos con los que vayas a tener relación.

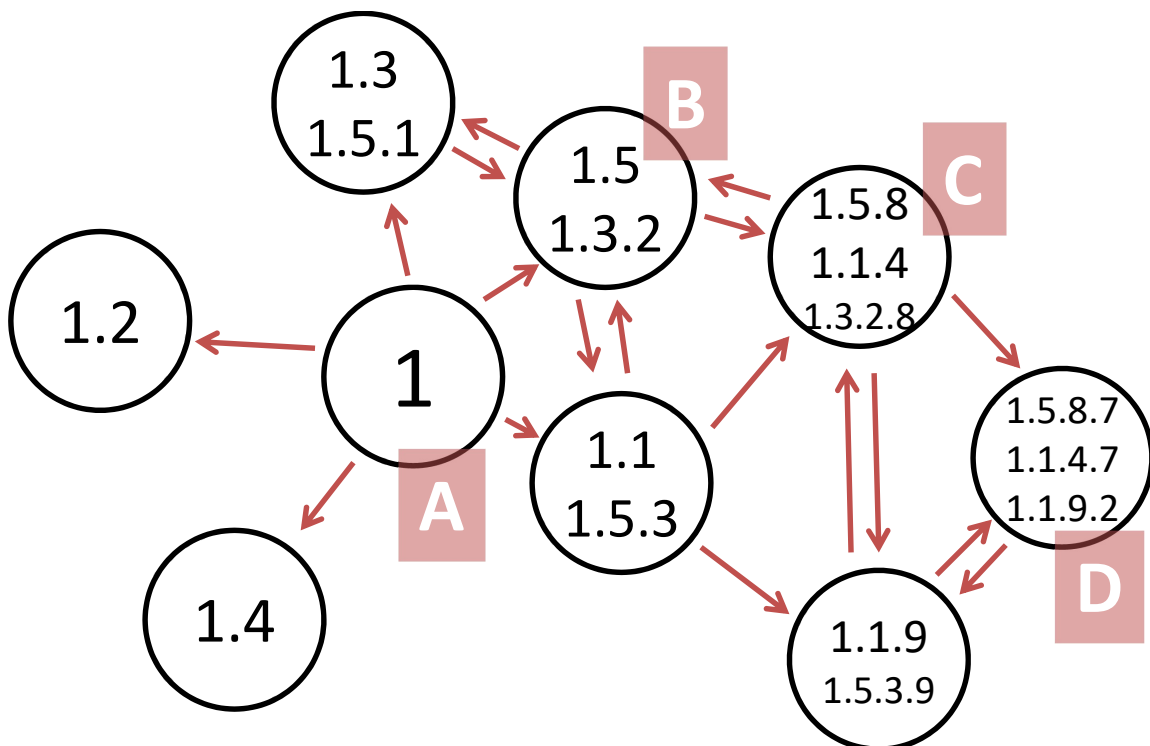


Figura 1.1: Operativa del protocolo de IoTorii [1]

Algorithm 1: Assignment of HLMACs in IoTorii

```

1 send 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

```

1.1.1.2. Configuración del protocolo IoTorii

1.2. Plataforma de desarrollo y validación

En este punto se va a valorar que plataforma se va a utilizar para desarrollar y validar el protocolo de comunicación in-band. Lo primera cuestión que se tiene que tener en cuenta cuando se va a elegir una plataforma de este tipo es, ¿Vamos a simular o a emular?. Aunque mucha gente considera que estos términos son equivalentes, no son lo mismo. Cuando habla-

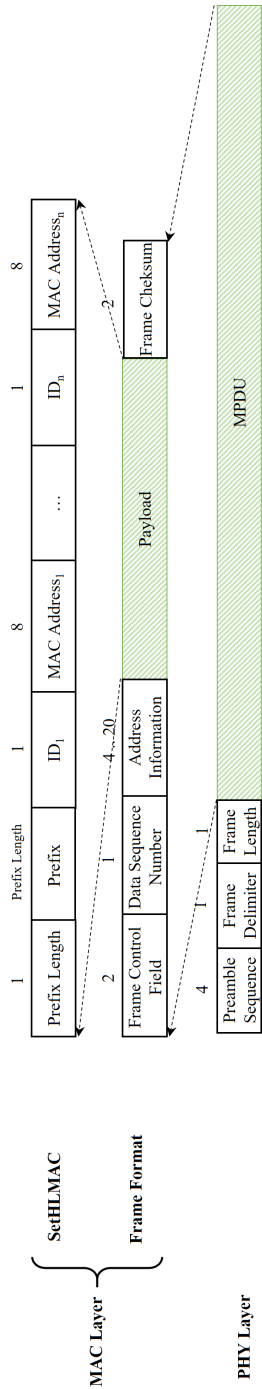


Figura 1.2: Mensajes de control en IoTorii [1]

mos 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 1.3) con controles reales estaríamos hablando entonces de emulación.



Figura 1.3: Entorno de emulación real de una cabina de avión a escala 1:1 [3]

Una vez que ya tenemos claro cuales son las diferencias sobre la emulación y la simulación, vamos a ver que opciones tenemos en ambos ambitos para elegir una plataforma que se adecue a las necesidades del proyecto. De entre todas las opciones barajadas nos hemos quedados 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.

¹<https://www.flightsimulator.com/>

- Contiki-ng haciendo uso de su simulador integrado llamado Cooja, explicado anteriormente en la sección ?? del estado del arte.
- Mininet-WiFi, explicado anteriormente en la sección ?? 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 [4, 5]. 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 o a su variante también IoT. Trabajar con una variante u otra 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 por que 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.

1.3. 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:

- Open vSwitch (OvS), explicado anteriormente en la sección ?? del estado del arte.
- BOFUSS, explicado anteriormente en la sección ?? 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 mainteiner.
- El OvS puede trabajar en modo standalone, 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 solido 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 rendimineto	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 1.1: Comparativa del OvS con el BOFUSS

Después de analizar las fortalezas y debilidades de cada opción, ver tabla 1.1, 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-band².

1.4. 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 ?? del estado del arte.
- Open Network Operating System (ONOS), explicado anteriormente en la sección ?? 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.

²<https://github.com/NETSERV-UAH/in-BOFUSS>

- Aunque una ventaja potencial de ONOS es su aplicación de descubrimiento topológico, que cuenta con una interfaz web bastante *fancy*, esta aplicación utiliza el protocolo Link Layer Discovery Protocol (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 [6] 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 solido 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 1.2: Comparativa del controlador ONOS con el controlador Ryu

Después de analizar las fortalezas y debilidades de cada opción, ver tabla 1.2, 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 más adecuada.

1.5. 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 responsable de procesar 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 procesamiento de paquetes. A través del parser, 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 procesamiento de paquetes, o recopilar estadísticas. Este binario es quien habilita al controlador llevar a ca-

bo una gestión centralizada y dinámica de las políticas de red, facilitando la adaptación y optimización de la infraestructura según las necesidades del entorno.

1.5.1. 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 pérdidas.

Código 1.1: 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.

1.5.2. Binario ofdatapath

La herramienta de 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 ??). 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 1.2 se indica la interfaz de comandos de este binario.

Código 1.2: 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 verdad³, es el 6653.

El valor del puerto por defecto se puede comprobar fácilmente, a continuación en las figuras 1.4 y 1.5. 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 que macro definen el puerto por defecto, lo cual se puede comprobar [aquí](#).

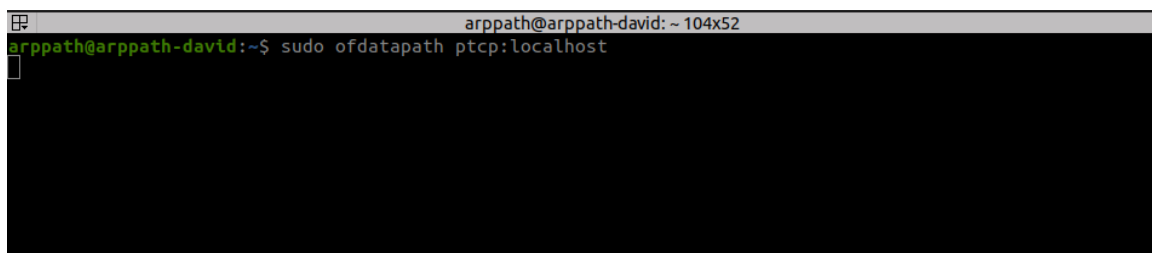


Figura 1.4: Ejecución del binario ofdatapath en modo standalone

Sigamos con los parámetros de configuración del software switch. Uno de los más importantes, es indicar los puertos a gestionar el switch. Es decir, que interfaces va a manejar.

³Se tuvo que modificar la wiki de la herramienta https://github.com/CPqD/ofsoftswitch13/wiki/Ofdatapath-Manual/_history

```

arppath@arppath-david: ~ 104x52
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 [::]: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 1.5: Comprobación del puerto de escucha del binario ofdatapath

- `-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 está opción no está indicada, por defecto, se creará una interfaz de tipo tap, tap0 o algo así, 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`. Se crea aquí. Para más información sobre las interfaces tun/tap, ver la sección ??.
- `--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.

1.6. 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 1.6, 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 Traffic control (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.

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 encontrados entre los repositorios de las plataformas de emulación y el *BOFUSS*, se debe 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 como 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 que comandos y llamadas al sistema se llevan a cabo para levantar una instancia de un software switch *BOFUSS*.

La topología que se va a desplegar se puede apreciar en la figura 1.7. Para lanzar dicha topología se tiene que lanzar un script de Python el cual se puede encontrar en el repositorio del TFM (Sección ??). 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 atraviesen su *datapath*. A continuación, en el bloque de código 1.3, se puede apreciar qué comandos se tienen que utilizar para desplegar el escenario.

Código 1.3: Puesta en marcha del escenario básico

```
1  # Lanzamos el script que pone en marcha el medio inalambrico via Mininet-WiFi
2  sudo python3 topo.py
3
4  # Lanzamos el controlador (en otra terminal)
```

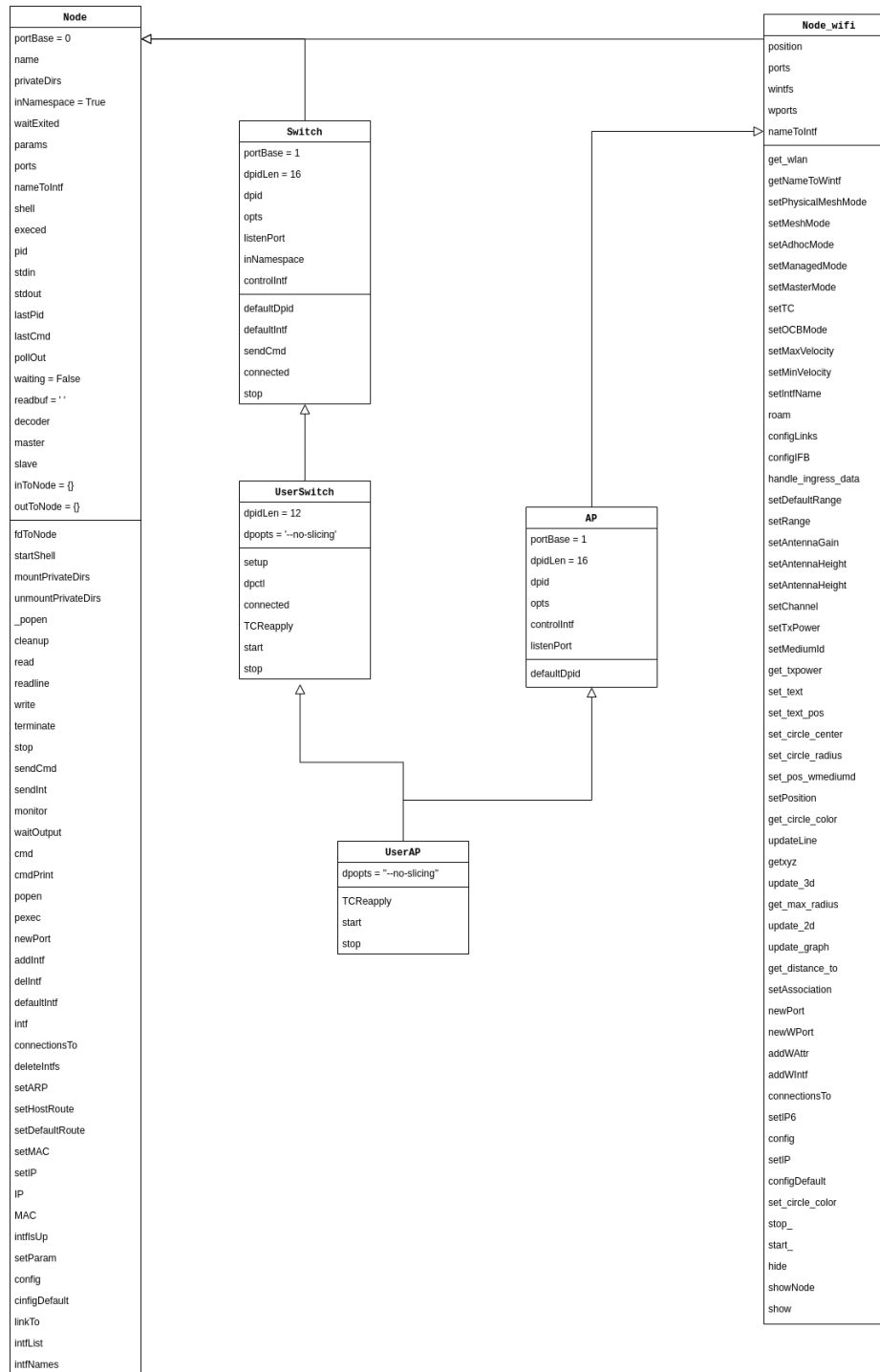


Figura 1.6: Diagrama UML de la clase UserAP

Algún lector podría preguntarse en este punto como va a llevar se 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 de 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*.

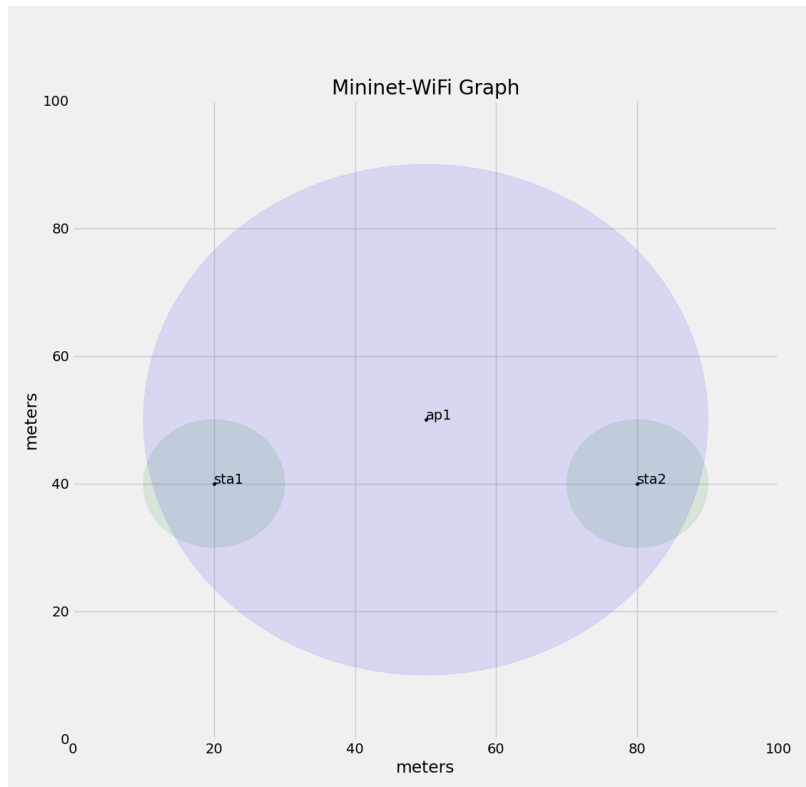


Figura 1.7: Topología básica haciendo uso del UserAP (BOFUSS)

A continuación, en el bloque de código 1.4, 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 que parte.

Código 1.4: 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 haciendole 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 1_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 hacinedo 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 ↵
    ↵ 2001:0:0:0:0:0:0:1/64 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 ↵
    ↵ 2001:0:0:0:0:0:0:2/64 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 ap1-wlan1 (1) to node ap1
93  *** ap1 : ('ip link set', 'ap1-wlan1', 'up')
94  *** ap1 : ('ethtool -K', <WirelessLink ap1-wlan1>, 'gro', 'off')
95  *** ap1 : ('ip link set', 'ap1-wlan1', 'down')
96  *** ap1 : ('ip link set', 'ap1-wlan1', 'address', '00:00:00:00:00:01')
97  *** ap1 : ('ip link set', 'ap1-wlan1', 'up')
98  *** ap1 : ("echo 'interface=ap1-wlan1\ndriver=nl80211\nssid=new-ssid\nwds_sta=1\nhw_mode=g\↵
    ↵ nchannel=1\nctrl_interface=/var/run/hostapd\nctrl_interface_group=0' > mn359884_ap1-↵
    ↵ wlan1.apconf",)
99  > > > > > > *** ap1 : ('hostapd -B mn359884_ap1-wlan1.apconf ',)
100  ap1-wlan1: interface state UNINITIALIZED->ENABLED
101  ap1-wlan1: AP-ENABLED
102  *** ap1 : ('ip link set', 'ap1-wlan1', 'down')
103  *** ap1 : ('ip link set', 'ap1-wlan1', 'address', '00:00:00:00:00:01')
104  *** ap1 : ('ip link set', 'ap1-wlan1', 'up')

```

```

105
106 ### Se configura la potencia y las características intrínsecas de los enlaces
107 _popen ['mnexec', '-da', '359897', 'tc', 'qdisc', 'replace', 'dev', 'ap1-wlan1', 'root', '↵
    ↵ handle', '2:', 'netem', 'rate', '54.0000mbit', 'latency', '1.00ms'] 360049*** ap1 : ('↵
    ↵ tc qdisc add dev ap1-wlan1 parent 2:1 handle 10: pfifo limit 1000',)
108 *** sta1 : ('iw dev', 'sta1-wlan0 set txpower fixed 1400')
109 *** sta2 : ('iw dev', 'sta2-wlan0 set txpower fixed 1400')
110 *** ap1 : ('iw dev', 'ap1-wlan1 set txpower fixed 1400')
111 *** Add links ***
112 added intf sta1-wlan0 (0) to node sta1
113 *** sta1 : ('ip link set', 'sta1-wlan0', 'up')
114 *** sta1 : ('ethtool -K', <WirelessLink sta1-wlan0>, 'gro', 'off')
115 *** executing command: tc qdisc show dev sta1-wlan0
116 *** sta1 : ('tc qdisc show dev sta1-wlan0',)
117 qdisc mq 0: root
118 qdisc fq_codel 0: parent :4 limit 10240p flows 1024 quantum 1514 target 5ms interval 100ms ↵
    ↵ memory_limit 32Mb ecn drop_batch 64
119 qdisc fq_codel 0: parent :3 limit 10240p flows 1024 quantum 1514 target 5ms interval 100ms ↵
    ↵ memory_limit 32Mb ecn drop_batch 64
120 qdisc fq_codel 0: parent :2 limit 10240p flows 1024 quantum 1514 target 5ms interval 100ms ↵
    ↵ memory_limit 32Mb ecn drop_batch 64
121 qdisc fq_codel 0: parent :1 limit 10240p flows 1024 quantum 1514 target 5ms interval 100ms ↵
    ↵ memory_limit 32Mb ecn drop_batch 64
122 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']
123 *** executing command: tc qdisc add dev sta1-wlan0 root handle 5:0 htb default 1
124 *** sta1 : ('tc qdisc add dev sta1-wlan0 root handle 5:0 htb default 1',)
125 *** executing command: tc class add dev sta1-wlan0 parent 5:0 classid 5:1 htb rate 11.000000↵
    ↵ Mbit burst 15k
126 *** sta1 : ('tc class add dev sta1-wlan0 parent 5:0 classid 5:1 htb rate 11.000000Mbit burst ↵
    ↵ 15k',)
127 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']
128 outputs: ['', '']
129 _popen ['mnexec', '-da', '359904', 'iwconfig', 'sta1-wlan0', 'essid', 'new-ssid', 'ap', '↵
    ↵ 00:00:00:00:00:01'] 360059
130 added intf sta2-wlan0 (0) to node sta2
131 *** sta2 : ('ip link set', 'sta2-wlan0', 'up')
132 *** 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 11.000000↵
    ↵ Mbit burst 15k
144 *** sta2 : ('tc class add dev sta2-wlan0 parent 5:0 classid 5:1 htb rate 11.000000Mbit burst ↵
    ↵ 15k',)
145 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']
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 100000000001 --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 1.5, se dejan las líneas extraídas del bloque anterior.

Código 1.5: Puesta en marcha del BOFUSS

```

1  # Binarío 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  # Binarío 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 estudiado en la sección anterior 1.5 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 descriptors de archivos UNIX para intercomunicar `ofdatapath` y el `ofprotocol`.

1.7. 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 1.5). 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 que estrategia vamos a seguir para depurar al switch. Según se puede apreciar en la figura 1.8, 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 1.6, 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.

1.7.1. 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 1.6).

Código 1.6: 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 desarro-

llo de Mininet-WiFi. Al examinar el contenido⁴ 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 1.9, 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.

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.

1.7.2. 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 1.7).

Código 1.7: Script de puesta en marcha del escenario - `launch.sh`

```
1 #!/bin/bash
```

⁴mininet-wifi/blob/master/mn_wifi/clean.py-L77

```

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 ${AP_MAC}
18 ip link set ap1-wlan1 up
19 iw ap1-wlan1 set txpower fixed 100
20 echo -e "interface=ap1-wlan1\ndriver=nl80211\nssid=${AP_SSID}\nwds_sta=1\nhws_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* emuladas, 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`⁵, y a continuación en el bloque de código 1.8 se indican algunos ejemplos de uso de la operativa básica de la herramienta.

Código 1.8: 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 1.10 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 1.11).

1.7.2.1. Troubleshooting

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 1.9: 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 1.10).

⁵https://github.com/patgrosse/mac80211_hwsim_mgmt

Código 1.10: 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.

1.7.3. 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 1.5) en el JSON de depuración. A continuación, en el bloque 1.11, se indica el JSON de configuración para la depuración.

Código 1.11: 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": "${workspaceFolder}/secchan/ofprotocol",
9              "args": [
10                 "unix:/tmp/ap1",
11                 "tcp:localhost:6653",
12                 "--fail=closed",
13                 "--listen=punix:/tmp/ap1.listen"
14             ],
15             "stopAtEntry": false,
16             "cwd": "${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 }
```

```

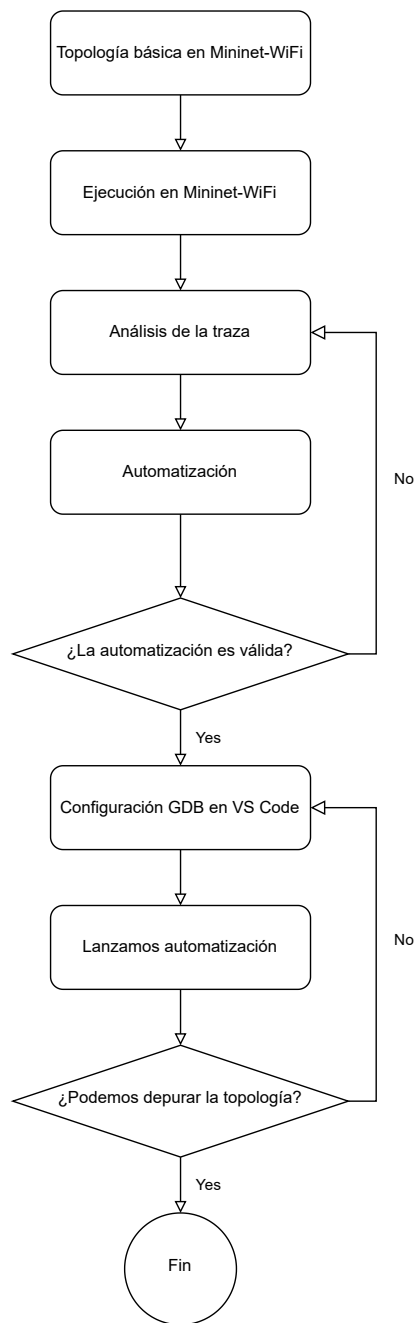
25         }
26     ]
27 },
28 {
29     "name": "(ap1)ofdatapath",
30     "type": "cppdbg",
31     "request": "launch",
32     "program": "${workspaceFolder}/udatapath/ofdatapath",
33     "args": [
34         "-i",
35         "ap1-wlan1",
36         "punix:/tmp/ap1",
37         "-d",
38         "000000000001",
39         "--no-slicing"
40     ],
41     "stopAtEntry": false,
42     "cwd": "${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 ],
55 "compounds": [
56     {
57         "name": "(ap1)ofprotocol/(ap1)ofdatapath",
58         "configurations": [
59             "(ap1)ofprotocol",
60             "(ap1)ofdatapath"
61         ],
62         "preLaunchTask": "${defaultBuildTask}",
63         "stopAll": true
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 adicional⁶⁷.

⁶<https://github.com/microsoft/vscode-cmake-tools/issues/2463>

⁷<https://github.com/microsoft/vscode-cpptools/issues/861>

**Figura 1.8:** Proceso de debug al BOFUSS

```

launch.sh M X
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
bash - inband_research + - 
n0obie@n0obie-Zenbook:~/TFM/src/inband_research$ sudo ./launch.sh
[+] Lanch UserAP config
Created device with ID 2
n0obie@n0obie-Zenbook:~/TFM/src/inband_research$ sudo find /sys/kernel/debug/ieee80211 -name hwsim | cut -d/ -f 6 | sort
intfUserAP
mn25156p00s00
mn25156p01s01
n0obie@n0obie-Zenbook:~/TFM/src/inband_research$

```

Figura 1.9: Listado de interfaces inalámbricas en `/sys/kernel/debug/ieee80211`

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
bash + - 
n0obie@n0obie-Zenbook:~/tfm-thesis$ lsmod | grep -e 'mac80211_hwsim'
mac80211_hwsim      94208      0
mac80211            1323008    2 iwlmvm,mac80211_hwsim
cfg80211            1052672    4 iwlmvm,mac80211_hwsim,iwlwifi,mac80211
n0obie@n0obie-Zenbook:~/tfm-thesis$

```

Figura 1.10: Comprobación de si el módulo `mac80211_hwsim` está cargado

```

n0obie@n0obie-Zenbook:~/TFM/src/inband_research$ iw dev
phy#12
    Interface wlan0
        ifindex 18
        wdev 0xc00000001
        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
n0obie@n0obie-Zenbook:~/TFM/src/inband_research$

```

Figura 1.11: Listado de *phy* inalámbricas usando el comando `iw`

Bibliografía

- [1] 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.
- [2] 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.
- [3] Pablo Collado, David Carrascal, “Mininet Internals,” 2020. [Online]. Available: <https://github.com/GAR-Project/project#mininet-internals>
- [4] 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.
- [5] 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>
- [6] 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.

Lista de Acrónimos y Abreviaturas

6G	the sixth generation of mobile technologies.
BOFUSS	Basic OpenFlow User-space Software Switch.
HLMAC	Hierarchical Local MAC.
IoT	Internet of Things.
LLDP	Link Layer Discovery Protocol.
ONOS	Open Network Operating System.
OvS	Open vSwitch.
RPL	Routing Protocol for Low Power and Lossy Networks.
SDN	Software-Defined Networking.
TC	Traffic control.
TFM	Trabajo Final de Máster.

A. Anexo I - Pliego de condiciones

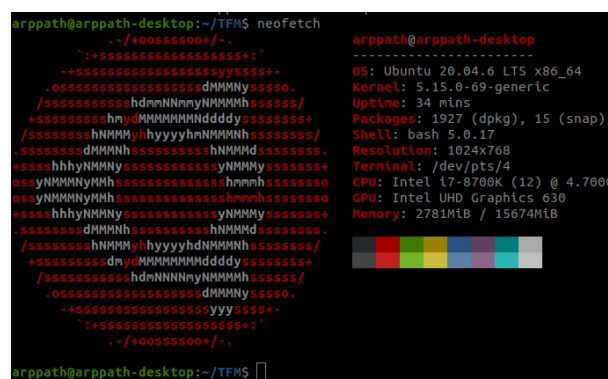
En este anexo se podrán encontrar las condiciones materiales de las distintas máquinas donde se ha llevado a cabo el desarrollo y evaluación del proyecto. De forma adicional, se han indicado las limitaciones *hardware* así como las especificaciones *software* para poder replicar el proyecto de manera 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
  .-/+oosssso+/-.
  :+ssssssssssss+:
  --ssssssssssss--
  .ossssssssssssssdMMMNysssso.
  /sssssssssssssdmmNNnnnyNNNNHssssso/
  +ssssssssshnydNNNNNNNNddddyssssssss+
  /ssssssssshNNNNHssssssssshNNNNHssssssso/
  .ssssssssshMMMNyssssssssshNNNNHssssssso.
  +ssssshhhyNNNNyssssssssshNNNNyssssssso
  ossyNNMMMNyMMHssssssssshmmhssssssso
  ossyNNMMMNyMMHssssssssshmmhssssssso
  +ssssshhhyNNNNyssssssssshNNNNyssssssso.
  .ssssssssshMMMNhssssssssshNNNNHssssssso.
  /ssssssssshNNNNyhyyyhdNNNNHssssssso/
  +ssssssssshdnydNNNNNNNNdddyssssssss+
  /ssssssssshdmmNNNNnyNNNNHssssso/
  .ossssssssssssssdMMMNysssso.
  --ssssssssssss--
  :+ssssssssssss+:
  .-/+oosssso+/-.

arppath@arppath-desktop
-----
OS: Ubuntu 20.04.6 LTS x86_64
Kernel: 5.15.0-69-generic
Uptime: 34 mins
Packages: 1927 (dpkg), 15 (snap)
Shell: bash 5.0.17
Resolution: 1024x768
Terminal: /dev/pts/4
CPU: Intel i7-8700K (12) @ 4.700GHz
GPU: Intel UHD Graphics 630
Memory: 2781MiB / 15674MiB
```

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

```
arppath@arppath-david:~$ neofetch
./+oossssoo+/-./
`:+ssssssssssssssssss+:`
-+ssssssssssssssssssyyssss+-
.ossssssssssssssssssdMMMMNyssssso.
/ssssssssssshdmmNNmmymMMMMhssssss/
+ssssssssshmydMMMMMMNdddyssssssss+
/ssssssssshNMMMyhhyyyyhmNMMNhs/
.ssssssssdMMMMNhssssssssshNMMMdssssssss.
+ssssshhhyNMMNysssssssssssyNMMMyssssss+
ossyNMMMNyMMhssssssssssshmmhssssssso
ossyNMMMNyMMhssssssssssshmmhssssssso
+ssssshhhyNMMNysssssssssssyNMMMyssssss+
.ssssssssdMMMMNhssssssssshNMMMdssssssss.
/ssssssssshNMMMyhhyyyyhdNMMNhs/
+sssssssssdmydMMMMMMNdddyssssssss+
/ssssssssshdMNNNmyNMMMHssssss/
.ossssssssssssssssssdMMMMNyssssso.
-+ssssssssssssssssssyyyssss+-
`:+ssssssssssssssssss+:`
./+oossssoo+/-./

arppath@arppath-david
-----
OS: Ubuntu 22.04.2 LTS x86_64
Host: HP EliteOne 800 G3 23.8-in Non-Touch AiO
Kernel: 5.19.0-38-generic
Uptime: 5 days, 19 hours, 36 mins
Packages: 2354 (dpkg), 17 (snap)
Shell: bash 5.1.16
Resolution: 1920x1080
DE: GNOME 42.5
WM: Mutter
WM Theme: Adwaita
Theme: Yaru-dark [GTK2/3]
Icons: Yaru [GTK2/3]
Terminal: x-terminal-emul
CPU: Intel i5-7500 (4) @ 3.800GHz
GPU: Intel HD Graphics 630
Memory: 7616MiB / 31984MiB
```

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

```
n0obie@n0obie-Zenbook:~$ neofetch
      .-/+00SSSS00+/- .
    `:+SSSSSSSSSSSSSSSS+:`
  -+SSSSSSSSSSSSSSSSyySSSS+-
 .oSSSSSSSSSSSSSSSSdMMMMySSSSo.
 /SSSSSSSSSSShdmmNNmmyNMMMMhSSSSS/
 +SSSSSSSSShmyeMMMMMMMNdddySSSSSSS+
 /SSSSSSSShNMMMyhhyyyhmNMMMNhSSSSSSS/
 .SSSSSSSSdMMMNhSSSSSSSSShNMMMdSSSSSSS.
 +SSShhhyNMMMySSSSSSSSSSSyNMMMySSSSSSS+
 ossyNMMMNyMMhSSSSSSSSSSShmmhSSSSSSSo
 ossyNMMMNyMMhSSSSSSSSSSShmmhSSSSSSSo
 +SSShhhyNMMMySSSSSSSSSSSyNMMMySSSSSSS+
 .SSSSSSSSdMMMNhSSSSSSSSShNMMMdSSSSSSS.
 /SSSSSSShNMMMyhhyyyhdNMMMNhSSSSSSS/
 +SSSSSSSSdmydMMMMMMMNdddySSSSSSS+
 /SSSSSSSSShdmmNNmmyNMMMMhSSSSSSS/
 .oSSSSSSSSSSSSSSSSdMMMMySSSSo.
 -+SSSSSSSSSSSSSSSSyySSSS+-
 `:+SSSSSSSSSSSSSSSS+:`
      .-/+00SSSS00+/- .

n0obie@n0obie-Zenbook
-----
OS: Ubuntu 22.04.2 LTS x86_64
Host: Zenbook UX3402ZA_UX3402ZA 1.0
Kernel: 5.19.0-35-generic
Uptime: 3 days, 13 hours, 37 mins
Packages: 2118 (dpkg), 12 (snap)
Shell: bash 5.1.16
Resolution: 2880x1800
DE: GNOME 42.5
WM: Mutter
WM Theme: Adwaita
Theme: Yaru-dark [GTK2/3]
Icons: Yaru [GTK2/3]
Terminal: gnome-terminal
CPU: 12th Gen Intel i7-1260P (16) @ 4.700GHz
GPU: Intel Alder Lake-P
Memory: 3257MiB / 15624MiB
```

Figura A.3: Especificaciones de la máquina C

Universidad de Alcalá
Escuela Politécnica Superior



ESCUELA POLITECNICA
SUPERIOR



Universidad
de Alcalá