Universidad Zaragoza
1542

Trabajo Fin de Grado

Implementación de un sistema para la monitorización y administración de comunicaciones IP en entornos Wi-Fi

Implementation of a system for monitoring and management of IP communications in Wi-Fi environments.

Autor/es

Zineb Helali Amoura

Director/es

Julián Fernández Navajas

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2025

Resumen

El análisis del uso de aplicaciones dentro de redes inalámbricas plantea múltiples desafíos, especialmente cuando se desea detectar patrones de comportamiento en tiempo real sin acceso al contenido de los datos. Este trabajo propone un sistema capaz de identificar eventos relevantes relacionados con el uso de aplicaciones como Spotify, TikTok o WhatsApp, con el objetivo de evaluar el grado de uso y aplicar medidas si se detecta un consumo excesivo.

Para ello, se ha desarrollado un mecanismo de observación que analiza las conexiones establecidas por los dispositivos conectados al punto de acceso. A través del estudio de patrones característicos, como la duración de las conexiones, la recurrencia de accesos o el tipo de contenido cargado, el sistema infiere el tipo de actividad que está teniendo lugar; desde una simple apertura de aplicación hasta la reproducción continuada de contenido multimedia.

Uno de los aspectos más complejos ha sido adaptar el sistema a las particularidades de cada aplicación, ya que su comportamiento en red varía significativamente. Además, se ha implementado una lógica de actuación que permite aplicar medidas de control en función del uso detectado, como la limitación temporal del tráfico para prevenir un uso abusivo.

Los resultados muestran que el sistema es capaz de detectar de forma precisa accesos, reproducciones y envío de mensajes, diferenciando entre actividades ocasionales y usos intensivos. Este trabajo demuestra la viabilidad de implementar un sistema inteligente de monitorización y gestión del tráfico orientado a mejorar el uso responsable de las redes Wi-Fi en entornos locales.

Índice general

1 Introducción 3

1.1 Contexto y motivación 3

1.2 Objetivos y alcance del proyecto 5

1.3 Metodología y herramientas 5

1.4 Organización de la memoria del Trabajo de Fin de Grado 7

2 Fundamentos previos 8

2.1 Flujos IP 8

2.2 Redes virtuales 9

2.3 Monitorización de tráfico en red 10

2.4 Filtrado de paquetes y control del tráfico 10

2.4.1 Conceptos generales 11

2.4.2 Mecanismos de filtrado a nivel de red 11

2.4.3 Mecanismos de filtrado a nivel de aplicación 11

2.4.4 Open vSwitch como herramienta de filtrado 11

2.5 Broker de mensajes y transmisión de eventos 12

3 Diseño del escenario y arquitectura del sistema 13

3.1 Esquema general del sistema 13

3.2 Topología de red y configuración del entorno 16

3.2.1 Configuración de red y servicios de conectividad 17

4 Desarrollo del sistema de monitorización 22

4.1 Observación inicial del tráfico con Wireshark 22

4.2 Desarrollo de los scripts de Zeek para la detección de eventos 26

4.3 Procesamiento de eventos y medidas de actuación 31

5 Experimentos y resultados **34**
5.1 Entorno y metodología de pruebas 34
5.2 Resultados obtenidos 35
5.2.1 Pruebas de validación y posibles escenarios en producción . . . 37

6 Conclusiones y trabajo futuro **39**
6.1 Conclusiones 39
6.2 Trabajo futuro 40

Bibliography **42**

Créditos y licencias de recursos gráficos **43**

A Configuración del punto de acceso **44**

B Fragmento de configuración de Open vSwitch **45**

C Configuración del servidor DHCP **46**

D Configuración de la red **47**

Índice de figuras

1.1 Alternativas para acceder al tráfico generado por los dispositivos móviles 4
1.2 Diagrama de Gantt que muestra el desarrollo del proyecto. 6

2.1 Representación del concepto de flujo IP a partir del 5-tuple. [4] 9

3.1 Esquema de la arquitectura desplegada en VirtualBox 15
3.2 Esquema de la arquitectura desplegada en Raspberry Pi 16
3.3 Adaptador de red configurado en modo NAT en VirtualBox 19
3.4 Puente virtual y replicación de tráfico en la Raspberry Pi 21
3.5 Puente virtual y replicación de tráfico en la máquina virtual 21

4.1 Reproducción de canciones en Spotify 23
4.2 Captura de tráfico generado al visualizar vídeos en TikTok 24

5.1 Flujo completo del sistema ante una reproducción en Spotify 35
5.2 Captura en Wireshark de un flujo TLS. 36
5.3 Evento correspondiente detectado por Zeek. 36
5.4 Eventos recibidos por el módulo Python desde Zeek 37
5.5 Ejemplo de corte de tráfico tras detección de reproducción 37

Índice de cuadros

1.1 Horas dedicadas a cada tarea del proyecto. 6

Lista de Acrónimos

AMQP	Advanced Message Queuing Protocol
AP	Access Point
API	Application Programming Interface
ARP	Address Resolution Protocol
CDN	Content Delivery Network
CeNIT	Communication Networks and Information Technologies
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
GNU	GNU's Not Unix
HTTP	HyperText Transfer Protocol
I3A	Instituto de Investigación en Ingeniería de Aragón
IP	Internet Protocol
JSON	JavaScript Object Notation
LAN	Local Area Network
NAT	Network Address Translation
OSI	Open Systems Interconnection
OVS	Open vSwitch
PSK	Pre-Shared Key
QoE	Quality of Experience
QoS	Quality of Service
QUIC	Quick UDP Internet Connections
SDN	Software Defined Networking
SIP	Session Initiation Protocol
SNI	Server Name Indication
SSID	Service Set Identifier

SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
USB	Universal Serial Bus
VM	Virtual Machine
VoIP	Voice over IP
Wi-Fi	Wireless Fidelity
WPA2	Wi-Fi Protected Access 2
XMPP	Extensible Messaging and Presence Protocol
XML	eXtensible Markup Language

Capítulo 1

Introducción

1.1. Contexto y motivación

En la actualidad, el uso de dispositivos conectados a Internet (teléfonos móviles, tabletas y ordenadores) se ha extendido de forma masiva y desde edades muy tempranas. Diversos estudios han alertado sobre los efectos que un uso prolongado y no supervisado de las pantallas puede tener sobre el desarrollo neurológico, el comportamiento social y el bienestar general, especialmente en población infantil y adolescente [1]. Este fenómeno ha dado lugar a una creciente preocupación por parte de familias, educadores e instituciones sobre cómo se utiliza la conectividad, qué tipo de contenido se consume y cuánto tiempo se permanece en línea.

En este contexto, resulta pertinente disponer de herramientas capaces de observar y analizar en tiempo real el comportamiento de los flujos de tráfico IP en redes Wi-Fi de uso común. No se trata únicamente de detectar amenazas de seguridad o ataques maliciosos, sino también de identificar patrones de uso que puedan derivar en hábitos perjudiciales o poco saludables para los usuarios. Estas situaciones, sin ser necesariamente maliciosas, pueden comprometer el equilibrio digital necesario para un desarrollo saludable de la vida personal, académica y social.

Aunque lo ideal sería analizar directamente el tráfico generado por cada dispositivo, las restricciones impuestas por los sistemas operativos móviles actuales, especialmente en Android e iOS, hacen que esto no sea posible sin acceso privilegiado. Además, el rooteo de dispositivos es cada vez menos viable por motivos técnicos y de seguridad[2]. Es importante señalar que los dispositivos móviles pueden conectarse tanto a través de redes móviles como mediante Wi-Fi; sin embargo, este trabajo se centra exclusivamente en el análisis del tráfico Wi-Fi, ya que es el único que puede observarse desde la red local sin intervención directa sobre los terminales.

Por ello, este trabajo adopta un enfoque basado en la observación pasiva del tráfico desde la propia red Wi-Fi, lo que permite inferir patrones de uso sin necesidad de modificar ni acceder a los dispositivos móviles. Esta solución es reproducible y puede desplegarse en plataformas de bajo coste, como una Raspberry Pi, facilitando así su aplicación en entornos reales.

Para ello, se han valorado distintas formas de capturar el tráfico generado por los dispositivos móviles. La Figura 1.1 muestra dos opciones: conectar un punto de acceso a la red local o crear un punto de acceso propio desde el equipo mediante un

adaptador USB. Esta última opción, más sencilla de implementar y sin necesidad de modificar la red existente, es la elegida en este trabajo.

Cabe destacar que también existen soluciones alternativas, como el uso de proxies basados en ARP o DHCP para interceptar tráfico sin necesidad de modificar la infraestructura. Sin embargo, estas técnicas pueden verse limitadas o bloqueadas por mecanismos de defensa cada vez más presentes en los dispositivos modernos, como la detección de suplantación o el aislamiento de redes sospechosas, lo que afecta negativamente a su fiabilidad.

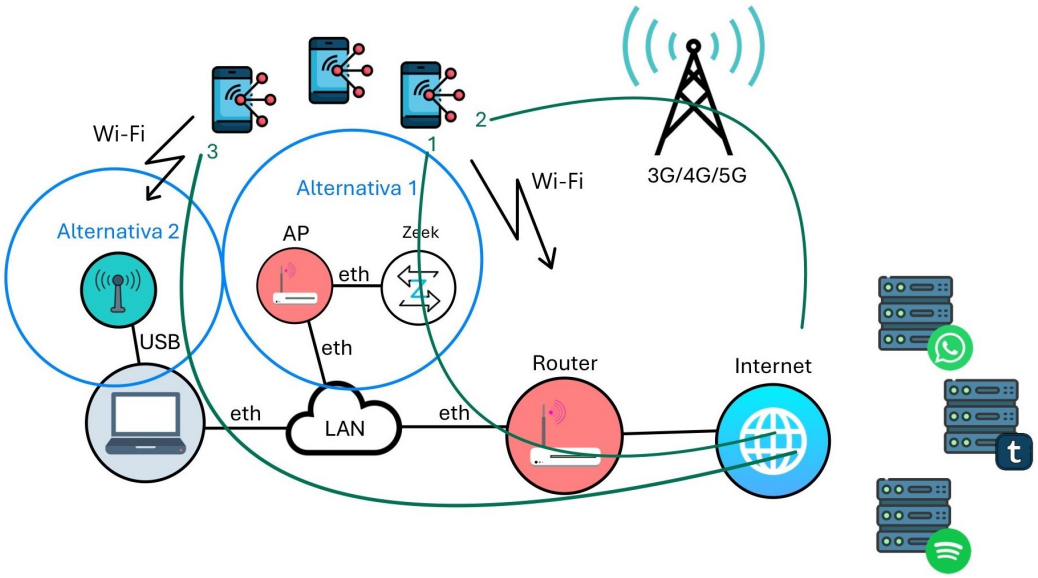

Figura 1.1: Alternativas para acceder al tráfico generado por los dispositivos móviles

Este enfoque define la base técnica sobre la que se construye el presente Trabajo de Fin de Grado, que se enmarca en el grupo de investigación CeNIT (*Communication Networks and Information Technologies*), reconocido como grupo de referencia por el Gobierno de Aragón y adscrito al Instituto de Investigación en Ingeniería de Aragón (I3A) de la Universidad de Zaragoza. CeNIT centra su actividad en el ámbito de las redes de comunicaciones y tecnologías para la sociedad de la información, con líneas de investigación que abarcan desde los sistemas avanzados de comunicaciones móviles y redes 4G/5G hasta la ciberseguridad y la calidad de servicio y experiencia (QoS/QoE) [3].

Este trabajo se sitúa en la intersección de esas líneas, proponiendo un sistema de monitorización y administración del tráfico IP en accesos Wi-Fi basado en un enfoque orientado a eventos. La motivación principal reside en proporcionar una solución técnica capaz de detectar, en tiempo real, flujos de datos que, por su duración, destino o frecuencia puedan indicar un uso abusivo de la red. Esta aproximación permitirá, en el futuro, desarrollar políticas adaptativas o mecanismos de intervención orientados a fomentar un uso más responsable y equilibrado de Internet en entornos educativos o familiares.

1.2. Objetivos y alcance del proyecto

El objetivo principal de este proyecto es desarrollar un sistema capaz de monitorizar y gestionar el tráfico IP en redes Wi-Fi, con el fin de detectar patrones de uso intensivo que puedan indicar un consumo poco equilibrado de los recursos de red. El sistema se basa en el análisis de eventos en tiempo real y está orientado a su aplicación en entornos domésticos o educativos. Durante la fase de desarrollo, el sistema se ejecutará inicialmente en un entorno virtualizado; es decir, sobre una máquina virtual alojada en un equipo de desarrollo, y posteriormente se adaptará a una plataforma física de bajo coste para facilitar su uso en escenarios reales. Este despliegue no debe confundirse con la topología de red virtual definida en el sistema, la cual se diseña específicamente para permitir la interceptación y análisis del tráfico generado por los dispositivos conectados.

A partir de este planteamiento general, se establecen los siguientes objetivos específicos:

1. Diseñar una topología de red virtual que permita interceptar y redirigir el tráfico generado por los dispositivos conectados.
2. Capturar y examinar el tráfico para identificar características o comportamientos asociados a un uso intensivo de la red.
3. Desarrollar reglas que permitan detectar, en tiempo real, eventos de interés a partir del tráfico observado.
4. Establecer un canal de comunicación que permita enviar los eventos detectados a un sistema externo para su procesamiento.
5. Implementar un módulo de respuesta que analice los eventos recibidos y permita tomar decisiones o aplicar medidas según el tipo de actividad detectada.
6. Instalar el sistema completo en un dispositivo físico portátil que pueda ser utilizado en contextos reales.

1.3. Metodología y herramientas

A lo largo del proyecto se han utilizado diversas herramientas y tecnologías, cada una con un propósito específico en el flujo de trabajo. A continuación se describen las más relevantes:

- **Bash:** Utilizado para la automatización de tareas en la configuración de red y despliegue de servicios.
- **LaTeX:** Utilizado para la redacción de la presente memoria.
- **Open vSwitch (OVS):** *Switch* virtual para la gestión flexible del tráfico en la red del punto de acceso Wi-Fi.
- **Python 3.10:** Lenguaje principal para el desarrollo del módulo de procesamiento automático que recibe eventos desde *Zeek* a través de *RabbitMQ*.

- **RabbitMQ:** Sistema de mensajería (*message broker*) utilizado para la comunicación eficiente entre *Zeek* y los scripts en *Python*.
- **Wireshark:** Herramienta de captura y análisis de tráfico en red empleada para la observación directa y la identificación de patrones antes de la automatización.
- **Zeek:** Monitor de seguridad en red basado en eventos, utilizado para el análisis y extracción de patrones relevantes en el tráfico IP.

Con el fin de representar de forma clara la evolución temporal del proyecto y la distribución del esfuerzo a lo largo del tiempo, se ha elaborado un diagrama de Gantt. En la Figura 1.2 se muestra la secuencia de tareas principales desde febrero hasta junio, incluyendo las fases de análisis, desarrollo, pruebas y redacción. Cada barra representa una actividad clave del proyecto, indicando su duración y posibles solapamientos. Además, en la Tabla 1.1 se recoge el número estimado de horas dedicadas a cada tarea.

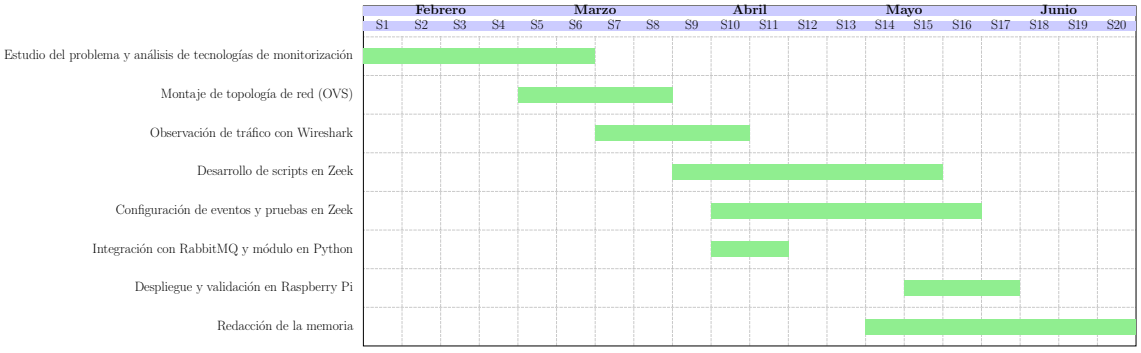

Figura 1.2: Diagrama de Gantt que muestra el desarrollo del proyecto.

Tareas	**Tiempo (en horas)**
Estudio del problema y tecnologías (Zeek, OVS...)	30
Montaje de topología de red	50
Observación del tráfico con Wireshark	25
Desarrollo de scripts personalizados en Zeek	90
Integración de Zeek con RabbitMQ y módulo Python	15
Experimentos	40
Despliegue y validación en Raspberry Pi	10
Reuniones	20
Redacción de la memoria	50
Total	**330**

Cuadro 1.1: Horas dedicadas a cada tarea del proyecto.

1.4. Organización de la memoria del Trabajo de Fin de Grado

Tras la exposición de la introducción, la motivación y los objetivos del proyecto, este documento se estructura en distintos capítulos que abordan de forma progresiva y detallada el sistema desarrollado y sus fundamentos técnicos.

En el Capítulo 2, se presentan los fundamentos previos necesarios para comprender el funcionamiento del sistema propuesto. Se abordan conceptos clave como los flujos IP, la virtualización de red con *Open vSwitch*, la monitorización de tráfico mediante herramientas como *Wireshark* y *Zeek*, los mecanismos de filtrado de paquetes y el uso de *RabbitMQ* como broker de mensajes. Estos fundamentos ofrecen el contexto técnico imprescindible para entender las decisiones tomadas durante el desarrollo.

El Capítulo 3 describe el diseño del escenario de trabajo y la arquitectura general del sistema. Se detalla la topología de red configurada con *Open vSwitch* y la forma en la que se integran las distintas herramientas empleadas para lograr la monitorización y el control de tráfico en un entorno virtualizado.

A continuación, el Capítulo 4 desarrolla en detalle el proceso práctico de construcción del sistema. Se explican las fases de observación inicial del tráfico con *Wireshark*, la implementación de scripts personalizados en *Zeek* para la detección de eventos relevantes, y la integración con un módulo de procesamiento en *Python* que actúa en tiempo real mediante las capacidades de control de tráfico de *Open vSwitch*.

El Capítulo 5 describe los experimentos realizados para validar la funcionalidad del sistema. Se detalla cómo se ha verificado que el sistema monitoriza correctamente las conexiones, detecta los patrones de tráfico y actúa para cortar o restablecer la conectividad según las necesidades observadas.

Finalmente, en el Capítulo 6 se recogen las conclusiones derivadas de este proyecto, así como posibles líneas de trabajo futuro orientadas a ampliar las capacidades del sistema, mejorar su precisión o adaptarlo a nuevos escenarios de uso.

Capítulo 2

Fundamentos previos

Este capítulo presenta los conocimientos técnicos necesarios para entender el funcionamiento del sistema desarrollado. Se abordan conceptos específicos que son clave para su implementación, como la estructura de los flujos IP, el uso de *switch* virtual para el reencaminamiento y la captura de tráfico, el modelo de eventos, el filtrado de paquetes, la transmisión de datos mediante *broker* de mensajes, o la configuración dinámica del sistema en distintos escenarios.

2.1. Flujos IP

En el análisis de tráfico de red, un flujo IP se define como un conjunto de paquetes que comparten las mismas características a nivel de red y transporte. Específicamente, se identifica mediante la combinación de cinco campos clave, conocida como *5-tupla*: dirección IP de origen, dirección IP de destino, puerto de origen, puerto de destino y protocolo de transporte (TCP o UDP). Estos cinco elementos permiten agrupar los paquetes que forman parte de una misma comunicación o sesión, como si fueran una única unidad lógica.

La agrupación de paquetes en flujos IP facilita tareas como la monitorización, el filtrado, la inspección de uso y la detección de patrones de comportamiento. Por ejemplo, un flujo puede corresponder a la reproducción de un vídeo en línea, una llamada por VoIP o la descarga de un archivo. Analizar estos flujos, en lugar de cada paquete por separado, permite obtener métricas más relevantes y significativas para evaluar la calidad y la naturaleza de la comunicación.

En el presente proyecto, los flujos IP constituyen la unidad básica de análisis para detectar patrones de uso que puedan indicar consumos excesivos o prolongados de Internet. A partir de cada flujo, el sistema extrae parámetros como la duración, el volumen de datos transferidos y el número de paquetes, con el objetivo de identificar situaciones que, aunque no necesariamente maliciosas, pueden reflejar comportamientos problemáticos o un uso desequilibrado de la red.

Aunque en este trabajo se asume que cada flujo IP representa una única comunicación lógica, es importante señalar que, en ciertos escenarios —como en protocolos que permiten multiplexación o reutilización de conexiones, como HTTP/2—, un mismo flujo puede contener múltiples intercambios independientes. Esta limitación no se aborda en esta implementación, pero se considera una línea de trabajo futura para lograr un análisis más preciso del comportamiento de red.

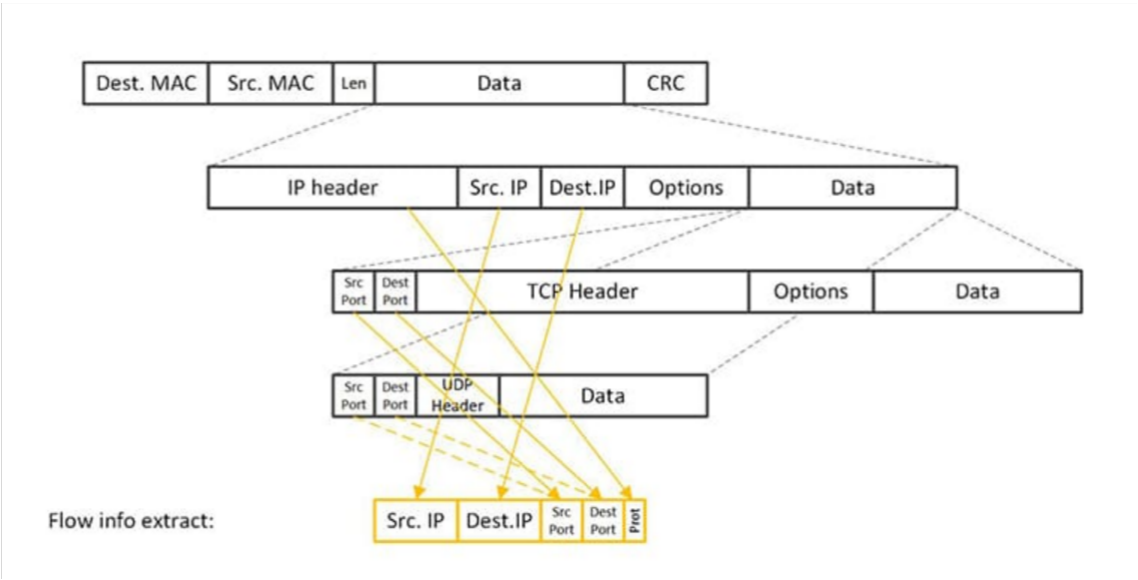

Figura 2.1: Representación del concepto de flujo IP a partir del 5-tuple. [4]

2.2. Redes virtuales

La virtualización de red es una tecnología que permite abstraer los recursos físicos de la red y crear entornos virtuales que funcionan como redes reales, pero con mayor flexibilidad y escalabilidad [5]. Esta abstracción facilita la gestión y configuración de redes, sin requerir modificaciones en la infraestructura física.

Un componente clave en este ámbito es *Open vSwitch (OVS)*, un conmutador virtual de código abierto que actúa como *switch* de red en entornos virtualizados [6]. OVS permite implementar políticas avanzadas de segmentación y gestión del tráfico, facilitando la creación de redes virtuales y su integración con tecnologías de virtualización como máquinas virtuales o contenedores. Sus principales características incluyen:

- Flexibilidad y adaptación: permite la creación y modificación dinámica de topologías de red virtual, adaptándose a distintos escenarios y necesidades.
- Compatibilidad con herramientas de análisis: posibilita la captura y redirección del tráfico de red a herramientas especializadas como *Wireshark* o *Zeek*, esenciales en la fase de monitorización [7, 8].
- Integración con sistemas de orquestación: OVS se integra fácilmente con plataformas de gestión y orquestación de servicios, como *OpenStack* o Kubernetes, facilitando la administración de redes virtuales [9].

En el marco de este trabajo, OVS desempeña un papel fundamental, ya que permite capturar y redirigir el tráfico de la red Wi-Fi del entorno monitorizado para su posterior análisis y filtrado. Este enfoque contrasta con soluciones tradicionales como iptables, que aunque también permiten el filtrado de paquetes, carecen de la granularidad y flexibilidad necesarias para entornos virtualizados o con múltiples interfaces virtuales [9]. Mientras que iptables opera en el nivel del núcleo del sistema operativo, OVS actúa en la capa de conmutación, lo que permite un control más detallado y adaptado a las necesidades actuales de redes definidas por software y virtualización de servicios [10].

2.3. Monitorización de tráfico en red

La monitorización del tráfico en red es un proceso esencial para comprender cómo los dispositivos y aplicaciones interactúan en un entorno de red. Consiste en capturar, almacenar y examinar los paquetes de datos que circulan, con el objetivo de identificar patrones de uso, detectar posibles fallos o anomalías y evaluar el rendimiento general de la red.

Desde un punto de vista técnico, esta tarea implica observar y analizar los paquetes que forman las comunicaciones activas en la red. Se tienen en cuenta aspectos como las direcciones IP, los puertos implicados, el protocolo de transporte utilizado y el momento en el que se producen los intercambios de datos. De esta forma, se puede caracterizar cada comunicación y obtener una visión más completa del comportamiento de la red.

La monitorización puede realizarse con dos enfoques principales:

- Análisis detallado de paquetes: Consiste en examinar cada paquete individualmente, obteniendo información de las cabeceras de todas las capas implicadas (enlace, red, transporte y aplicación) y del contenido transmitido. Este tipo de análisis permite identificar de forma precisa el tráfico de red y resulta especialmente útil para depurar problemas concretos o entender el funcionamiento de protocolos. Herramientas como *Wireshark* permiten realizar esta tarea, ofreciendo una visión exhaustiva y pormenorizada de cada transmisión.

- Análisis basado en eventos o flujos: Este enfoque agrupa los paquetes en conexiones lógicas y genera eventos de alto nivel que resumen la actividad observada. Se centra en registrar acciones relevantes, como el inicio y fin de las conexiones, o la aparición de ciertos patrones de tráfico. Herramientas como *Zeek* utilizan este enfoque para estructurar y procesar la información de forma automatizada, reduciendo la necesidad de intervención manual y permitiendo la detección de patrones de comportamiento en redes de gran volumen.

En este trabajo, se han utilizado ambas herramientas para cubrir las necesidades de análisis: *Wireshark* se ha empleado para capturar y validar el tráfico en bruto a nivel de red, aportando información precisa sobre las cabeceras y datos transmitidos [11, 12]. Por su parte, *Zeek* ha permitido estructurar y procesar el tráfico de forma continua, generando eventos que reflejan las conexiones y comportamientos observados [8].

2.4. Filtrado de paquetes y control del tráfico

El filtrado de paquetes y el control del tráfico son componentes esenciales en la gestión de redes, especialmente en entornos donde es necesario garantizar la seguridad y el rendimiento de las comunicaciones. Estas técnicas permiten decidir qué paquetes pueden circular y cómo se deben gestionar, basándose en diferentes criterios.

2.4.1. Conceptos generales

El filtrado de paquetes consiste en examinar cada paquete de red para determinar si debe ser aceptado, rechazado o modificado, en función de reglas predefinidas. Estas reglas pueden basarse en direcciones IP, puertos, protocolos, estados de conexión u otros atributos del tráfico.

El control del tráfico, por su parte, amplía el filtrado con acciones adicionales, como la priorización o limitación del ancho de banda, lo que permite gestionar el rendimiento de la red y garantizar la calidad de servicio (QoS).

2.4.2. Mecanismos de filtrado a nivel de red

A nivel de red y transporte, el filtrado de paquetes suele apoyarse en herramientas como *iptables*, que operan en el núcleo del sistema operativo. Estas herramientas permiten definir reglas que controlan el tráfico entrante y saliente, basándose en la información contenida en las cabeceras de los paquetes (por ejemplo, dirección IP de origen, dirección IP de destino, puerto de origen y puerto de destino).

Este tipo de filtrado es eficiente para tareas de seguridad básica y control de acceso, pero no tiene visibilidad sobre los datos de las aplicaciones.

2.4.3. Mecanismos de filtrado a nivel de aplicación

En capas más altas del modelo OSI, se pueden aplicar técnicas de filtrado a nivel de aplicación. Estas soluciones permiten analizar el contenido de los paquetes para tomar decisiones más avanzadas, como detectar patrones específicos en protocolos como HTTP, DNS o SIP. Ejemplos de herramientas en este ámbito son los proxies de aplicación y los *firewall* de capa 7.

2.4.4. Open vSwitch como herramienta de filtrado

En este proyecto se ha utilizado *Open vSwitch (OVS)* como herramienta principal para el control del tráfico y la monitorización. OVS está diseñado para operar principalmente en las capas de conmutación (niveles 2 y 3 del modelo OSI), permitiendo aplicar reglas de filtrado más flexibles y adaptadas a entornos virtualizados. No obstante, su arquitectura también permite actuar a nivel de capa 4 (transporte), mediante el filtrado por puertos TCP/UDP, e incluso puede interactuar con capas superiores cuando se utiliza en combinación con controladores SDN compatibles con *OpenFlow*.

A diferencia de herramientas como *iptables*, que operan en el núcleo del sistema operativo y permiten establecer reglas de filtrado a nivel básico, *Open vSwitch* ofrece un mayor grado de flexibilidad y control directo sobre el tráfico gracias a su funcionamiento en la capa de conmutación (niveles 2 y 3 del modelo OSI). En este proyecto, OVS se emplea para aplicar políticas de control de tráfico de manera precisa y dinámica, actuando sobre flujos específicos de la red inalámbrica monitorizada.

Este enfoque resulta útil en un entorno donde se desea mantener la visibilidad total de las conexiones y, al mismo tiempo, disponer de la capacidad de actuar en tiempo real para cortar o modificar el tráfico de ciertos servicios detectados como

problemáticos. OVS permite definir reglas de flujo (*flow rules*) que especifican exactamente qué paquetes deben ser bloqueados, redirigidos o reenviados en función de criterios detallados, superando las limitaciones de granularidad que presentan soluciones como *iptables* en escenarios con múltiples interfaces virtualizadas o configuraciones de red más complejas [13].

2.5. Broker de mensajes y transmisión de eventos

Un *broker* de mensajes es un componente en sistemas distribuidos que facilita la comunicación asíncrona entre diferentes módulos o aplicaciones. Su función principal consiste en recibir, almacenar temporalmente y reenviar mensajes o eventos de forma eficiente y confiable, desacoplando a los productores (publicadores) de los consumidores (suscriptores). Esto permite que cada componente funcione de manera independiente, favoreciendo la escalabilidad en arquitecturas.

RabbitMQ es uno de los *broker* más empleados en la actualidad. Basado en el protocolo AMQP (*Advanced Message Queuing Protocol*), ofrece múltiples ventajas:

- Enrutamiento flexible: RabbitMQ utiliza intercambios (*exchanges*) para dirigir los mensajes a las colas apropiadas según reglas de enlace (*bindings*), permitiendo patrones como *fanout*, *direct*, *topic* y *headers* [14].
- Persistencia y fiabilidad en la entrega de eventos, asegurando que los mensajes no se pierdan incluso en caso de fallos temporales.
- Compatibilidad con diversos lenguajes de programación y entornos.
- Herramientas de administración y monitorización que simplifican la gestión y la supervisión de las comunicaciones.

En el presente proyecto, se emplea un *broker* de mensajes como mecanismo de comunicación entre el módulo de monitorización de tráfico y el sistema encargado de aplicar las acciones de control, permitiendo así una arquitectura desacoplada y extensible.

Capítulo 3

Diseño del escenario y arquitectura del sistema

3.1. Esquema general del sistema

Antes de entrar en detalle sobre la configuración técnica y los componentes implicados, se presenta una visión esquemática de la arquitectura desplegada. Con el objetivo de ilustrar de forma clara el comportamiento global del sistema, se incluye un pseudocódigo que recoge, de manera simplificada, las distintas etapas que componen la arquitectura propuesta. El sistema fue desarrollado inicialmente en un entorno virtual, ejecutado mediante VirtualBox sobre un equipo de desarrollo, y posteriormente trasladado a un entorno físico, utilizando una Raspberry Pi como nodo independiente.

Algorithm 1: Funcionamiento general del sistema de detección

```
Iniciar punto de acceso Wi-Fi con hostapd;
Iniciar servidor DHCP con isc-dhcp-server;
Crear puente virtual con Open vSwitch;
Duplicar el tráfico hacia interfaz interna gateway1;
Iniciar Zeek para escuchar tráfico en gateway1;
while exista tráfico de red do
    if la conexión coincide con patrones de Spotify, TikTok o WhatsApp
     then
        Clasificar el evento;
        Escribir el evento en log local;
        Enviar el evento a RabbitMQ;
Iniciar módulo en Python que escucha eventos de RabbitMQ;
while llegue un nuevo evento do
    if el evento indica uso intensivo then
        Esperar unos segundos;
        Bloquear el tráfico de esa IP con iptables;
        Esperar tiempo de penalización;
        Eliminar la regla de bloqueo;
```

Ambos entornos comparten una estructura lógica común, diseñada para situar el sistema en una posición estratégica entre los dispositivos móviles que se conectan a una red Wi-Fi gestionada por el propio sistema y su salida a Internet. Esta similitud arquitectónica permite que la solución desarrollada sea portable y reproducible, asegurando que lo validado en el entorno virtual funcione de forma equivalente en el entorno físico. La posición intermedia del sistema permite capturar el tráfico generado de forma pasiva, sin interferir en la experiencia del usuario ni requerir cambios en las aplicaciones utilizadas por los dispositivos conectados.

La lógica de funcionamiento del sistema se basa en tres bloques principales: la provisión de conectividad inalámbrica mediante un punto de acceso Wi-Fi controlado por software; la monitorización del tráfico a través de herramientas de análisis pasivo; y la capacidad de actuar en tiempo real sobre el flujo de red en función de los eventos detectados. Para ello, se ha diseñado una arquitectura modular en la que cada uno de estos bloques opera de forma independiente, pero conectada mediante una infraestructura de red virtual gestionada por software.

El entorno virtual ha sido clave durante las fases de desarrollo, prueba y ajuste del sistema, permitiendo reproducir escenarios reales en un entorno controlado. Por su parte, el entorno físico ha demostrado la viabilidad del despliegue en condiciones reales, utilizando hardware de bajo coste como una Raspberry Pi y un adaptador Wi-Fi USB externo.

En las figuras siguientes se muestran los esquemas correspondientes a ambos entornos. Aunque las implementaciones difieren en aspectos como el sistema anfitrión o la gestión del acceso a Internet, el flujo lógico de red y la ubicación del sistema en la arquitectura se mantienen inalterados. Esta consistencia ha permitido validar que el sistema es portable y adaptable, sin requerir modificaciones profundas al cambiar el entorno de ejecución.

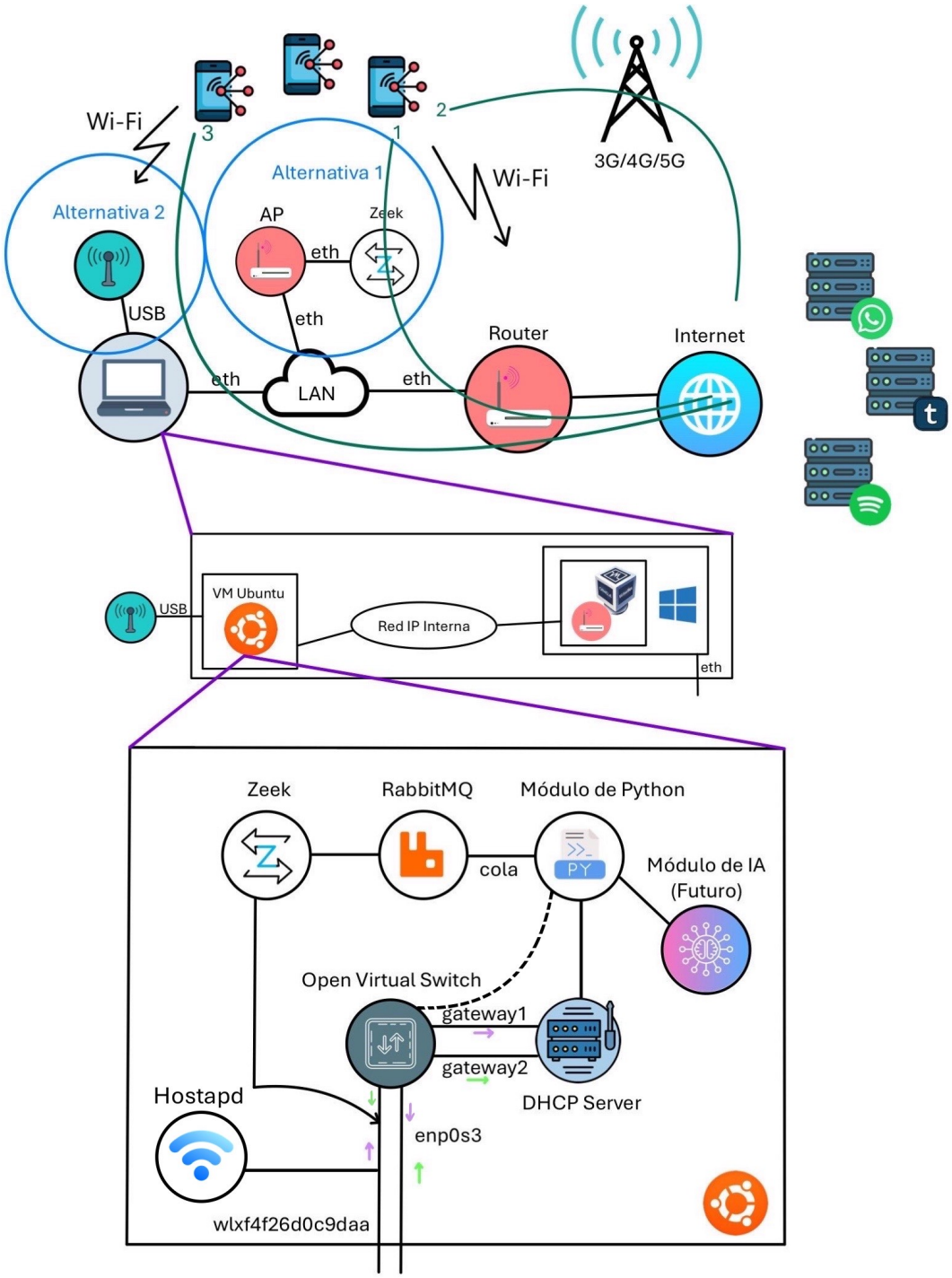

Figura 3.1: Esquema de la arquitectura desplegada en VirtualBox

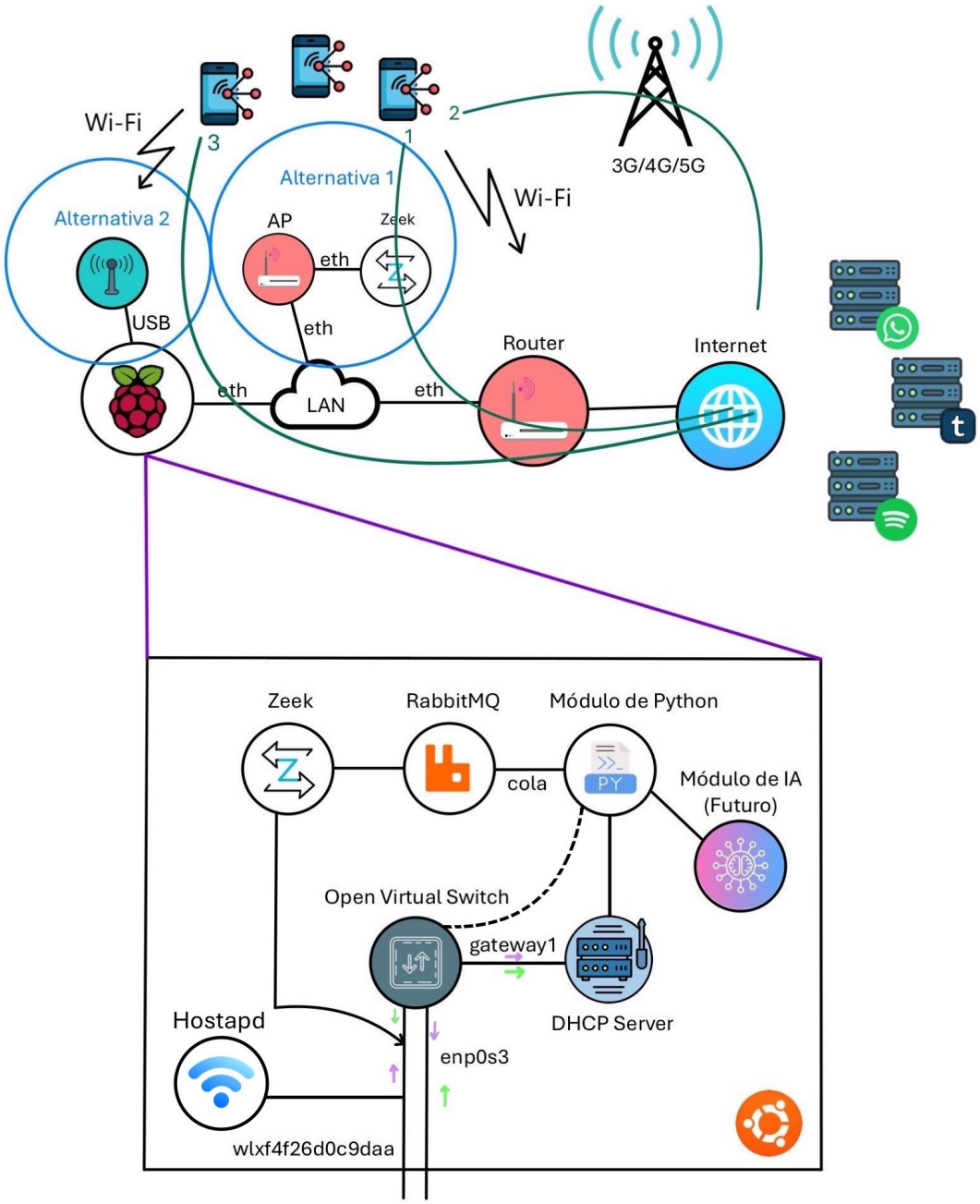

Figura 3.2: Esquema de la arquitectura desplegada en Raspberry Pi

3.2. Topología de red y configuración del entorno

Este apartado describe la topología de red común que comparten tanto el entorno virtual desarrollado en *VirtualBox* como el entorno físico desplegado sobre Raspberry Pi. A pesar de las diferencias en el hardware subyacente y los mecanismos concretos de conexión a Internet, ambos entornos reproducen la misma arquitectura lógica, lo que permite aplicar una configuración unificada en cuanto a provisión de conectividad, captura de tráfico y control del flujo de red. Esta arquitectura compartida está representada en las Figuras 3.1 y 3.2, donde se ilustran los elementos

principales y la posición que ocupa el sistema en el flujo de red.

La arquitectura diseñada para este proyecto tiene como propósito monitorizar y analizar el tráfico de red generado por dispositivos móviles al acceder a servicios de red mediante conexiones Wi-Fi. Para ello, se ha desarrollado un sistema que se sitúa entre los dispositivos del usuario y su salida a Internet, actuando como intermediario en las comunicaciones y permitiendo capturar las interacciones sin alterar el funcionamiento habitual desde la perspectiva del cliente.

Durante la fase de diseño se contemplaron dos alternativas para insertar este sistema en el flujo de red: por un lado, desplegar un punto de acceso controlado directamente por el sistema (Alternativa 1); por otro, ofrecer un punto de acceso completamente aislado, gestionado de forma autónoma desde el entorno virtual (Alternativa 2). Ambas opciones permiten interceptar el tráfico generado por los dispositivos conectados, aunque presentan implicaciones diferentes en cuanto a la integración en la red existente.

La primera alternativa plantea insertar el sistema en una red local ya configurada, en la que el punto de acceso forma parte de una LAN conectada a un *router* convencional. Esta opción permite que los dispositivos se conecten como lo harían habitualmente, con la particularidad de que el sistema puede observar el tráfico gracias a su posición intermedia. No obstante, esta configuración requiere conocer con precisión los detalles de la infraestructura de red existente, como las direcciones IP asignadas, la puerta de enlace, los rangos de direcciones en uso y los posibles conflictos con otros servidores DHCP activos. Esta dependencia introduce un grado de rigidez e incertidumbre que limita su aplicabilidad en escenarios reales o no controlados.

Por este motivo, se ha optado por implementar la segunda alternativa, basada en la creación de un punto de acceso Wi-Fi completamente independiente desde la propia máquina virtual que contiene el sistema. En este enfoque, el entorno virtual actúa como proveedor de conectividad, ofreciendo una red Wi-Fi gestionada por el propio sistema y aislada de la red local. Esta red se construye utilizando un adaptador inalámbrico USB, a través del cual se habilita un punto de acceso controlado por software. Esta alternativa ofrece una ventaja crítica: no requiere conocer ni modificar la configuración de la red local subyacente, ya que el sistema asume todas las funciones de gestión, desde la emisión de la señal inalámbrica hasta la asignación de direcciones IP y el enrutamiento hacia Internet.

No obstante, este enfoque también implica una condición práctica; para que el sistema controle completamente la conectividad, es necesario anular la red Wi-Fi preexistente, de modo que los dispositivos móviles se conecten exclusivamente al punto de acceso gestionado por el sistema.

3.2.1. Configuración de red y servicios de conectividad

Con el objetivo de asegurar el correcto funcionamiento del sistema tanto en el entorno virtual de desarrollo como en el físico de despliegue (basado en una Raspberry Pi), se ha diseñado una arquitectura de red gestionada por software. Esta arquitectura permite emitir una red Wi-Fi desde cero, controlar las conexiones entrantes y,

de forma simultánea, monitorizar el tráfico que generan los dispositivos conectados.

En lugar de emplear un *router* comercial, se ha optado por construir una red inalámbrica desde el propio sistema operativo GNU/Linux utilizando herramientas de bajo nivel. El elemento central para esta tarea es el servicio `hostapd`, que permite transformar una interfaz de red inalámbrica en un punto de acceso (AP, *Access Point*) siempre que el hardware sea compatible con dicho modo. Para ello, se requiere tanto una interfaz Wi-Fi física (por ejemplo, `wlan1`) como un controlador que implemente correctamente el estándar `nl80211`, utilizado por `hostapd` para comunicarse con el kernel de Linux.

El motivo de utilizar `hostapd` frente a soluciones más automatizadas (como el uso de un *router* doméstico) radica en la necesidad de tener un control total sobre la red; qué dispositivos se conectan, qué direcciones IP se les asignan y cómo se enruta y monitoriza su tráfico.

El punto de acceso Wi-Fi se ha implementado mediante el servicio `hostapd`, que permite transformar una interfaz inalámbrica en un AP funcional, gestionado por software. Su configuración se detalla en el Anexo A, e incluye parámetros como el nombre de la interfaz (`interface`), el SSID visible desde los dispositivos, el canal de emisión, el tipo de cifrado (WPA2-PSK), y otras opciones relacionadas con la seguridad y la gestión interna del demonio. El fichero de configuración principal se ubica en `/etc/hostapd/hostapd.conf`, y su ruta se declara explícitamente en `/etc/default/hostapd` para que el servicio pueda arrancar correctamente.

Este tipo de montaje presenta ciertas limitaciones técnicas, ya que no todos los adaptadores inalámbricos permiten operar en modo AP, y su uso dentro de una máquina virtual puede requerir pasos adicionales, como la asignación manual del adaptador USB desde el hipervisor y la instalación de controladores compatibles. Durante el desarrollo se evaluaron múltiples combinaciones de hardware y configuraciones hasta alcanzar un funcionamiento coherente y estable en ambos entornos. Finalmente, se seleccionó el adaptador TP-Link TL-WN722N v1, ampliamente compatible con `hostapd` y capaz de operar en modo AP tanto en entornos físicos como virtualizados.

El adaptador USB inalámbrico cumple, por tanto, un papel clave en esta arquitectura, ya que permite al sistema actuar simultáneamente como punto de acceso Wi-Fi y como pasarela de salida a Internet. Al estar directamente conectado a la máquina que ejecuta el sistema (ya sea una Raspberry Pi o una máquina virtual), se consigue emitir una red aislada, gestionada íntegramente por el software, lo que evita interferencias con otras redes del entorno y facilita la recolección controlada del tráfico generado por los dispositivos conectados.

En el entorno virtual se han definido dos interfaces virtuales internas, `gateway1` y `gateway2`, conectadas al switch virtual gestionado mediante OVS. Esta arquitectura permite duplicar el tráfico que atraviesa la red y enviar una copia a `gateway1`, donde puede ser analizado en tiempo real por Zeek sin interferir en el flujo original. La interfaz `gateway2` se incluyó como parte de un diseño experimental. En el entorno físico, por simplicidad, se emplea únicamente la interfaz `gateway1`. Estas diferencias estructurales se aprecian en las Figuras 3.1 y 3.2, que ilustran ambos despliegues.

Para que los dispositivos conectados a la red Wi-Fi puedan acceder a Internet, es necesario que el propio sistema (ya sea la máquina virtual o la Raspberry Pi) actúe como intermediario entre la red local y el exterior. Para ello, se define una interfaz virtual interna, llamada `gateway1` con una dirección IP estática (`192.168.100.1/24`), que servirá como punto de salida para los dispositivos conectados. Esta asignación se realiza editando el fichero `/etc/network/interfaces`, donde también se definen el modo de operación del resto de interfaces implicadas. En concreto, la interfaz cableada `enp0s3` se configura en modo manual, mientras que la interfaz virtual `gateway2` obtiene su configuración mediante DHCP. La interfaz Wi-Fi `wlxf4f26d0c9daa`, encargada de emitir la red inalámbrica a través de `hostapd`, también se activa automáticamente, aunque no requiere dirección IP asignada directamente. En el entorno virtual, esta interfaz Wi-Fi se conecta al sistema mediante un adaptador USB asignado desde el hipervisor, como se muestra en la Figura 3.1. En el caso de la Raspberry Pi (Figura 3.2), el adaptador se conecta directamente a uno de los puertos USB del dispositivo.

En cuanto al acceso a Internet, este se proporciona desde la máquina anfitriona utilizando mecanismos distintos según el entorno. En el caso virtual, la salida a la red se obtiene mediante un adaptador de red configurado en modo NAT (*Network Address Translation*), como se ilustra en la Figura 3.3. En el entorno físico, la conexión se realiza directamente a través de la interfaz Ethernet o Wi-Fi convencional del dispositivo, sin necesidad de mecanismos de virtualización.

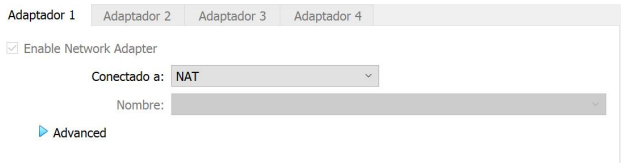

Figura 3.3: Adaptador de red configurado en modo NAT en VirtualBox

Si bien es cierto que los dispositivos podrían configurarse manualmente con una dirección IP estática, lo habitual, y más conveniente en entornos dinámicos, es que obtengan sus parámetros de red automáticamente. En este caso, además, dicha automatización resulta indispensable, ya que se busca mantener un control completo sobre la red generada, incluyendo la asignación de direcciones IP y el seguimiento del tráfico asociado a cada dispositivo.

Para ello, se ha desplegado un servidor DHCP con el servicio `isc-dhcp-server`, encargado de proporcionar automáticamente a cada cliente conectado una dirección IP válida, la puerta de enlace predeterminada y los servidores DNS. Esta solución permite que los dispositivos se incorporen al entorno monitorizado sin intervención manual y de forma integrada.

La configuración de este servidor se ha realizado a través del fichero `dhcpd.conf`, donde se definen explícitamente las características de la red gestionada. Entre los parámetros establecidos se incluyen el nombre de dominio simulado, los servidores DNS utilizados para la resolución de nombres, el rango de direcciones IP asignables, el *router* por defecto y los tiempos de concesión. De este modo, se asegura que cada dispositivo conectado reciba automáticamente todos los valores requeridos

para una operación funcional, pero dentro de un entorno completamente controlado.

Además, se ha deshabilitado la actualización dinámica de DNS mediante la opción `ddns-update-style none`, dado que no se dispone de un servidor DNS interno. También se declara la red como gestionada exclusivamente por este sistema con la opción `authoritative`, lo que evita posibles interferencias de otros servidores DHCP presentes en el entorno.

El bloque `subnet` delimita la red local asignada, estableciendo su máscara de red, la puerta de enlace (correspondiente a la interfaz que actúa como punto de acceso) y el rango de direcciones IP dinámicas asignables. La configuración completa utilizada se incluye en el Anexo C, con el objetivo de documentar el entorno operativo y permitir su reproducción.

Una vez desplegada la red inalámbrica, se configura un puente virtual denominado `br0` utilizando *Open vSwitch* (OVS), al que se conectan todas las interfaces necesarias para gestionar el tráfico. En concreto, se añaden tres elementos fundamentales; la interfaz de salida a Internet (`enp0s3`), la interfaz Wi-Fi desde la que se emite el punto de acceso (`wlxf4f26d0c9daa`), y una interfaz virtual interna denominada `gateway1`. Esta última no tiene dirección IP asignada y actúa exclusivamente como puerto espejo para la monitorización pasiva del tráfico.

El flujo de datos dentro del puente virtual gestionado por *Open vSwitch* se ha configurado para permitir la replicación selectiva del tráfico hacia interfaces internas. Concretamente, los paquetes que llegan desde la interfaz Wi-Fi, es decir, desde los dispositivos móviles conectados al punto de acceso, se reenvían por duplicado. Por un lado, hacia la interfaz de salida a Internet, permitiendo la navegación normal, y por otro, hacia la interfaz interna `gateway1`, donde pueden ser analizados por *Zeek*. Esta replicación permite inspeccionar el tráfico saliente (de cliente a servidor) de forma pasiva.

De forma simétrica, los paquetes que regresan desde Internet a través de la interfaz cableada también se duplican. En este caso, uno de los duplicados se reenvía al cliente correspondiente mediante la interfaz Wi-Fi, asegurando la entrega del contenido solicitado, mientras que el otro se redirige hacia la interfaz `gateway2`. A diferencia de `gateway1`, esta segunda interfaz no está destinada a monitorización, sino que actúa como pasarela de red interna, representa el punto de salida hacia Internet configurado para los clientes dentro del entorno virtual. Su inclusión en el flujo permite mantener la conectividad de los dispositivos conectados a la red.

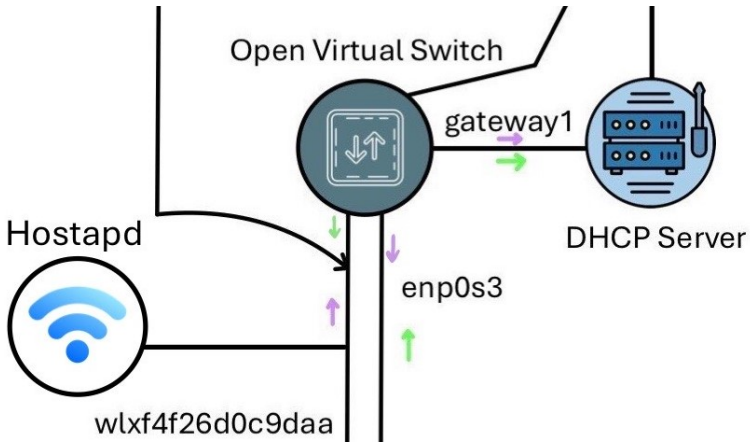

Figura 3.4: Puente virtual y replicación de tráfico en la Raspberry Pi

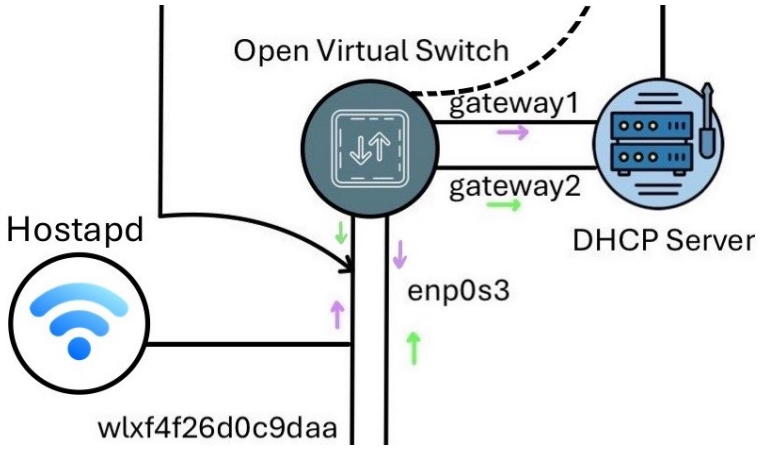

Figura 3.5: Puente virtual y replicación de tráfico en la máquina virtual

Este diseño corresponde al entorno virtual desplegado en la Figura 3.5. En el despliegue físico representado en la Figura 3.4, se ha optado por una solución más sencilla, utilizando únicamente la interfaz `gateway1`, que cumple simultáneamente el papel de punto de análisis y de pasarela hacia Internet.La configuración empleada para establecer este comportamiento se detalla en el Anexo B.

Capítulo 4

Desarrollo del sistema de monitorización

En este capítulo se presenta el desarrollo práctico del sistema de monitorización y administración del tráfico IP en redes Wi-Fi, enfocado en detectar patrones de uso excesivo o prolongado en servicios de *streaming* como *Spotify* y *TikTok*. Se detallan las fases de observación manual con Wireshark, el diseño e implementación de los scripts de *Zeek* para la identificación de eventos relevantes, y la integración con un módulo *Python* para actuar de forma automática mediante el uso de *Open vSwitch*.

4.1. Observación inicial del tráfico con Wireshark

Antes de diseñar las reglas de detección en *Zeek*, ha sido necesario realizar una fase de observación previa del tráfico real generado por las aplicaciones. Esta etapa permite identificar patrones específicos, como dominios, protocolos o secuencias de paquetes, asociados a acciones concretas del usuario. Para ello se utiliza *Wireshark*, una herramienta que permite inspeccionar el tráfico capturado sin ningún tipo de procesamiento intermedio. Aunque *Zeek* es una plataforma potente para la monitorización automatizada, no está diseñada para el descubrimiento inicial de patrones, ya que trabaja a partir de eventos ya definidos. Intentar analizar directamente con *Zeek* sin conocer previamente las características del tráfico puede conducir a reglas poco efectivas. Por este motivo, el análisis inicial se lleva a cabo con *Wireshark*, que proporciona una visión completa del tráfico en crudo, sobre la que se construyen posteriormente los scripts de *Zeek*.

Durante el proceso de análisis se planteó la posibilidad de emplear un dispositivo móvil rooteado o realizar las acciones desde un ordenador con el fin de capturar todo el tráfico de forma más controlada. Sin embargo, esta opción se descartó ya que no reflejaría fielmente el entorno real de uso de un dispositivo móvil, que constantemente está generando conexiones a distintos servicios, actualizaciones de aplicaciones y sincronizaciones en segundo plano. Así, se decidió capturar el tráfico en un escenario más cercano al día a día de un usuario, donde los datos recogidos reflejan de manera más precisa las conexiones que se producen de forma natural mientras se utilizan aplicaciones como *Spotify*, *WhatsApp* o *TikTok*.

La metodología general consistió en conectar un dispositivo móvil iOS al punto de acceso inalámbrico configurado en el entorno experimental y, desde un ordenador, iniciar capturas con la herramienta *Wireshark* sobre la interfaz del punto de

acceso. Para cada prueba se realizaron acciones concretas en las aplicaciones (por ejemplo, reproducir una canción en *Spotify* o abrir un vídeo en *TikTok*) y, tras un tiempo, se detuvo la captura. Posteriormente, se llevó a cabo un análisis manual de los paquetes recogidos, lo que permitió identificar patrones característicos de cada aplicación [11, 12, 7].

Dada la variabilidad del tráfico generado por un dispositivo en uso normal, fue necesario repetir las pruebas en varias ocasiones y comparar las capturas para aislar el tráfico propio de cada acción frente al resto de conexiones de fondo. Este procedimiento permitió identificar dominios específicos (por ejemplo, `audio-fa.scdn.co` en *Spotify* o `tiktokcdn.com` en *TikTok*) y rangos de direcciones IP asociados a las CDNs de cada servicio.

Como ejemplo ilustrativo de esta metodología se expone el caso de la reproducción de canciones en *Spotify*. En todas las capturas analizadas se observó que, por cada canción reproducida, aparecía exactamente una trama TLS con el dominio `audio-fa.scdn.co` en el campo SNI. Para confirmar este patrón se realizaron capturas adicionales reproduciendo varias canciones completas, y en todos los casos se mantuvo el mismo comportamiento. Además, se constató un retardo aproximado de 30 segundos entre el inicio de la reproducción y la aparición de la conexión. Este fenómeno responde a la estrategia de *buffering* [15] de *Spotify*, que descarga de manera anticipada un bloque de audio suficientemente amplio para garantizar una reproducción fluida sin depender de la red en tiempo real. Solo cuando el *buffer* comienza a agotarse, el cliente abre una nueva conexión al dominio `audio-fa.scdn.co` para solicitar más fragmentos.

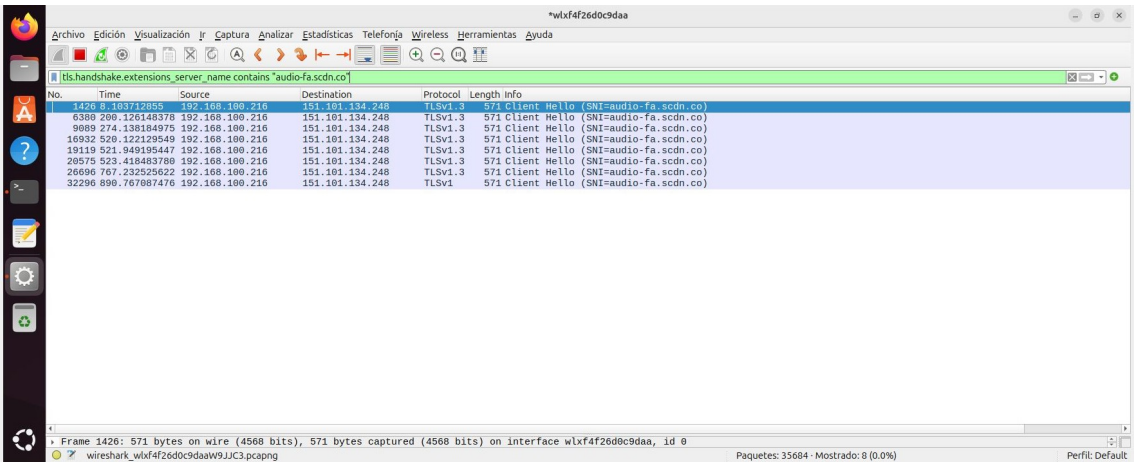

Figura 4.1: Reproducción de canciones en Spotify

La Figura 4.1 muestra un ejemplo de este comportamiento. En la captura, filtrada con `tls.handshake.extensions_server_name contains audio-fa.scdn.co`, se observa cómo al saltar rápidamente entre varias canciones aparecen varios *Client Hello* seguidos, mientras que en reproducciones normales las conexiones se espacian en torno a decenas de segundos, en consonancia con la estrategia de *buffering*. Aunque esta señal no indica el momento exacto en que se pulsa *play*, constituye una evidencia fiable de que una canción se está reproduciendo o va a reproducirse, ya que si no se inicia la reproducción no se genera ninguna conexión hacia dicho dominio.

Como segundo ejemplo representativo se muestra el comportamiento de la aplicación *TikTok*. En este caso, las capturas revelaron un patrón distinto al observado en *Spotify*, consecuencia directa de la naturaleza de su servicio de reproducción continua de vídeos cortos. En la Figura 4.2, en la que se utiliza el comando `tls.handshake.extensions_server_name contains tiktokcdn.com` para filtrar, se observa que al iniciar la visualización aparecen varias conexiones TLS hacia diferentes subdominios de la CDN de TikTok, responsables de descargar los primeros vídeos de manera individual.

De forma característica, al llegar aproximadamente al tercer vídeo se detecta un cambio en el patrón: aparecen conexiones adicionales basadas en el protocolo *QUIC* (sobre UDP), en las que se intercambian mensajes como `Initial`, `Crypto`, `Ping` y `Padding`. Este tráfico no corresponde únicamente al vídeo que el usuario está visualizando en ese momento, sino a la descarga en bloque de un conjunto mucho mayor de contenidos (en torno a 12–15 vídeos). La aplicación implementa así una estrategia de *prefetching* masivo, precargando de forma anticipada un lote completo de vídeos para garantizar que el desplazamiento rápido por el *feed* se realice sin interrupciones ni tiempos de espera perceptibles.

Este comportamiento está íntimamente relacionado con la arquitectura de *TikTok*, que se apoya en técnicas de *edge caching* y en protocolos modernos como *QUIC* para reducir la latencia y optimizar el rendimiento. La combinación de estas conexiones TLS iniciales con el tráfico posterior en *QUIC* constituye, por tanto, un indicador inequívoco de que la aplicación está en uso activo reproduciendo vídeos. A diferencia de otras conexiones de fondo que puede generar el dispositivo (sincronizaciones o actualizaciones), la aparición conjunta de dominios de la CDN de TikTok y de descargas masivas vía *QUIC* permite inferir de manera fiable la actividad de visualización de vídeos.

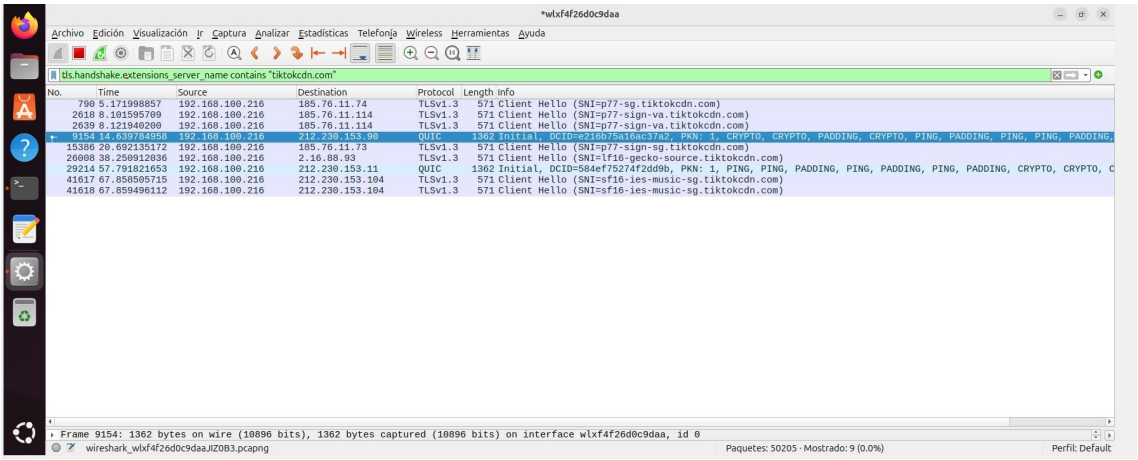

Figura 4.2: Captura de tráfico generado al visualizar vídeos en TikTok

En el caso del resto de aplicaciones, el procedimiento seguido fue el mismo; realizar capturas específicas, aislar el tráfico relevante y buscar patrones consistentes en los dominios o rangos de IP utilizados. Cabe destacar que este proceso no es inmediato, ya que exige repetir pruebas, inspeccionar manualmente los paquetes y contrastar las hipótesis con nuevas capturas hasta confirmar que el comportamiento observado se mantiene de forma estable.

A partir de los patrones identificados mediante este análisis manual, como dominios específicos, secuencias temporales características o protocolos particulares, se definen los eventos que luego deberán ser detectados automáticamente por los scripts desarrollados en *Zeek*. Es decir, cada comportamiento observado en las capturas se traduce en una condición concreta que Zeek deberá reconocer durante la monitorización en tiempo real.

4.2. Desarrollo de los scripts de Zeek para la detección de eventos

Tras la fase de observación con *Wireshark*, los patrones identificados se trasladaron a *scripts* en *Zeek*. La ejecución de estos se integra dentro de la instalación estándar, en el directorio `/usr/local/zeek/share/zeek/site/`. En dicho directorio se encuentra el fichero `local.zeek`, que actúa como punto de entrada principal; cada módulo adicional que se quiera activar debe incluirse en este fichero mediante la directiva `@load`. De esta forma, los scripts creados (`spotify.zeek`, `tiktok.zeek`, `whatsapp.zeek`) residen en ese mismo directorio y se cargan automáticamente cuando Zeek arranca, sin necesidad de modificar la instalación base.

En el caso de *Spotify*, se ha implementado un módulo específico capaz de analizar el tráfico cifrado generado por la aplicación y, a partir de metadatos como el nombre del servidor o la duración de la conexión, clasificar cada interacción observada en categorías funcionales (acceso, reproducción o actividad en segundo plano). Este análisis se lleva a cabo mediante el sistema de eventos de Zeek, sin necesidad de descifrar el contenido de los paquetes. En particular, el script responde al evento `ssl_established`, que es lanzado por Zeek cada vez que se establece una conexión TLS, y permite acceder a campos como el `server_name` (extraído del SNI del handshake), la IP de destino, el protocolo usado y, una vez finaliza la conexión, su duración total.

A partir de estos datos, el script evalúa una serie de reglas diseñadas tras estudiar capturas reales con Wireshark. Por ejemplo, las conexiones a `audio-fa.scdn.co` cuya duración supera los 25 segundos se consideran representativas de una reproducción de contenido musical, ya que ese dominio corresponde al sistema de entrega de audio en streaming de Spotify. Por otro lado, las conexiones breves, de menos de 3 segundos, a dominios como `login5.spotify.com` o `spclient.wg.spotify.com` suelen coincidir con el inicio de sesión, la apertura de la aplicación o la validación de tokens de usuario. En aquellos casos donde no se puede extraer un nombre de servidor (por ejemplo, cuando se emplea QUIC y no se observa SNI), el script utiliza la dirección IP de destino como criterio alternativo. Si esta IP pertenece a un rango previamente identificado como parte de la infraestructura de Spotify, y la conexión es corta, se interpreta como una actividad secundaria, como una reapertura o una sincronización en segundo plano.

Todas las decisiones de clasificación se realizan dentro del propio script de Zeek, que genera un registro estructurado por cada evento detectado. Este registro incluye información como la IP origen, la IP destino, el protocolo utilizado, el nombre del servidor (si está disponible), la duración de la conexión y el tipo de evento. Posteriormente, estos registros pueden ser consumidos por otros módulos del sistema, como el componente de control escrito en Python, que se encarga de aplicar las reacciones correspondientes.

El siguiente algoritmo, expresado en pseudocódigo, resume la lógica implementada en el script de detección de Spotify:

Algorithm 2: Detección de eventos en Spotify con Zeek

```
Inicializar lista de dominios_reproducción ← {audio-fa.scdn.co}
Inicializar lista de dominios_acceso ← {login5.spotify.com, ...}
Inicializar conjunto de rangos_CDN con IPs conocidas de Spotify
while se detecta una nueva conexión c do
    Extraer SNI ← nombre de servidor extraído de c
    Calcular duración ← tiempo de vida de la conexión
    Identificar protocolo ← SSL o QUIC en función del puerto usado
    if SNI ∈ dominios_reproducción y duración > 25s then
        Clasificar evento como REPRODUCCIÓN
    else if SNI ∈ dominios_acceso y duración < 3s then
        Clasificar evento como ACCESO
    else if SNI está vacío y IP ∈ rangos_CDN y duración < 3s then
        Clasificar evento como MINIMIZACIÓN
    else
        Clasificar evento como POSIBLE_SPOTIFY
    Crear registro con {timestamp, IP origen/destino, puerto, protocolo,
     SNI, duración, tipo de evento}
    Guardar en el log personalizado de Zeek
    Enviar el evento como mensaje estructurado a RabbitMQ
```

Por otro lado, el módulo desarrollado para *TikTok* emplea un enfoque similar al de *Spotify*, aunque ajustado a las características específicas de la plataforma y a su patrón distintivo de tráfico. En este caso, el objetivo del script es identificar momentos de uso activo de la aplicación, en especial aquellos asociados a la visualización de vídeos o a interacciones breves con la interfaz de usuario.

El script responde a eventos generados por Zeek tanto en conexiones TLS como en aquellas establecidas mediante QUIC (`quic_server_name`), lo cual permite abarcar la mayoría del tráfico generado por TikTok, incluyendo conexiones sobre TCP y UDP. Durante el análisis, Zeek extrae metadatos como el nombre del servidor (cuando está disponible en el *Server Name Indication*, SNI), la duración de la conexión y el volumen de datos intercambiados en ambos sentidos

La detección se basa en tres estrategias complementarias que permiten inferir el uso real de la aplicación sin necesidad de inspeccionar el contenido de los paquetes. En primer lugar, las conexiones hacia dominios de entrega de contenido como `tiktokcdn.com`, `tiktokcdn-eu.com` o `tiktokv.com`, observadas tanto sobre TLS como QUIC, se interpretan como eventos de `VISUALIZACIÓN`, ya que corresponden a la descarga activa de vídeos. En segundo lugar, la aparición de conexiones breves con subdominios específicos como `api`, `mon`, `frontier`, `v16m` o `v45`, siempre que la duración sea inferior a 1 segundo y el tamaño de los paquetes reducido (menos de 3 kB enviados y 10 kB recibidos), se interpreta como una `INTERACCIÓN` breve del usuario con la aplicación. Por último, cuando no se puede extraer el SNI (por ejemplo, en conexiones QUIC con cifrado extremo), el sistema recurre a la dirección IP de destino. Si esta IP pertenece a un conjunto de subredes previamente identificadas como parte de la infraestructura de TikTok, y se establece conexión en el puerto 443, se asume razonablemente que se trata de un evento de `VISUALIZACIÓN`

Al igual que en el módulo de Spotify, cada conexión que cumpla con alguno de los criterios anteriores genera un registro estructurado que incluye información como direcciones IP y puertos, nombre del servidor (si está disponible), duración de la conexión, tamaño de los paquetes, protocolo utilizado y tipo de evento detectado. Este registro se guarda en un log personalizado y se publica como mensaje estructurado a través de RabbitMQ, permitiendo al módulo de control en Python aplicar bloqueos u otras acciones en tiempo real

El siguiente algoritmo resume la lógica implementada en el script de detección de TikTok:

Algorithm 3: Detección de eventos en TikTok con Zeek

Inicializar lista de `dominios_CDN` ← {`tiktokcdn.com,tiktokv.com ...`}
Inicializar patrones de `subdominios_API` ← `mon`, `frontier`, `v16m`, `v45`, ... ;
Inicializar conjunto de `rangos_TikTok` con IPs de sus CDNs conocidas ;
while *se detecta una nueva conexión c* **do**
  Extraer *SNI* ← nombre de servidor (si existe) ;
  Calcular *duración* ← tiempo de vida de la conexión ;
  Calcular *bytes_orig* y *bytes_resp* ← tamaños de tráfico ;
  Identificar *protocolo* ← SSL o QUIC ;
  if *SNI contiene dominio de* ***dominios_CDN*** **then**
    Clasificar evento como `VISUALIZACIÓN`
  else if *SNI contiene patrón de* ***subdominios_API*** ***y*** *duración <1s* ***y*** *bytes_orig <3000* ***y*** *bytes_resp <10000* **then**
    Clasificar evento como `INTERACCIÓN`
  else if *no hay SNI* ***y*** *IP destino ∈* ***rangos_TikTok*** ***y*** *puerto destino = 443* **then**
    Clasificar evento como `VISUALIZACIÓN`
  else
    Ignorar o clasificar como `INDETERMINADO`
  Crear registro con {*timestamp*, IPs, puertos, protocolo, duración, tipo de evento, etc.}
  Guardar en el log personalizado de Zeek
  Enviar el evento como mensaje estructurado a RabbitMQ

También se desarrolló un script para detectar eventos relacionados con el uso de *WhatsApp*, centrado en identificar el envío o recepción de mensajes. A diferencia de otras aplicaciones, WhatsApp genera una gran cantidad de conexiones breves y cifradas en segundo plano, que dificultan la identificación precisa de acciones concretas.

En este caso, la detección se encuentra limitada únicamente a mensajes de texto. Además, se comprobó que esta detección solo resulta efectiva en dispositivos Android, ya que en terminales iOS la información de red relevante (como direcciones IP y puertos) se encuentra completamente cifrada a nivel del sistema operativo, impidiendo su inspección por parte del sistema de monitorización. Esta limitación fue inesperada inicialmente, ya que otros servicios como *Spotify* o *TikTok* sí permitían observar conexiones útiles incluso desde dispositivos iOS.

El script responde tanto a conexiones cifradas mediante TLS (puerto 443) como a aquellas que utilizan el puerto 5222/TCP, característico del protocolo *XMPP*, que WhatsApp emplea como canal de señalización entre cliente y servidor. En ambos casos, se analizan metadatos como el nombre del servidor, la duración total de la conexión y el volumen de datos intercambiados en cada sentido.

Cuando se detecta una conexión hacia dominios como `whatsapp.net`, y esta supera ciertos umbrales de duración y volumen de datos recibidos, se interpreta como un evento de `ENVÍO/RECEPCIÓN DE MENSAJE`. Alternativamente, si el puerto de destino es el 5222, se considera también indicativo de actividad relacionada con WhatsApp, incluso si no se dispone de información sobre el nombre del servidor. En última instancia, cuando no es posible extraer el SNI, pero la IP destino pertenece a un rango previamente identificado como parte de la infraestructura de WhatsApp, la conexión se clasifica como `POSIBLE WHATSAPP`.

Los valores de umbral utilizados (más de 5 segundos de duración y más de 10 kB de datos recibidos) se definieron tras analizar múltiples capturas reales con *Wireshark*, en las que se observó que la mayoría de las conexiones breves o silenciosas, generadas por procesos de sincronización o notificaciones en segundo plano, no superaban dichos límites. En cambio, cuando el usuario enviaba o recibía mensajes activamente, se establecían conexiones con una duración superior y un volumen de tráfico más elevado.

Al igual que en los módulos anteriores, cada evento identificado genera un registro que se almacena en un log personalizado de Zeek y se publica como mensaje estructurado a través de RabbitMQ.

El siguiente algoritmo resume la lógica implementada en el script de detección de WhatsApp:

Algorithm 4: Detección de eventos en WhatsApp con Zeek

```
Inicializar lista de dominios_WhatsApp ← {*.whatsapp.net,
 *.whatsapp.com, ...}
Inicializar conjunto de rangos_WhatsApp con subredes de su infraestructura
Inicializar puerto puerto_XMPP ← 5222/tcp ;
while se detecta una nueva conexión c do
    Extraer SNI (si está disponible) ;
    Calcular duración de la conexión ;
    Calcular bytes_orig y bytes_resp ;
    Obtener IP y puerto de destino ;
    Identificar protocolo ← SSL, QUIC o XMPP ;
    if puerto destino == 5222 then
        Clasificar evento como WHATSSAP
    else if SNI contiene dominio de dominios_WhatsApp then
        if duración > 5s y bytes_resp > 10000 then
            Clasificar como ENVÍO/RECEPCIÓN DE MENSAJE
    else if SNI vacío y IP destino ∈ rangos_WhatsApp then
        Clasificar como POSIBLE WHATSAPP
    else
        Clasificar como INDETERMINADO
    Crear registro con {timestamp, IPs, puertos, duración, bytes, protocolo,
     tipo de evento}
    Guardar en log personalizado de Zeek
    Publicar evento como mensaje estructurado en RabbitMQ
```

De forma transversal a todos los módulos implementados, los eventos detectados se registran localmente y se reenvían al sistema de procesamiento externo para su análisis y actuación. Para ello, cada script utiliza la instrucción `Log::write` de *Zeek*, generando entradas en un fichero de log personalizado a partir de un registro estructurado denominado `Info`. Este registro encapsula campos como las direcciones IP de origen y destino, los puertos implicados, el nombre del servidor (si está disponible), el protocolo de transporte utilizado y una etiqueta descriptiva del tipo de evento detectado.

Además de su almacenamiento local, cada evento es exportado en formato *JSON* y publicado en un canal de *RabbitMQ*, desde donde puede ser consumido por el módulo de control desarrollado en Python. Para ello, cada script de Zeek realiza una llamada al comando del sistema que ejecuta un cliente de publicación (por ejemplo, `rabbitmqadmin`), utilizando un *exchange* de tipo `topic` configurado previamente en el servidor de mensajería.

El *routing key* del mensaje se define a partir del tipo de evento detectado, como por ejemplo `spotify.key`, lo que permite al consumidor filtrar de forma eficiente únicamente aquellos eventos relevantes para la acción que desea aplicar

El módulo de control en Python escucha continuamente este *exchange* mediante una cola suscrita a un patrón de `routing keys`, procesa cada mensaje recibido y, en función de su contenido, aplica las acciones correspondientes (por ejemplo, el

bloqueo temporal del tráfico mediante *Open vSwitch*).

A modo de ejemplo, el siguiente fragmento muestra un mensaje típico generado por el script de Spotify y enviado a través de RabbitMQ:


```
{
  "ts": "2025-08-21T10:52:31Z",
  "uid": "C6a3n91ZgH6qk8",
  "id": {
    "orig_h": "192.168.100.216",
    "orig_p": 61043,
    "resp_h": "151.101.134.248",
    "resp_p": 443
  },
  "orig_h": "192.168.100.216",
  "orig_p": 61043,
  "resp_h": "151.101.134.248",
  "resp_p": 443,
  "server_name": "audio-fa.scdn.co",
  "protocol": "SSL",
  "nota": "SNI SPOTIFY"
}
```


4.3. Procesamiento de eventos y medidas de actuación

Una vez que *Zeek* identifica un evento relevante, como una reproducción prolongada en *Spotify*, este se registra localmente y se transforma en un mensaje estructurado en formato `JSON`. Posteriormente, se reenvía a través de la herramienta de línea de comandos `amqp-publish`, que actúa como intermediario entre el script de *Zeek* y el sistema de mensajería *RabbitMQ*. Para ello, se define un `exchange` de tipo `direct`, denominado `zeek_exchange`, al que se publican los mensajes junto con una clave de enrutamiento específica que identifica el tipo de tráfico detectado, como `spotify_key` o `tiktok_key`.

En el otro extremo, un módulo desarrollado en *Python* permanece en escucha sobre este `exchange`, y procesa en tiempo real los mensajes que van llegando. Su función principal es aplicar medidas de reacción en función del tipo de evento recibido. Por ejemplo, puede bloquear temporalmente el tráfico generado por un dispositivo si se detecta un patrón de uso abusivo.

El siguiente algoritmo describe el funcionamiento general del módulo de reacción:

Algorithm 5: Procesamiento de eventos en el módulo Python

```
Inicializar conjunto bloqueadas como vacío
Conectar con el servidor RabbitMQ
Declarar exchange zeek_exchange de tipo direct
Crear una cola temporal exclusiva
Vincular la cola a las claves spotify_key, tiktok_key ...
while se recibe un nuevo mensaje desde RabbitMQ do
    Decodificar el contenido del mensaje
    if el mensaje no contiene la etiqueta buscada then
        Ignorar el evento
    Extraer IP de origen y destino
    if la IP de origen no se puede extraer then
        Mostrar advertencia y continuar
    if la IP ya está en la lista bloqueadas then
        Ignorar el evento
    Esperar 30 segundos
    Ejecutar comando ovs-ofctl add-flow br0
     "priority=1000,ip,nw_src=<IP>,actions=drop"
    Añadir la IP a la lista bloqueadas
    Esperar 60 segundos
    Ejecutar comando ovs-ofctl del-flows br0 ïp,nw_src=<IP>"
    Eliminar la IP de la lista bloqueadas
```

Cuando se identifica un evento relevante, como una reproducción continuada en *Spotify*, el sistema localiza la dirección IP del dispositivo implicado y aplica una regla temporal de corte de tráfico. Este bloqueo se mantiene durante un periodo predefinido y, posteriormente, se restablece automáticamente la conectividad mediante la eliminación de la regla, simulando así una interrupción breve que actúe como mecanismo disuasorio. Todo este proceso se realiza a través de *Open vSwitch* (OVS), donde se inserta una regla de flujo de alta prioridad que descarta todo el tráfico IP cuyo origen coincide con la dirección implicada.

Además de actuar ante eventos individuales, el módulo Python incorpora una lógica acumulativa que permite detectar patrones de uso reiterado. Por ejemplo, si un mismo dispositivo reproduce varias canciones en un intervalo corto, aunque cada reproducción por sí sola no implique una acción, el sistema considera el conjunto como uso abusivo y aplica igualmente una medida de bloqueo. Esta capacidad de observación longitudinal permite detectar comportamientos sostenidos que no serían identificables con un análisis puntual.

Para garantizar que las medidas se apliquen sobre el dispositivo correcto, el sistema mantiene una tabla actualizada que relaciona direcciones IP con direcciones MAC. Esta asociación es fundamental en entornos con asignación dinámica, donde la IP de un dispositivo puede variar con el tiempo.

Aunque la lógica de actuación puede extenderse a múltiples aplicaciones, es importante definir criterios específicos para cada una, especialmente cuando se comparten infraestructuras comunes. Durante las pruebas, se observó que algunas conexiones de *Instagram* compartían servidores con *WhatsApp*, debido a que ambas pertenecen a la red de *Meta*. Esto pone de manifiesto la necesidad de aplicar reglas de filtrado más allá de las IPs, empleando atributos como el puerto de destino, el protocolo utilizado o el volumen de datos transferidos.

Capítulo 5

Experimentos y resultados

5.1. Entorno y metodología de pruebas

Para comprobar el correcto funcionamiento del sistema se realizaron pruebas en un entorno controlado, donde tanto la máquina virtual como la raspberry actuaban como punto de acceso Wi-Fi. Los dispositivos conectados generaban tráfico que era interceptado y analizado por *Zeek*. Los eventos detectados se enviaban a través de *RabbitMQ* a un módulo *Python* encargado de aplicar acciones de control, como cortar el tráfico mediante reglas OvS.

Durante la fase de desarrollo, se utilizó una funcionalidad del sistema operativo del móvil para limitar el uso de aplicaciones, de modo que únicamente permaneciera activa la que se deseaba probar (por ejemplo, *Spotify* o *TikTok*). Esto permitió aislar el comportamiento de cada aplicación y verificar la correcta detección y clasificación del tráfico por parte del sistema.

Sin embargo, en las pruebas finales se optó por mantener el dispositivo móvil sin restricciones, en condiciones de uso normales, para validar el sistema en un escenario realista. En estas pruebas, se accedía libremente a diversas aplicaciones (*Spotify*, *TikTok*, *WhatsApp*, etc.) y se observaba cómo el sistema reaccionaba ante cada tipo de tráfico, incluso en presencia de procesos en segundo plano u otras conexiones simultáneas.

Cabe destacar que no fue necesario modificar ningún parámetro del sistema ni realizar cambios en el código para permitir el uso libre del dispositivo durante estas pruebas, ya que el sistema está diseñado para monitorizar pasivamente todo el tráfico de red, independientemente de la actividad concreta o de las aplicaciones utilizadas.

5.2. Resultados obtenidos

Para ilustrar de manera concreta el funcionamiento del sistema, se presenta un ejemplo completo en el que se detecta un flujo de *Spotify* y se aplica una medida de corte temporal. La Figura 5.1 representa el recorrido completo de un paquete en el sistema, desde que el usuario pulsa *play* hasta que se ejecuta la acción de bloqueo. Cada paso del diagrama está numerado y se corresponde con los eventos observados en las siguientes capturas de tráfico.

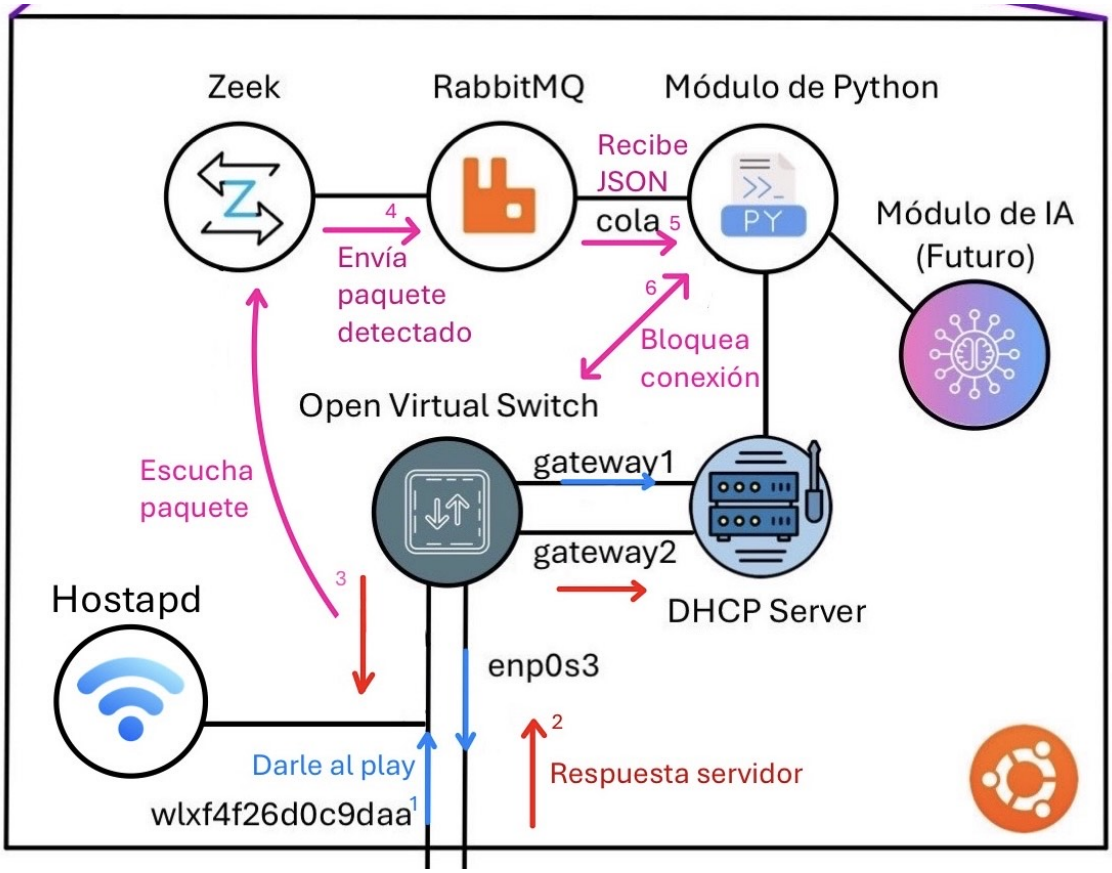

Figura 5.1: Flujo completo del sistema ante una reproducción en Spotify

Todo comienza cuando el usuario pulsa *play* en una canción desde su dispositivo móvil, conectado a la red Wi-Fi emitida por el punto de acceso. Esta acción genera una conexión saliente que atraviesa el puente virtual configurado, saliendo por la interfaz `enp0s3` hacia el exterior. El servidor de *Spotify* responde con una conexión TLS 1.3 que llega de vuelta al sistema. En este punto, `Open vSwitch` replica de forma interna todo el tráfico y lo envía también hacia la interfaz `gateway1`, donde puede ser analizado de forma pasiva por *Zeek*.

En la Figura 5.2 se muestra una de estas capturas de tráfico realizada con *Wireshark*. Concretamente, puede observarse el establecimiento de una conexión TLS entre la dirección IP local del dispositivo (`192.168.100.216`) y el servidor de destino `146.75.90.248`. El mensaje `Client Hello` incluye el campo *Server Name Indication (SNI)* con el dominio `audio-fa.scdn.co`. Este dominio actúa como identificador único del servicio, permitiendo reconocer el tráfico específico de reproducción de audio entre todas las conexiones cifradas del sistema.

Este paquete duplicado es analizado por *Zeek*, que ha sido configurado con scripts personalizados para detectar este patrón concreto. Al identificar la conexión como una reproducción de audio, Zeek genera un evento que encapsula en un mensaje `JSON` y lo envía a través de *RabbitMQ*. En la Figura 5.3 se muestra el resultado de esta detección, donde el flujo se clasifica como *reproducción* y se registra la dirección IP del cliente junto con el dominio implicado.

El mensaje es recibido por el módulo de control en *Python*, que evalúa si el evento cumple las condiciones necesarias para aplicar una medida de control. En caso afirmativo, el módulo ejecuta una orden sobre *Open vSwitch* que introduce una nueva regla de flujo destinada a bloquear la comunicación entre el dispositivo y el servidor correspondiente. De este modo, se interrumpe automáticamente el tráfico sin afectar al resto de la red.

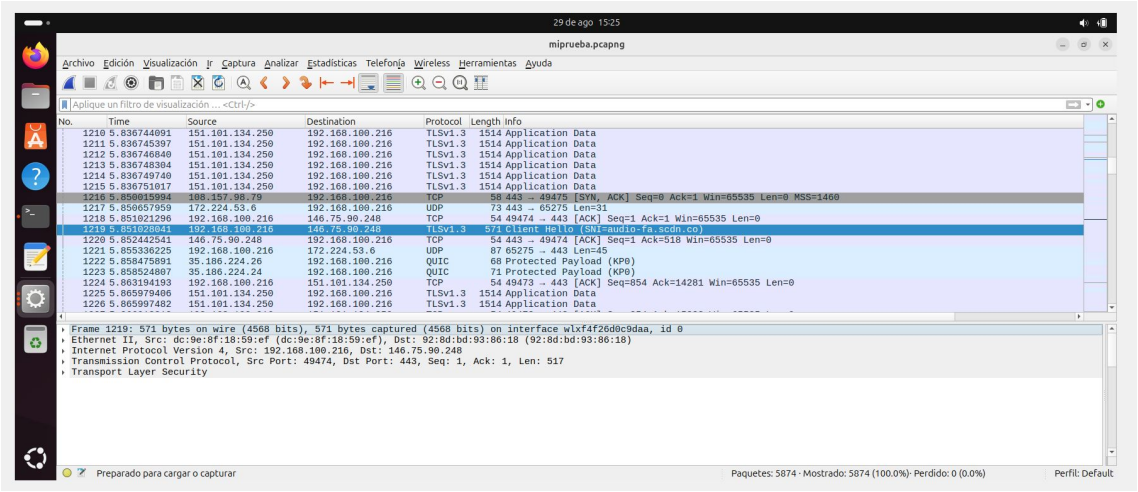

Figura 5.2: Captura en Wireshark de un flujo TLS.

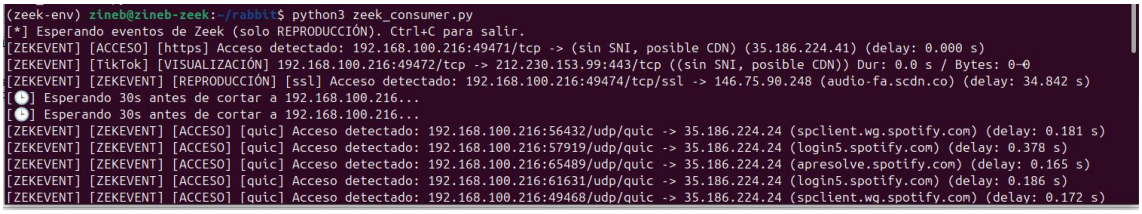

Figura 5.3: Evento correspondiente detectado por Zeek.

Durante las pruebas finales del sistema, se realizaron múltiples sesiones de navegación en las que se accedió a servicios como *Spotify*, *Whatssap* o *TikTok* desde diferentes dispositivos conectados al punto de acceso. El objetivo era comprobar la capacidad de detección del sistema ante eventos relevantes, como el inicio de reproducción de contenido en segundo plano.

En la Figura 5.4 se muestran algunos de los eventos capturados por el módulo *Python*, incluyendo accesos, reproducciones y medidas aplicadas como el corte de tráfico. Estos eventos son recibidos desde *Zeek* en tiempo real y tratados por el sistema.

Figura 5.4: Eventos recibidos por el módulo Python desde Zeek

En algunos casos, como puede observarse en la Figura 5.5, el sistema aplicó correctamente el corte de tráfico al detectar un patrón de uso intensivo, bloqueando la comunicación saliente del dispositivo implicado.

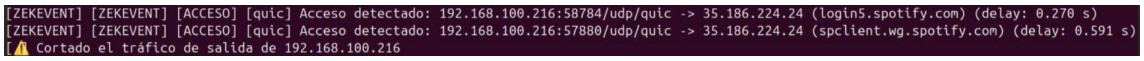

Figura 5.5: Ejemplo de corte de tráfico tras detección de reproducción

5.2.1. Pruebas de validación y posibles escenarios en producción

Las pruebas realizadas a lo largo de este proyecto se desarrollaron en un entorno controlado, creado con el objetivo de facilitar el análisis del tráfico generado por acciones concretas en aplicaciones como Spotify o TikTok. Para ello, se cerraron todas las aplicaciones en segundo plano y se limitaron al máximo las interacciones del dispositivo, con el fin de reducir el ruido y poder identificar los patrones con mayor claridad. Este enfoque permitió afinar los scripts de detección y comprobar que el sistema respondía correctamente ante eventos concretos y repetibles.

Sin embargo, este tipo de pruebas, aunque muy útiles en fase de desarrollo, no reflejan todas las situaciones que pueden darse en un entorno real. En un contexto de producción, los dispositivos suelen tener varias aplicaciones abiertas, generan tráfico de fondo de manera constante (sincronizaciones, notificaciones, actualizaciones, etc.), y las condiciones de red o el comportamiento del usuario son mucho más variables. Todo esto podría influir en cómo el sistema detecta los eventos o aplica las acciones de control.

Por eso, si se quisiera desplegar este sistema en un entorno real, sería necesario probarlo también en escenarios más realistas. Sería interesante, por ejemplo, realizar sesiones con varios dispositivos conectados a la vez, utilizar diferentes tipos de tráfico a la vez (vídeos, mensajería, navegación, etc.), o incluso introducir variaciones en la red, como aumentos de latencia o pequeñas pérdidas de paquetes. También se podrían hacer pruebas de estrés, generando muchos eventos en poco tiempo, para comprobar si el sistema es capaz de reaccionar bien sin saturarse.

Gracias a estas pruebas más avanzadas se podrían obtener datos objetivos sobre cómo funciona el sistema en condiciones reales; si detecta bien los eventos, cuánto tarda en reaccionar, cuánta carga impone al sistema o si genera errores (como falsos

positivos o negativos). Todo ello permitiría seguir mejorando el sistema y prepararlo para un posible uso más general.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

En conclusión, el sistema desarrollado ha demostrado ser capaz de detectar de forma precisa el uso de aplicaciones populares como *WhatsApp*, *Spotify* o *TikTok* en redes Wi-Fi locales, sin necesidad de inspeccionar directamente el contenido de los paquetes. A lo largo de este trabajo se ha implementado una lógica basada en el análisis de patrones de tráfico, como la duración de las conexiones, la frecuencia de acceso o la presencia de ciertos dominios, que permite inferir distintos tipos de actividad, desde una simple apertura de la aplicación hasta la reproducción sostenida de contenido multimedia.

Una de las dificultades más relevantes ha sido identificar patrones de comportamiento que resultaran verdaderamente representativos y transformarlos en reglas programables. Cada aplicación presenta características muy particulares en su comportamiento en red, con conexiones que a menudo son breves, dispersas o difíciles de distinguir de otras actividades. Lograr que el sistema interpretase correctamente estas señales, minimizando al mismo tiempo los falsos positivos, ha exigido un trabajo intensivo de observación, experimentación y ajuste fino. En este proceso resultó especialmente complejo establecer reglas de detección que fueran representativas del uso real. Por ejemplo, en *Spotify* fue necesario diferenciar entre conexiones muy breves, que corresponden a la apertura de la aplicación o a peticiones de control, y aquellas más prolongadas, asociadas a la reproducción de canciones. En *TikTok* por ejemplo, las dificultades vinieron dadas por la naturaleza dinámica del servicio, con múltiples conexiones cortas y continuas que aparecen incluso cuando se cargan varios vídeos de manera anticipada, lo que complica decidir a partir de qué momento debe considerarse una reproducción efectiva.

A pesar de ciertas limitaciones, especialmente en lo relativo a la heterogeneidad de dispositivos y escenarios de prueba, los resultados obtenidos han sido satisfactorios. El sistema ha mostrado un rendimiento sólido en condiciones reales, diferenciando con precisión entre usos puntuales y patrones de uso más intensivo. Asimismo, se ha incorporado un mecanismo de actuación capaz de aplicar medidas de control automático en caso de detectar un consumo excesivo, con el objetivo de fomentar un uso más equilibrado de la red.

En conjunto, este trabajo pone de manifiesto que es posible desarrollar un sistema de monitorización eficiente y adaptable sin necesidad de recurrir a soluciones complejas o invasivas. A través de una observación cuidadosa del comportamiento del tráfico, es viable construir herramientas útiles para la gestión responsable del uso de aplicaciones en entornos compartidos.

6.2. Trabajo futuro

El sistema propuesto en este proyecto se basa actualmente en reglas y patrones estáticos, lo que supone una limitación para adaptarse a las variaciones y a los nuevos comportamientos que aparecen en los servicios digitales, siempre en evolución. Por ello, como línea de trabajo futuro, se plantea la integración de un sistema de inteligencia artificial que pueda aprender y detectar de forma más automática estos patrones en el tráfico de red.

La idea sería aprovechar los registros obtenidos durante las pruebas para entrenar un modelo de aprendizaje supervisado, capaz de reconocer y clasificar las distintas interacciones en función del tráfico generado. Esto permitiría que el sistema no dependa únicamente de reglas fijas, adaptándose a cambios y comportamientos que no se hayan contemplado inicialmente, y reduciendo así la necesidad de intervención manual.

Con el tiempo, la integración de inteligencia artificial podría aportar mayor precisión y flexibilidad en el análisis en tiempo real, identificando incluso patrones complejos o poco frecuentes. De esta manera, se busca que el sistema pueda evolucionar y adaptarse de manera progresiva a las nuevas necesidades que surjan en el análisis del tráfico de red.

Bibliografía

[1] Jean M. Twenge y W. Keith Campbell. «Associations between screen time and lower psychological well-being among children and adolescents: Evidence from a population-based study». En: *Preventive Medicine Reports* 12 (2018), págs. 271-283. DOI: 10.1016/j.pmedr.2018.10.003. URL: https://doi.org/10.1016/j.pmedr.2018.10.003.

[2] Long Nguyen-Vu et al. «Android Rooting: An Arms Race between Evasion and Detection». En: *Security and Communication Networks* 2017 (2017), págs. 1-17. DOI: 10.1155/2017/4121765. URL: https://onlinelibrary.wiley.com/doi/10.1155/2017/4121765 (visitado 04-06-2025).

[3] CeNIT - Communication Networks and Information Technologies. *Grupo de investigación CeNIT*. 2024. URL: https://i3a.unizar.es/es/grupos-de-investigacion/cenit (visitado 13-05-2025).

[4] Napatech. *What is a flow?* Ilustración del concepto de flujo IP mediante 5-tuple. 2024. URL: https://www.napatech.com/what-is-a-flow/ (visitado 15-05-2025).

[5] Nick McKeown et al. «OpenFlow: Enabling Innovation in Campus Networks». En: *Proceedings of the ACM SIGCOMM Conference on Data Communication*. Association for Computing Machinery, 2008, págs. 69-74. DOI: 10.1145/1355734.1355746. URL: https://dl.acm.org/doi/pdf/10.1145/1355734.1355746 (visitado 21-02-2025).

[6] Ben Pfaff et al. «The Design and Implementation of Open vSwitch». En: *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, 2015, págs. 117-130. URL: https://www.usenix.org/system/files/conference/nsdi15/nsdi15-paper-pfaff.pdf (visitado 21-02-2025).

[7] B. Kalaiselvi y K. Aruna. «Network Traffic Analysis Using Wireshark». En: *International Journal of Research Publication and Reviews* 4.11 (2023), págs. 1960-1965. DOI: 10.55248/gengpi.4.1223.123506. URL: https://ijrpr.com/uploads/V4ISSUE11/IJRPR19300.pdf.

[8] Steffen Haas, Robin Sommer y Mathias Fischer. «zeek-osquery: Host-Network Correlation for Advanced Monitoring and Intrusion Detection». En: *arXiv preprint arXiv:2002.04547* (2020). URL: https://arxiv.org/pdf/2002.04547v2 (visitado 16-02-2025).

[9] Azwad Hasan, Md Omar Faisal y Sariful Haque Khan. «Exploring SDN Based Firewall and NAPT: A Comparative Analysis with Iptables and OVS in Mininet». En: *ResearchGate* (2024). URL: https://www.researchgate.net/publication/384062702_Exploring_SDN_Based_Firewall_and_NAPT_A_Comparative_Analysis_with_Iptables_and_OVS_in_Mininet (visitado 23-02-2025).

[10] An Wang et al. «UMON: Flexible and Fine-Grained Traffic Monitoring in Open vSwitch». En: *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*. Association for Computing Machinery, 2015, 15:1-15:13. DOI: 10.1145/2716281.2836100. URL: https://dl.acm.org/doi/pdf/10.1145/2716281.2836100 (visitado 09-02-2025).

[11] Ruchi Tuli. «Analyzing Network Performance Parameters Using Wireshark». En: *International Journal of Network Security & Its Applications (IJNSA)* 15.1 (2023), págs. 1-10. DOI: 10.5121/ijnsa.2023.15101. URL: https://aircconline.com/ijnsa/V15N1/15123ijnsa01.pdf.

[12] Ryufath Alief Adhyaksa Putera Soepeno. «Wireshark: An Effective Tool for Network Analysis». En: *ResearchGate* (2023). URL: https://www.researchgate.net/publication/374675769_Wireshark_An_Effective_Tool_for_Network_Analysis.

[13] John Hurley. *Evolving Stateful Firewalling: OVS+iptables, OVS+Conntrack, and Conntrack Acceleration*. Presentación de John Hurley en Netronome que detalla la evolución de los firewalls con estado y la integración de Conntrack con Open vSwitch. 2016. URL: https://www.youtube.com/watch?v=Wx9ZqCYW5sk.

[14] RabbitMQ. *AMQP Concepts Guide*. Guía conceptual que explica los intercambios, colas, enlaces y otras nociones clave de AMQP en RabbitMQ. 2024. URL: https://www.rabbitmq.com/tutorials/amqp-concepts (visitado 02-03-2025).

[15] Anika Schwind et al. «QoE Analysis of Spotify Audio Streaming and App Browsing». En: (2019), págs. 1-6. DOI: 10.1145/3349611.3355546.

[16] Flaticon. *Flaticon - Free icons and stickers*. 2024. URL: https://www.flaticon.com (visitado 07-06-2025).

[17] Freepik. *Freepik - Graphic resources for everyone*. 2024. URL: https://www.freepik.com (visitado 07-06-2025).

[18] SeekLogo. *SeekLogo - Vector Logos for free download*. 2024. URL: https://seeklogo.com (visitado 07-06-2025).

Créditos y licencias de recursos gráficos

En el Capítulo 3 de este trabajo se han utilizado dos imágenes que representan la arquitectura y la topología del sistema desarrollado. Para su elaboración se han empleado iconos y logotipos procedentes de las siguientes plataformas, bajo licencias gratuitas con atribución:

- Iconos descargados de Flaticon [16], creados por diversos autores y utilizados bajo licencia gratuita con atribución.
- Recursos vectoriales decorativos de Freepik [17], empleados como base visual para la composición de los diagramas del sistema.
- Logotipos de herramientas como Zeek, RabbitMQ, Open vSwitch o Wireshark obtenidos de SeekLogo [18], utilizados únicamente con fines académicos para identificar los componentes en las imágenes.

Todos estos elementos se han utilizado exclusivamente con fines educativos y sin ánimo de lucro, respetando los términos de uso de cada plataforma. Las imágenes se han generado específicamente para ilustrar la arquitectura propuesta y no se emplean con fines comerciales ni de redistribución.

Anexo A

Configuración del punto de acceso

```
interface=wlxf4f26d0c9daa
driver=nl80211
ssid=Zeek-APZ
hw_mode=g
channel=6
auth_algs=1
wpa=2
wpa_passphrase=password
wpa_key_mgmt=WPA-PSK
rsn_pairwise=CCMP
ctrl_interface=/var/run/hostapd
```

Anexo B

Fragmento de configuración de Open vSwitch

El siguiente código muestra la configuración mínima de *Open vSwitch* (OVS) empleada para conectar las interfaces y duplicar el tráfico hacia la interfaz `gateway1`, permitiendo su análisis por parte de `Zeek`:

```
# Crear puente virtual y añadir interfaces
ovs-vsctl del-br br0
ovs-vsctl add-br br0
ovs-vsctl add-port br0 enp0s3
ovs-vsctl add-port br0 wlxf4f26d0c9daa
ovs-vsctl add-port br0 gateway1 -- set interface gateway1 type=internal
ovs-vsctl add-port br0 gateway2 -- set interface gateway2 type=internal

# Reglas de flujo: duplicación y reenvío
ovs-ofctl add-flow br0 in_port=1,actions=output:3
ovs-ofctl add-flow br0 in_port=2,actions=output:4
ovs-ofctl add-flow br0 in_port=3,actions=output:1
ovs-ofctl add-flow br0 in_port=4,actions=output:2
```

Anexo C

Configuración del servidor DHCP

A continuación se muestra el contenido del fichero `/etc/dhcp/dhcpd.conf`, utilizado en el sistema como configuración base para la asignación dinámica de direcciones IP:

```
option domain-name "srv.world";
option domain-name-servers 8.8.8.8, 8.8.4.4;

default-lease-time 600;
max-lease-time 7200;

ddns-update-style none;
authoritative;

subnet 192.168.100.0 netmask 255.255.255.0 {
  option routers 192.168.100.1;
  option subnet-mask 255.255.255.0;
  range dynamic-bootp 192.168.100.200 192.168.100.254;
}
```

Anexo D

Configuración de la red

A continuación se muestra el contenido del fichero `/etc/network/interfaces`

```
auto enp0s3
iface enp0s3 inet manual

auto wlxf4f26d0c9daa

auto gateway1
iface gateway1 inet static
address 192.168.100.1
netmask 255.255.255.0

auto gateway2
iface gateway2 inet dhcp
```