UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Sniffer IV

Presentado por Raúl Merinero Sanz
en Universidad de Burgos — 22 de septiembre de 2022
Tutor: José Manuel Sáiz Diez

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. José Manuel Sáiz Diez, profesor del departamento de Ingeniería Informática, área de Lenguajes y Sistemas Informáticos.

Expone:

Que el alumno D. Raúl Merinero Sanz, con DNI 71483362Y, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado Sniffer IV.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 22 de septiembre de 2022

Firmado por SAIZ DIEZ JOSE MANUEL - 14590420W el día 23/09/2022 con un certificado emitido por AC FNMT Usuarios

Vº. Bº. del Tutor:

D. José Manuel Sáiz Diez

Resumen

Hoy en día, las redes de comunicaciones son un aspecto muy importante de nuestras vidas. Una buena seguridad respecto a ellas es un punto poco valorado socialmente. Es realmente necesario tener herramientas para la gestión de estas redes, y ahí es donde pueden ayudar los *Sniffers*. Estos, se usan para comprobar el correcto funcionamiento de una aplicación que use cierto protocolo, al poder ver el contenido de sus paquetes, así como para analizar ataques masivos a las máquinas, entre otros usos.

La aplicación desarrollada en este proyecto es un *Sniffer*, una herramienta para la gestión de redes, que permite capturar y enviar paquetes con determinados protocolos, así como definir nuevos protocolos creados por el usuario, entre otras funcionalidades.

Se han realizado ciertas investigaciones sobre las bibliotecas en las que se basa el proyecto, que han resultado en una mejora de la biblioteca usada actualmente, con la inclusión de nuevos protocolos. Aún así, se considera una buena opción de futuro el cambio de bibliotecas, llegando a implementar la interfaz nativa de Java, JNI, que permitirá la utilización de *libPcap* y *nPcap*, siendo esta última una mejora de la obsoleta *winPcap*.

En cuanto a la aplicación, propiamente dicha, se han implementado varias modificaciones para mejorar la interfaz gráfica, ya que se permiten mostrar datos que antes no y se ha aumentado la usabilidad de cara al usuario. También, se ha modificado alguna clase concreta para aumentar la velocidad en la realización de capturas. Otra modificación realizada es la actualización de las diversas bibliotecas que utiliza la aplicación.

Descriptores

Redes, comunicaciones, sniffer, protocolos de red, captura de tráfico, internet, paquetes, windows, linux, java, máquinas virtuales

Abstract

Nowadays, communication networks are a very important aspect of our lives. Having a good security regarding them is a point that is little valued socially. It is really necessary to have tools for the management of this networks, and that is exactly where *Sniffers* can help. These are used to check the correct operation of an application that uses a certain protocol, by being able to see the content of its packets, as well as to analyze massive attacks on machines, among other uses.

The application developed in this project is a *Sniffer*, a network management tool that allows capturing and sending packets with certain protocols, as well as defining new packets created by the user, among other functionalities.

Some research has been done on the libraries on which the project is based, which has resulted in an improvement of the currently used library, with the inclusion of new protocols. Even so, changing libraries is considered a good option for the future, eventually implementing the Java Native Interface, JNI, which will allow the use of *libPcap* and *nPcap*, the latter being an improvement on the obsolete *winPcap*.

Regarding the application itself, several modifications have been implemented to improve the graphical interface, since it allows data to be displayed that was not previously possible, and usability has been ensured for the user. Also, some specific classes have been modified to increase the speed in making captures. Another modification made is updating the various libraries used by the application.

Keywords

Networks, comunications, sniffer, network protocols, traffic capture, internet, packages, windows, linux, java, virtual machines

Índice general

Índice de figuras

Índice de tablas

Introducción

En este apartadado se explica en qué consiste el proyecto, la motivación que llevó al alumno a escogerlo y la forma en que la memoria está estructurada.

1.1. Descripción general

Este proyecto ha consistido en la mejora de la aplicación *Sniffer*. Esta es una herramienta de software que permite al usuario visualizar el tráfico de la red en tiempo real, así como capturar los paquetes que entran y salen de su equipo. Otras funcionalidades son la definición de protocolos propios y la inserción de paquetes en la red.

Esta aplicación ha sido creada y modificada en diferentes Trabajos de Fin de Grado anteriores. Los autores son: Leonardo García y Ramón Gutiérrez (*Sniffer I*), Carlos Mardones Muga (*Sniffer II*) y Rodrigo Sánchez González (*Sniffer III*).

Además, está relacionada con otros proyectos, como son *Localización de intrusos* e *IDS*, con los autores: Nuria González Mata (*Localización de Intrusos II*), Beatriz Gómez Merino (*Localización de Intrusos III*), Álvaro García Gutiérrez (*Localización de Intrusos IV*), Jesús Ángelo González del Pino y Sergio Samaniego Angulo (*Sniffer para creación de un conjunto de datos II o IDS I*), Francisco José Güemes Sevilla y Roberto Miñón García (*IDS II*), y Ana María Castro Merino (*IDS III*).

1.2. Motivación

A lo largo de los cursos que forman este grado, hemos trabajado con diversas aplicaciones del ámbito de las redes. De todas ellas, una de las más destacables es Wireshark, el analizador de protocolos más utilizado del mundo.

Así, cuando se me ofreció la posibilidad de trabajar sobre el *Sniffer*, que comparte alguna funcionalidad con esta, no me quedaron dudas y me decanté por ella.

Además, mi tutor me comentó que pensaba utilizarla en alguna asignatura, y la idea de poder interactuar con una aplicación que pueda usarse en docencia es muy llamativa, ya que puede ampliar los conocimientos de los alumnos, al mostrarles de forma práctica algunos de los conceptos que estudian.

1.3. Estructura de la memoria

Este documento comienza dejando claros los objetivos definidos en un primer momento, y explicando algunos conceptos teóricos que puedan generar problemas de comprensión, sobre todo en aquellas personas ajenas a este ámbito de la informática. Posteriormente, se habla de las herramientas utilizadas y la metodología seguida durante el desarrollo de la aplicación. Para finalizar, se destacan algunos aspectos que se han considerado relevantes, se mencionan versiones anteriores y trabajos relacionados, y se realizan una serie de conclusiones, acompañadas de unas posibles líneas de trabajo futuras.

En cuanto a los anexos, se ha realizado un *Plan de Proyecto Software*, con una planificación temporal y un estudio de viabilidad. También se han especificado concretamente los diferentes requisitos de la aplicación, así como el diseño de los datos y la forma en que está estructurada, en los apéndices *Especificación de Requisitos* y *Especificación de diseño*, respectivamente. En cuanto a la documentación técnica, se ha recogido la estructura de directorios y paquetes y las diversas pruebas realizadas sobre la aplicación, en la *Documentación técnica de programación.* Como último apéndice, tenemos la *Documentación de usuario*, donde detallamos los requisitos de los usuarios, los pasos que estos deben seguir en el proceso de instalación, y el manual de usuario, que los ayudará a conocer al detalle el funcionamiento de la aplicación.

1.4. Materiales adjuntos

Los materiales que aparecen adjuntos con la documentación son:

- Entorno virtual para el despliegue directo de la aplicación, sin necesidad de realizar todo el proceso de instalación. Este también contiene el entorno de programación
- Directorio con las herramientas de instalación. Su uso está explicado en el cuarto apartado del apéndice *Documentación técnica de programación.*
- Directorio con la biblioteca *jNetPcap* completa, siendo la última versión oficial.
- Directorio del proyecto de Eclipse, sobre el que se ha desarrollado la aplicación.
- Directorio del proyecto de de la biblioteca modificada, sobre el que se ha construido la biblioteca.

El repositorio de Github se encuentra en [7].

Objetivos del proyecto

Este apartado explica de forma precisa y concisa cuáles son los objetivos que se persiguen con la realización del proyecto. Se distingue entre el objetivo general y unos requisitos más específicos, que explican más detalladamente lo que se ha intentado lograr, comentando también algunos objetivos técnicos.

2.1. Objetivo general

El principal objetivo de este proyecto es mejorar la versión anterior de la aplicación, sobre todo, priorizando mejorar la usabilidad y ampliar sus posibilidades.

2.2. Objetivos específicos

Los principales objetivos del proyecto son los siguientes:

1. Investigar sobre las bibliotecas actuales y otras posibilidades.
2. Actualizar el código obsoleto y las versiones de las diferentes bibliotecas utilizadas.
3. Probar el correcto funcionamiento de la aplicación en Windows y Linux, con ambas arquitecturas, x64 y x32, montando las correspondientes máquinas virtuales sobre las que realizar las pruebas.
4. Mejorar la interfaz de la aplicación, incluyendo la posibilidad de ordenar la tabla de paquetes por los diferentes campos, modificando la visualización del detallado de paquetes para poder escoger cuántos

paquetes ver a la vez, y en qué distribución, y permitiendo visualizar los primeros bytes, o todos, de cada paquete, tanto en formato haxadecimal como en decimal.

5. Parametrizar los diferentes valores de preferencias del usuario relativos al detallado de paquetes.

6. Mejorar el protocolo HTTP. Este se encuentra implementado en la última versión de la biblioteca *jNetPcap*, pero no está incluido en la aplicación. Se pretende crear una clase analizadora en la aplicación, con el objetivo de que se muestre la información relacionada con este protocolo en el panel del detallado de paquetes.

7. Parametrizar el guardado en XML al realizar una captura, permitiendo grabar únicamente un fichero con formato ".pcap". El proceso de exportación a XML es excesivamente lento y no siempre puede ser requerido por el usuario.

8. Calcular la longitud de las cabeceras de los paquetes. Este campo está mostrado en la tabla de paquetes de la aplicación, pero no muestra ningún valor. Lo que se pretende hacer es modificar la aplicación para que este campo muestre el valor real de la longitud de la cabecera de cada paquete de la tabla.

9. Mostrar los paquetes capturados en el panel de la tabla de paquetes durante la captura, y no solo al finalizar la captura.

10. Mejorar la biblioteca JnetPcap para la implementación de 3 nuevos protocolos: DNS, ICMPv6 e IGMP.

2.3. Objetivos técnicos

Complementando los objetivos anteriores, tenemos también fijados una serie de objetivos técnicos, que se resumen en:

1. Mantener la compatibilidad de la aplicación con los sistemas operativos Windows y Linux, para las arquitecturas x64 y x32.

2. Conservar las funcionalidades previas al proyecto.

3. No aumentar la cantidad de recursos que acapara la aplicación al ser ejecutada, e incluso, reducirla, corrigiendo bucles de código previos.

Conceptos teóricos

En este apartado se explican una serie de conceptos para una correcta comprensión del trabajo realizado.

3.1. Sniffer [2]

La palabra *Sniffer* se refiere a una aplicación que permite capturar el tráfico de una red. Tiene posibles usos muy diferentes. Algunos son maliciosos, como interceptar correos electrónicos o robar contraseñas. Por otro lado, puede usarse de manera eficiente para comprobar el correcto funcionamiento de una aplicación que use cierto protocolo, al poder ver el contenido de sus paquetes, así como para analizar ataques masivos a las máquinas, entre otros usos.

Entre los más conocidos está Wireshark, de licencia libre. Un ejemplo de una captura de Wireshark es la siguiente imagen:

Como se puede observar, encontramos 3 partes diferenciadas en la pantalla. La primera, en el apartado superior, se trata de una tabla de paquetes, donde aparece recogida información general de cada uno de los paquetes, como las direcciones IP de origen y destino, o el protocolo que define al paquete. El siguiente recuadro, ubicado en el centro de la imagen, muestra los protocolos que forman el paquete, cuya información puede ser expandida. Por último, en la parte inferior, encontramos la visualización de los bytes del paquete, pudiendo ver cuáles corresponden a cada protocolo.

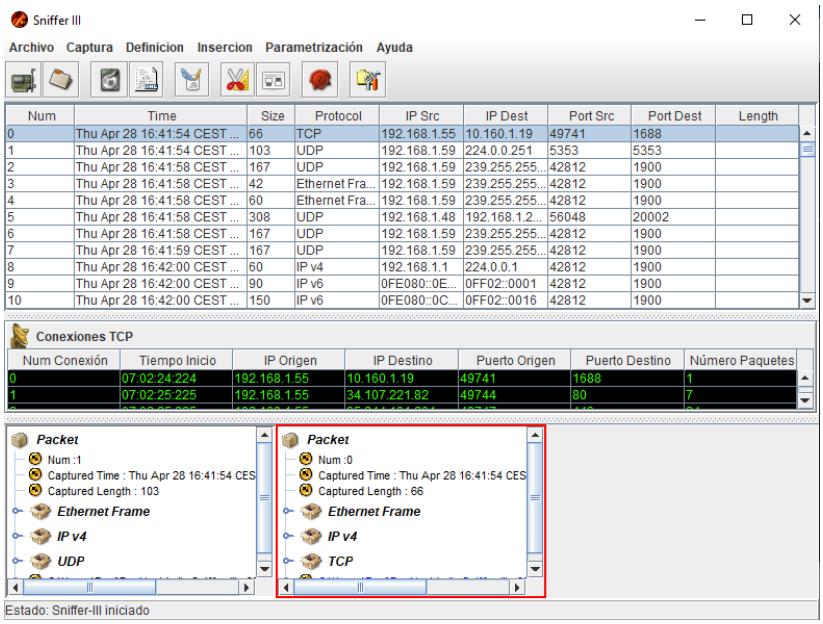

Figura 3.1: Ejemplo de una captura realizada en el Sniffer.

3.2. WinPcap [20]

WinPcap es un driver y una biblioteca en lenguaje C/C++ que extiende al sistema operativo y le permite tener acceso a las capas de la red de nivel bajo.

Además, contiene la versión para Windows de la API LibPcap de Unix.

Aún así, actualmente se encuentra obsoleta, por lo que ha aparecido otra biblioteca como una mejora de *winPcap*, y es *nPcap*.

La aplicación desarrollada no funciona en Windows si no está esta biblioteca instalada, dado que la biblioteca *jNetPcap* hace uso de ella.

3.3. LibPcap [6]

LibPcap es una biblioteca en lenguaje C/C++ de código abierto para capturar paquetes de red. Aún así, dispone de más funcionalidades, como crear y manipular paquetes a partir de archivos guardados.

La aplicación desarrollada no funciona en Linux si no están esta biblioteca y sus complementos debidamente instalados, dado que la biblioteca *jNetPcap* hace uso de *libPcap*, que a su vez necesita esos complementos.

3.4. JNI [23]

JNI, o *Java Native Interface*, es un framework de programación que permite ejecutar programas escritos en lenguaje Java, que interactúan con programas escritos en lenguaje C o C++, mediante la máquina virtual Java.

3.5. JnetPcap [19]

JnetPcap es una biblioteca de Java de código abierto que funciona como un contenedor Java para las bibliotecas *winPcap* y *libPcap*, que permite utilizar ambas bibliotecas, escritas en lenguaje C/C++, en lenguaje Java.

Es también la biblioteca sobre la que se basa la aplicación desarrollada en este proyecto, el *Sniffer*.

3.6. nPcap [14]

nPcap es una biblioteca en lenguaje C/C++ para la captura y envío de paquetes en Microsoft Windows. Apareció como una forma de actualizar la antigua biblioteca *winPcap*, cuyo proyecto se abandonó en 2013. De esta forma, *nPcap* brinda unas mayores seguridad y velocidad, además de la compatibilidad con Windows 10. De la misma forma que *winPcap*, contiene una versión de *libPcap*, sobre la que se basa para adaptarla a Windows, pero actualizada. También tiene características más avanzadas como la captura e inyección de tráfico *Loopback*.

La siguiente tabla muestra una comparativa de las características básicas de *nPcap* y *winPcap*, encontrada en [15]:

Info	Npcap	WinPcap
Actively maintained and supported	**Yes**	**No** (Note)
Last release date	August 19, 2022	March 8, 2013
libpcap version	1.10.1 (2021)	1.0.0 (2008)
License	Free for personal use	BSD-style
Supported commercial/redistributable version	**Yes:** Npcap OEM	**No** (Note)

Figura 3.2: Comparativa de nPcap y winPcap.

3.7. Rfc [9]

Un RFC (Request For Comments") es un documento que propone y define estándares correspondientes a nuevas arquitecturas, protocolos y servicios relacionados con comunicaciones. La gestión de estos es llevada a cabo por el IETF (Ïnternet Engineering Task Force"), el consorcio de colaboración técnica más importante de Internet.

3.8. Protocolos de red [25]

Los protocolos de comunicaciones son conjuntos de reglas y formatos, que permiten que diferentes entidades de un sistema de comunicación establezcan una conexión entre sí para transmitir información.

DNS [12]

DNS, o Sistema de Nombres de Dominio (*Domain Name System*), es un protocolo de nivel de aplicación que sirve para traducir los nombres de dominio a direcciones IP, con el fin de que los usuarios no necesiten conocer las direcciones IP, sino solo los nombres "vulgares" de los servicios o máquinas donde se encuentren los recursos.

La estructura de este protocolo se divide en 5 secciones:

- Header
- Question/s
- Answer/s
- Authority/ies
- Additional/s

La parte estática de la cabecera del protocolo se define por los siguientes campos:

El primer campo es un identificador de 16 bits, y se genera en cualquier tipo de consulta que se realice. Sirve para hacer coincidir las respuestas a las consultas pendientes, por parte del solicitante.

Los siguientes campos son *flags*, o banderas, que representan un valor correspondiente a una serie de opciones, en ocasiones, binarias. Por ejemplo,

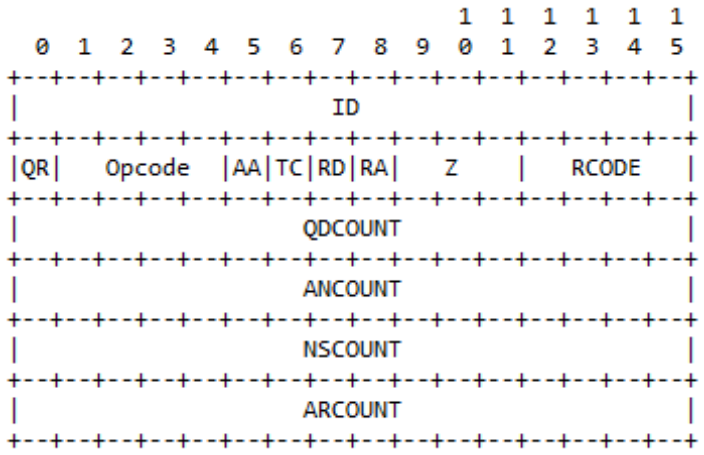

Figura 3.3: Campos estáticos del protocolo DNS.

el primero de los campos, denominado *QR*, determina si el mensaje es una consulta o una respuesta, en función del valor del bit, 0 ó 1, respectivamente.

En cuanto a los siguientes campos, *QDCOUNT* representa el número de entradas que se encuentran en la sección de las consultas; *ANCOUNT*, el número de entradas en la sección de respuestas; *NSCOUNT*, el número de registros de recursos del servidor de nombres en la sección de registros de autoridad, y *ARCOUNT*, el número de registros de recursos en la sección de registros de autoridad.

ICMPv6 [1]

ICMPv6 o Protocolo de Mensajes de Control de Internet Version 6 (*Internet Control Message Protocol Version 6*), es una nueva versión de ICMP para el nuevo direccionamiento IPv6. Este protocolo combina funciones que anteriormente se subdividían entre varios protocolos, como ICMP, IGMP y ARP. También, elimina ciertos aspectos obsoletos de estos protocolos.

El formato de los paquetes depende del tipo de mensaje que se envíe, pero el aspecto genérico es el siguiente:

El primer campo establece el tipo del mensaje que se envía, que define la estructura del cuerpo del mensaje. Existen dos grandes grupos de mensajes, identificados por estos tipos: los mensajes de error y los mensajes informativos.

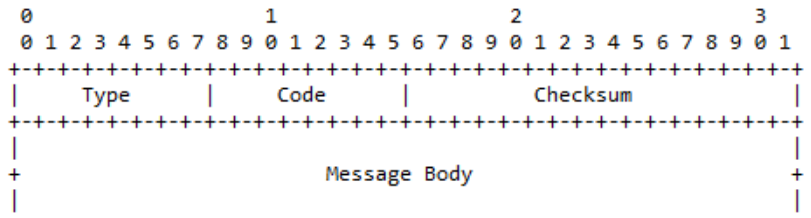

Figura 3.4: Formato general del protocolo ICMPv6.

El campo del código depende del valor que se encuentre en el campo del tipo, y sirve como subtipo de este, con el objetivo de añadir información adicional del objetivo del mensaje.

Por último, el *checksum* o suma de comprobación, informa de si se han producido errores en el envío del paquete. Es calculado a partir de tanto la cabecera como los datos.

IGMP [3]

IGMP o Protocolo de Gestión de Grupos de Internet (*Internet Group Management Protocol*), es un protocolo que se utiliza para informar a los enrutadores de multidifusión vecinos de sus pertenencias a grupos de multidifusión de IP.

Este protocolo tampoco tiene una estructura fija, ya que además existen 3 versiones de él. Analizando una de las posibles estructuras, un mensaje de consulta de pertenencia a un grupo o *Query*, dispone del siguiente formato:

El campo del tipo define la estructura del protocolo y el objetivo del mensaje. Existen 5 tipos:

- **0x11**: consulta de pertenencia al grupo.
- **0x12**: informe de pertenencia al grupo, de la versión 1.
- **0x16**: informe de pertenencia al grupo, de la versión 2.
- **0x17**: dejar el grupo.
- **0x22**: informe de pertenencia al grupo, de la versión 3.

El siguiente campo, el del código de respuesta máxima, establece el tiempo máximo permitido antes de enviar un informe de respuesta.

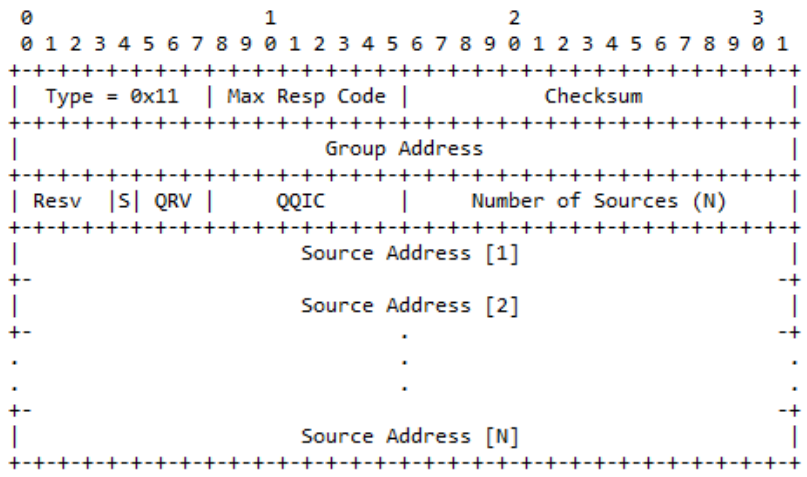

Figura 3.5: Formato general de una *Query* del protocolo IGMP.

El *checksum* cumple con el mismo objetivo que para el protocolo ICMPv6, visto en el apartado anterior.

El *Group Adress* establece los receptores del mensaje, teniendo un valor de 0 en caso de que se envíe una consulta general.

Los siguientes campos son una serie de *flags* que brindan diversa información, mientras que el *Number of Sources* establece el número de direcciones presentes en la consulta, que define el número de octetos restantes que forman el paquete. Estos contendrán las dichas direcciones IP unidifusión.

HTTP [17]

HTTP o Protocolo de Transferencia de HiperTexto (*HyperText Transfer Protocol*), es un protocolo de nivel de aplicación, genérico y sin estado, que puede ser utilizado para muchas tareas más allá de su uso para la transferencia de archivos en la *World Wide Web*.

Fue desarrollado en una colaboración realizada entre el *World Wide Web Consortium* y la *Internet Engineering Task Force*, saliendo a la luz su primera versión en forma de RFC en 1999.

El protocolo HTTP no está estructurado de forma similar a los anteriores, sino que su estructura es la siguiente:

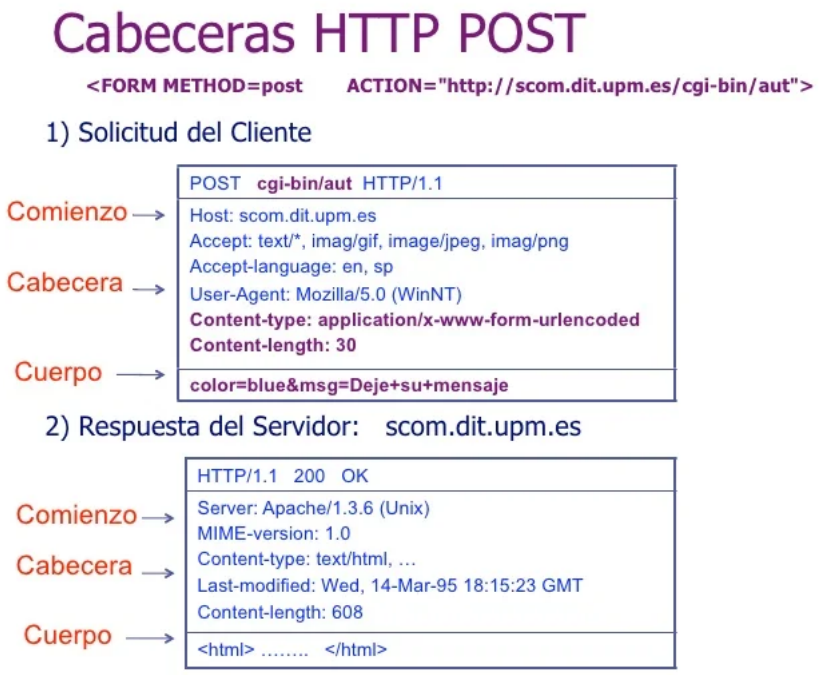

Figura 3.6: Formato del protocolo HTTP.

Como un ejemplo de cómo se realiza un diálogo HTTP, tenemos el siguiente, obtenido de [21]:

Para obtener un recurso de la URL http://www.example.com/index.html, se siguen estos pasos:

1. Se abre una conexión en el puerto 80 del host (www.example.com). El puerto 80 es el puerto predefinido para HTTP. Si se quisiera utilizar otro puerto, debería ser especificado en la propia dirección del recurso, de la forma: http://www.example.com:XXXX/index.html, siendo XXXX el identificador del puerto.

2. Se envía un mensaje en el estilo siguiente:

```
GET /index.html HTTP/1.1
Host: www.example.com
Referer: www.google.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0
Connection: keep-alive
[Línea en blanco]
```

Figura 3.7: Petición de recurso con HTTP.

3. La respuesta del servidor está formada por encabezados seguidos del recurso solicitado, en el caso de una página web:

```
HTTP/1.1 200 OK
Date: Fri, 31 Dec 2003 23:59:59 GMT
Content-Type: text/html
Content-Length: 1221

<html lang="eo">
<head>
<meta charset="utf-8">
<title>Título del sitio</title>
</head>
<body>
<h1>Página principal de tuHost</h1>
(Contenido)
  .
  .
  .
</body>
</html>
```

Figura 3.8: Respuesta a la petición con HTTP.

Técnicas y herramientas

Esta parte de la memoria tiene como objetivo presentar las técnicas metodológicas y las herramientas de desarrollo que se han utilizado para llevar a cabo el proyecto.

4.1. Herramientas

A continuación, se explican las diferentes herramientas utilizadas para el desarrollo del proyecto, junto con las versiones concretas utilizadas.

Primero, recogemos un resumen de las diferentes herramientas y sus versiones:

Herramienta	**Versión**
VirtualBox	6.1.34
Windows	Windows 10 Home
Ubuntu	22.04 LTS (x64) y 17.04 (x32)
Eclipse	2022-06
gEdit	por defecto de Ubuntu
Java	1.8.0_321
Ant	plug-in de Eclipse

Tabla 4.1: Versiones anteriores del Sniffer.

Virtualbox [11]

Version: 6.1.34

Oracle VM VirtualBox es el software de virtualización multiplataforma de código abierto más popular del mundo. Este permite ejecutar múltiples sistemas operativos en un solo equipo.

Figura 4.1: Icono de la aplicación VirtualBox.

Windows [24]

Version: Windows 10 Home (x64 y x32)

Windows es una familia de distribuciones de software para diversos dispositivos, creado por la empresa Microsoft. Como tal, no es un sistema operativo, sino que contiene uno, siendo este MS-DOS o Windows NT. Este sistema operativo es presentado con una interfaz gráfica y una gran cantidad de software de la misma empresa.

La versión actual más estable es Windows 10, aunque se ha comenzado con la transición a Windows 11.

Figura 4.2: Icono de Windows 10.

Ubuntu [26]

Version: 22.04 LTS (x64) y 17.04 (x32)

Ubuntu es una distribución de software compuesta por un núcleo Linux y diversas herramientas del proyecto GNU y el sistema de ventanas X Window System. Está basada, concretamente, en Debian GNU/Linux, de software libre y código abierto.

Figura 4.3: Icono de Ubuntu (2022).

Eclipse [5]

Version: 2022-06

Eclipse es una aplicación orientada a la programación, que dispone de herramientas de código abierto multiplataforma. Usualmente, es usada para programar en los lenguajes Java y C.

Figura 4.4: Icono de la aplicación Eclipse.

gEdit [22]

Version: por defecto de Ubuntu

gEdit es un editor de textos compatible con UTF-8 y, por tanto, con muchas plataformas. Aún así, es utilizado mayoritariamente en Linux. Incluye herramientas para la edición de código fuente y textos estructurados, como lenguaje de marcado.

Figura 4.5: Icono de la aplicación gEdit.

Lenguaje Java [10]

Version: 1.8.0.321

Java es uno de los lenguajes de programación más utilizados. Una amplia variedad de sitios web lo necesitan para su correcto funcionamiento. Es utilizado en todas las plataformas.

Figura 4.6: Icono de Java.

Ant [16]

Ant es una herramienta que aparece en Eclipse como plug-in. Esta permite construir los proyectos de una forma más eficiente mediante *scripts*, realizando tareas como la compilación de todos los ficheros de código.

Figura 4.7: Icono de Apache Ant.

4.2. Metodología

Este apartado puede no ser muy concreto, dados los objetivos planificados y los problemas encontrados a la hora de intentar alcanzarlos. Además, la planificación temporal no ha sido óptima, por lo que ha sido difícil seguir una metodología ágil.

En un primer momento de planteó la posibilidad de seguir la metodología SCRUM, dado que varias de sus características encajaban con las de este proyecto.

SCRUM [18] es una metodología ágil basada en un conjunto de buenas prácticas para trabajar de forma colaborativa, buscando optimizar la versión final del producto. Se establecen unos periodos fijos y cortos denominados *Sprints*, en los que se intentan alcanzar unos objetivos establecidos al comienzo de este. Cuando cada uno acaba, se realiza una reunión del equipo con el *Scrum Master* (puede ser un miembro del equipo), se analizan los objetivos alcanzados y se establecen otros para el *Sprint* siguiente. Además, se realizan reuniones diarias muy cortas (de unos 15 minutos), llamadas *Daily Scrum*, con las que se pretende que todos los miembros del equipo de desarrollo conozcan lo que han hecho y lo que les queda por hacer a los demás miembros.

De esta forma, por ejemplo, el rol del *Scrum Master* estaba marcado, siendo este el tutor, y se procuraban realizar reuniones semanales, pudiendo haber dejados establecidos una serie de *Sprints*. Obviamente, siendo un trabajo individual, no hay un equipo de desarrolladores, pero podemos entender que el equipo es el propio alumno. En cuanto al *Daily Scrum*, este no se realizaba todos los días, únicamente se hablaba por correo en caso de que hubiese algún inconveniente en los objetivos marcados.

Aún así, al no ser un proyecto de programación por completo, si no que se dedica mucho tiempo en investigación sobre temas inciertos y en realizar pruebas para comprobar las mejoras que se pueden implementar, no es fácil ajustar el proyecto a una de estas metodologías.

Aspectos relevantes del desarrollo del proyecto

Este apartado recoge los aspectos más interesantes que han surgido y se han considerado relevantes durante el desarrollo del proyecto.

5.1. Conocimiento de la aplicación y antiguos proyectos

El *Sniffer* es una aplicación que lleva mucho tiempo en desarrollo. La complejidad de esta aplicación ha ido aumentando con cada uno de los proyectos, y aunque los objetivos propuestos han orientado el proyecto hacia una rama concreta, de entre la gran cantidad de posibilidades que había, era necesario conocer todas las funcionalidades de la aplicación. Esto es sobre todo porque las pruebas de funcionalidad contemplan toda la aplicación, y no solo aquellos aspectos que se han usado. Aunque también hay que tener en cuenta que las modificaciones de la aplicación podían ser propuestas por el alumno al tutor, valorando éste su valor futuro y decidiendo si implementarlas.

La aplicación presenta un problema en algunas ocasiones, pues a veces no realiza las funcionalidades que se le piden, y se requiere un reinicio de la aplicación. En concreto, al intentar abrir un fichero de capturas, la interfaz gráfica parece que se actualiza, pero no muestra nada más que la pantalla principal. Esto viene dado por una cantidad insuficiente de memoria virtual de la máquina virtual Java, aspecto que se puede modificar al inicio de la aplicación.

5.2. Investigación de bibliotecas

Como fase propia de análisis e investigación tenemos la búsqueda de información relativa a las bibliotecas que utiliza la aplicación. Rápidamente se vio que la biblioteca actual (*jNetPcap*) lleva años sin ser actualizada, y que una de las bibliotecas que ella utiliza lo está también (*winPcap*).

Con ello, se buscaron alternativas a *jNetPcap*, permitiendo así poder seguir programando en lenguaje Java, sin hacer uso de JNI, pero no se encontraron.

No pasó lo mismo para las alternativas a *winPcap*, ya que nos encontramos con *nPcap*, surgida como una actualización de esta biblioteca escrita en lenguaje C.

Como resultado, vimos que solo había dos posibilidades: seguir con la biblioteca actual y modificarla, o sustituir algunas clases y métodos para adaptar la aplicación a JNI.

5.3. Métodos obsoletos

La actualización, sustitución o adaptación de código con métodos obsoletos es un proceso delicado. Por desgracia, o por querer realizarlo de forma rápida, esto se descubrió de la peor forma posible.

Al empezar con este paso, presente en los objetivos del proyecto, se buscaron los métodos con los que se podían sustituir aquellos que estaban obsoletos. Así, se modificó el código hasta que no hubiese ninguno en ese estado, lo que generó fallos en la aplicación.

Esta meta tan sencilla, relativamente, ocupó más tiempo del debido, ya que posteriormente se tuvo que retroceder al código previo e implementar los cambios uno por uno probando la aplicación. De esta forma, se consiguió encontrar aquellos métodos que estaban dando problemas y estos se pudieron solucionar. Algunos de ellos son los siguientes:

- `java.lang.Thread.stop()`

  se sustituye por

  `java.lang.Thread.interrupt()`

- `new Integer(int)`

  se sustituye por

  `Integer.valueOf(int)`

- `new Long(int)`

  se sustituye por

  `Long.valueOf(int)`

- `new Boolean(boolean)`

  se sustituye por

  `Boolean.valueOf(boolean)`

- `java.awt.Window.show(boolean)`

  se sustituye por

  `java.awt.Window.setVisible(boolean)`

- `java.awt.Window.resize(int width, int height)`

  se sustituye por

  `java.awt.Window.setSize(int width, int height)`

5.4. Implementación de mejoras propias de la aplicación

Inicialmente, se optó por realizar una serie de mejoras para mejorar la presentación de los datos y agilizar el proceso de las capturas. Esto resultó ser moderadamente complicado, ya que, si bien se conocían las funcionalidades de la aplicación y la estructuración de los paquetes, no se conocían exactamente las clases y sus correspondientes métodos. Cabe destacar la eliminación de un bucle de código en la captura de paquetes, que hacía escribir un mismo fichero reiteradas veces en una misma ejecución. Además, estas capturas se guardaban tanto en formato ".pcap", como en formato XML. Este aspecto se ha parametrizado para permitir la opción de no guardar en XML, ya que la exportación a este formato consume una elevada cantidad de recursos.

5.5. Protocolo HTTP [17]

El protocolo HTTP es uno de los más comunes en las conexiones a Internet en algún navegador, y de hecho, es el utilizado para inicial el establecimiento de una comunicación con un Servicio WEB. Investigando la biblioteca utilizada (*jNetPcap*), se observó que este estaba implementado y podía ser usado en nuestra aplicación. Aún así, en nuestra aplicación no se habían definido sus campos, por lo que no se mostraba información relativa a este protocolo.

Así, se decide implementar en la aplicación como primera toma de contacto con los analizadores de protocolos, es decir, las clases encargadas de tomar aquella información de los protocolos que nosotros creamos conveniente.

Además, es un protocolo ampliamente usado y permite aumentar las posibilidades de nuestro *Sniffer*.

En cuanto a la implementación realizada, este protocolo estaba definido en la biblioteca *jNetPcap*, pero esto no es suficiente para que nuestra aplicación lo interprete. Para ello, se necesita una clase que actúe de intermediaria entre la biblioteca y la aplicación. Esta clase, llamada analizador, no contenía campos relevantes del protocolo HTTP, por lo que se han implementado algunos de los más importantes, como la versión del protocolo.

5.6. JNI vs jNetPcap

Una vez realizados todos los cambios que se vieron necesarios en la aplicación, se optó por la inclusión de nuevos protocolos con *jNetPcap*, o el paso a las bibliotecas escritas en C mediante JNI.

Se quiso intentar primero con la segunda opción. Se buscó información en un proyecto anterior que tuvo el propósito de pasar también a JNI, así como en Internet, pero tras un tiempo tratando de configurar correctamente el entorno de trabajo y probando unos programas sencillos, se consideró que la inversión de tiempo para esta opción iba a ser excesiva. Un ejemplo de un artículo seguido para comprender su uso es [4], con fecha de 2020, ya que no existe mucha información relacionada con esta herramienta.

En ese momento, se comenzó con un análisis profundo de la biblioteca *jNetPcap*, su estructura y la manera en que está programada.

Una vez comprendida, relativamente, la biblioteca, se realizó la implementación de los 3 nuevos protocolos (DNS, ICMPv6 e IGMP), incluyendo

las nuevas clases en el paquete *org.jnetpcap.protocol*, en concreto, DNS en *application*, y los protocolos ICMPv6 e IGMP en *network*.

5.7. DLL [8]

Uno de los archivos necesarios para el funcionamiento de la aplicación en un equipo con sistema operativo Windows es una DLL, o biblioteca de enlaces dinámicos, una biblioteca que permite ser usada por varios programas a la vez.

De esta forma, se pensó el antiguo DLL de *jNetPcap* debía ser reconstruido, tras las mejoras realizadas en la biblioteca. Tras mucha investigación, se realizaron varios intentos fallidos en Microsoft Visual Studio, utilizando el comando *javah* para obtener los ficheros de cabecera ".h" a partir de los ficheros compilados de Java, ".class", ya que las DLL se construyen con ficheros ".h" y ".c".

Más adelante, se comprendió que aquello no era lo que se necesitaba para actualizar la biblioteca. El *DLL* contiene los archivos escritos en C que se encuentran en el directorio /jNetPcap/src/c/ y, como no habían sido modificados, no se necesitaba reconstruir el DLL.

Finalmente, se actualizó el JAR que contiene los archivos de la biblioteca, encontrado en el directorio /lib/ del proyecto de la aplicación. De esta forma, se pudo construir los analizadores de los protocolos correctamente.

5.8. Nuevos protocolos en *jNetPcap*

Como la gran mejora de la aplicación a nivel de *backend*, tenemos la inclusión de 3 nuevos protocolos a la biblioteca utilizada. Estos han sido codificados siguiendo el estilo de programación de aquellos ya implementados.

Aún así, ha surgido un problema a la hora de que estos funcionen, ya que si bien el *Sniffer* los reconoce, hay un problema que impide que estos aparezcan en el detallado de paquetes, es decir, no se muestran en nuestra aplicación.

Hay una función en la biblioteca que identifica las cabeceras de los protocolos, y carga esa parte del paquete en una variable, de la cual obtenemos los datos correspondientes a la información del protocolo. Esto es exactamente lo que no termina de funcionar, ya que se ha investigado profundamente dicho método y no se encuentra una solución.

Cabe destacar, eso sí, que el protocolo ICMP, ya implementado en la biblioteca y precursor del ICMPv6, tampoco funciona para nuestra aplicación. Por ello, se cree que el fallo no es del alumno, si no de las posibilidades de la biblioteca, o de alguna clase o método no encontrados que deban ser modificados para reconocer las cabeceras de los nuevos protocolos.

Trabajos relacionados

En este apartado se van a comentar los trabajos que están relacionados con la aplicación desarrollada.

6.1. Wireshark [27]

WireShark es un potente analizador libre de protocolos de redes para máquinas con sistemas operativos Unix y Windows. Este permite capturar los datos directamente de una red o bien obtener información a partir de una captura en disco, pudiendo leer más de veinte tipos de formatos distintos. Soporta más de trescientos protocolos, debido a su licencia GPL y sus más de doscientos colaboradores de todo el mundo.

Figura 6.1: Logo de la aplicación Wireshark.

Wireshark hace uso de las bibliotecas *libPcap* y *winPcap* para capturar el tráfico de la red, y soporta los dos sistemas operativos necesarios para el proyecto. Así, se convierte en la única aplicación que comparte estas características con nuestro *Sniffer*. Solo hay otro similar, *ettercap*, pero solo es válido para LANs con *switch*.

6.2. Versiones anteriores del *Sniffer*

Esta aplicación ha sido desarrollada a lo largo de varios proyectos. Por ejemplo, uno de los antecedentes es el *Sniffer-I*, desarrollado por Leonardo García y Ramón Gutiérrez, o el *Sniffer-II*, de Rodrigo Sánchez González, que disponían de muchas menos funcionalidades que la aplicación actual. También podemos encontrar el trabajo *Localización de intrusos*, de Álvaro García Gutiérrez, y el trabajo *Localización de intrusos (IDS)*, de Beatriz Gómez Merino. Se recogen en la siguiente tabla:

Trabajo	**Autor/es**
Sniffer I	Leonardo García y Ramón Gutiérrez
Sniffer II	Carlos Mardones Muga
Sniffer III	Rodrigo Sánchez González
Localización de Intrusos II	Nuria González Mata
Localización de Intrusos III	Beatriz Gómez Merino
Localización de Intrusos IV	Álvaro García Gutiérrez
IDS I	Jesús Ángelo González del Pino y Sergio Samaniego
IDS II	Francisco José Güemes Sevilla y Roberto Miñón García
IDS III	Ana María Castro Merino

Tabla 6.1: Versiones anteriores del Sniffer.

Sniffer I

Aplicación de base desarrollada por Leonardo García y Ramón Gutiérrez como TFC sobre la que se han desarrollado las mejoras hasta llegar a la aplicación que es hoy en día. La aplicación origen permite la captura de datos directamente de una red e interpretarlos. También permite la interpretación de las capturas a partir de ficheros previamente capturados.

Sniffer II

Proyecto desarrollado por Carlos Mardones Muga. Implementa las opciones de definición de paquetes, que permiten crear nuevos protocolos simples.

Sniffer III

Aplicación desarrollada por Rodrigo Sánchez González. Consigue que la aplicación funcione en entornos de 32 y 64 bits y maximizar los recursos del sistema haciedo que la aplicación sea más rápida y eficaz.

Localización de Intrusos II

Desarrollado por Nuria González Mata, haciendo uso de Sniffer III y Creación y análisis de conjuntos de datos II. Hace que los ficheros XML de Sniffer III resultantes de la captura de paquetes se generen de forma circular al igual que los ficheros de captura y elimina bugs de la aplicación.

Localización de Intrusos III

Desarrollado por Beatriz Gómez Merino como TFM. Crea nuevas mejoras para el proyecto Localización de Intrusos II para obtener una aplicación que realice la captura parametrizada en la conexión hacia la red WAN de una organización, la filtre, localice los datos relevantes y los exporte, de forma continuada.

Localización de Intrusos IV

Desarrollado por Álvaro García Gutiérrez. Implementa nuevas mejoras para el proyecto Localización de Intrusos III, habilita la opción de ejecutar la aplicación en Linux y crea un entorno virtual donde se pueda ejecutar sin problemas y sin necesidad de realizar el proceso de instalación.

IDS I

Desarrollado por Jesús Ángelo González del Pino y Sergio Samaniego Angulo. Crean una herramienta para el descubrimiento de intrusos mediante algoritmos de redes neuronales.

IDS II

Implementado por Francisco José Güemes Sevilla y Roberto Miñón García. Añaden al proyecto anterior nuevos algoritmos neuronales.

IDS III

Proyecto desarrollado por Ana María Castro Merino. Se encarga del ensamblado de la herramienta para el descubrimiento de intrusos mediante algoritmos de redes neuronales y la herramienta de tratamiento de datos XML. Tras su configuración inicial se ejecuta de forma distribuida y en modo comando y desatendido.

Conclusiones y Líneas de trabajo futuras

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.

7.1. Conclusiones

En primer lugar, hablaremos de los resultados obtenidos, teniendo de referencia los objetivos iniciales.

- La investigación de las diferentes posibilidades de utilización de bibliotecas ha llevado a descubrir que winPcap está obsoleta y libPcap no está siendo actualizada desde hace más de un año. También, ha permitido conocer *nPcap*, una mejora de la antigua *winPcap*. Esto permite una nueva línea de trabajo futura.

- El código obsoleto ha sido correctamente actualizado o sustituido, comprobando que no aparecen errores inesperados.

- El funcionamiento de la aplicación en plataformas con arquitecturas x32 y x64 es correcto tanto para Windows como para Linux.

- La tabla de paquetes permite ahora su ordenación por los diferentes campos.

- El detallado de paquetes ha sido mejorado satisfactoriamente, permitiendo escoger cuántos paquetes ver simultáneamente y en qué distribución. También, se permite la visualización de los primeros bytes, o de todos, de cada paquete. Todo esto ha sido parametrizado para mejorar la usabilidad frente al usuario.

- Se ha mejorado el analizador del protocolo HTTP de nuestra aplicación, mostrando ahora información relativa a varios campos del protocolo.

- Se ha parametrizado el guardado en XML al realizar una captura, lo que reduce mucho el tiempo y los recursos que utilizaba esta, puesto que había un pequeño bucle en el código que hacía que se escribiese varias veces el mismo fichero.

- Se ha conseguido que los paquetes capturados se muestren en la tabla de paquetes durante la captura.

- Se ha corregido la funcionalidad de mostrar la longitud de los paquetes en la tabla de paquetes, ya que no se mostraba nada.

- Se probó a introducir JNI para volver a las bibliotecas *winPcap* y *libPcap*, pero el tiempo necesario para su comprensión era demasiado para este proyecto, después de las mejoras que ya se habían realizado por aquel momento.

  Aún así, se cree que la introducción de JNI en la aplicación es la mejor opción de futuro, aunque podría requerir gran parte de un TFG, concretamente para este objetivo, ya que la aplicación es considerablemente amplia.

  Una opción alternativa puede ser JNA [13], que no requiere interactuar con el código nativo escrito en C, aunque puede resultar mucho más lenta.

- Se ha mejorado la biblioteca JnetPcap para la implementación de los protocolos DNS, ICMPv6 e IGMP, pero no se ha conseguido que funcionen, ya que no se detectan sus cabeceras. Es decir, estos protocolos están implementados al completo, con las diferentes posibles estructuras de cada uno, pero la lectura de las cabeceras por parte de un método de la biblioteca no se realiza correctamente. Ocurre lo mismo con los protocolos IPv6 e ICMP, previamente implementados

en la biblioteca original de *jNetPcap*, es decir, antes de ser modificada en este proyecto.

Como conclusión personal, tengo que destacar la oportunidad de aprendizaje que ha supuesto este proyecto. Nunca he destacado en el ámbito de las redes, pero dadas las múltiples mejoras en la usabilidad de la aplicación realizadas, gracias a las peticiones de mi tutor, ha resultado satisfactorio lograr estas mejoras de la aplicación. En cuanto a la implementación de los nuevos protocolos en la biblioteca, ha resultado ser un tema de investigación realmente interesante ver las estructuras de cada uno de ellos y comprender su funcionamiento de una forma más profunda. En cuanto a las horas empleadas, sabía de antemano que la implementación de estas mejoras requiere conocer el código y las bibliotecas utilizadas, por lo que se iba a superar el tiempo establecido para la realización del trabajo, sobre todo tras el inicio tardío, que obligaba a dedicar muchas más horas diarias.

7.2. Líneas de trabajo futuras

Tras la finalización del proyecto, se proponen algunas posibles mejoras futuras:

- Vuelta a las bibliotecas anteriores, escritas en C, mediante JNI. *JnetPcap* está obsoleta, y parece que no reconoce los últimos protocolos implementados, tanto en este proyecto, como en las últimas versiones oficiales. Como *winPcap* también se encuentra obsoleta, las bibliotecas que utilice JNI deberían ser *nPcap* y *libPcap*.
- Corrección de la biblioteca *JnetPcap* para que reconozca nuevos protocolos y poder seguir implementándolos en ella, siguiendo los pasos dados para estos 3 nuevos protocolos.
- Intentar ampliar la compatibilidad de plataformas de la aplicación para poder ser usada en Android, aunque esto puede resultar casi imposible, pues puede dar problemas con hacer promiscuas las tarjetas de red y, además, acapara muchos recursos.

Bibliografía

[1] S. Deering A. Conta, Transwitch. Internet control message protocol (icmpv6) for the internet protocol version 6 (ipv6) specification. https://datatracker.ietf.org/doc/html/rfc4443.

[2] S. Ansari, S.G. Rajeev, and H.S. Chandrashekar. Packet sniffing: a brief introduction. *IEEE Potentials*, 21(5):17–19, 2003.

[3] S. Deering B. Cain, Cereva Networks. Internet group management protocol, version 3. https://datatracker.ietf.org/doc/html/rfc3376.

[4] BogoToBogo. Eclipse cdt and jni with 64 bit mingw. https://www.bogotobogo.com/cplusplus/eclipse_CDT_JNI_MinGW_64bit.php.

[5] Eclipse Foundation. Eclipse ide. https://www.eclipse.org/.

[6] The Tcpdump Group. Tcpdump & libpcap. https://www.tcpdump.org.

[7] Raúl Merinero. Sniffer tfg (repo github). https://github.com/rms1005/Sniffer_TFG.

[8] Microsoft. Dinamic link library. https://learn.microsoft.com/es-es/troubleshoot/windows-client/deployment/dynamic-link-library.

[9] NFON. Request for comments (rfc). https://www.nfon.com/es/get-started/cloud-telephony/lexicon/base-de-conocimiento-destacar/request-for-comments-rfc.

[10] Oracle. Java. https://www.java.com/es/.

[11] Oracle. Virtualbox. https://www.virtualbox.org/.

[12] ISI P. Mockapetris. Internet group management protocol, version 3. https://datatracker.ietf.org/doc/html/rfc1035.

[13] Yiğit PIRILDAK. Java native access: A cleaner alternative to jni? https://levelup.gitconnected.com/java-native-access-a-cleaner-alternative-to-jni-954b53b77398.

[14] Nmap Project. Npcap. https://npcap.com/.

[15] Nmap Project. Npcap or winpcap? https://npcap.com/vs-winpcap.html.

[16] The Apache Ant Project. Apache ant. https://ant.apache.org/.

[17] J. Gettys R. Fielding, UC Irvine. Hypertext transfer protocol – http/1.1. https://www.rfc-editor.org/rfc/rfc2616.

[18] Scrum.org. Scrum. https://www.scrum.org/.

[19] Sly Technologies. jnetpcap - libpcap/winpcap java wrapper. https://sourceforge.net/projects/jnetpcap/.

[20] Riverbed Technology. Winpcap - the industry-standard windows packet capture library. https://www.winpcap.org/.

[21] Wikipedia. Ejemplo de diálogo http. https://es.wikipedia.org/wiki/Protocolo_de_transferencia_de_hipertexto#Ejemplo_de_di%C3%A1logo_HTTP.

[22] Wikipedia. gedit. https://es.wikipedia.org/wiki/Gedit.

[23] Wikipedia. Java native interface. https://es.wikipedia.org/wiki/Java_Native_Interface.

[24] Wikipedia. Microsoft windows. https://es.wikipedia.org/wiki/Microsoft_Windows.

[25] Wikipedia. Protocolo de comunicaciones. https://es.wikipedia.org/wiki/Protocolo_de_comunicaciones.

[26] Wikipedia. Ubuntu. https://es.wikipedia.org/wiki/Ubuntu.

[27] Wireshark. Wireshark. https://www.wireshark.org/.