UNIVERSIDAD
REY JUAN CARLOS

TITULACIÓN EN MAYÚSCULAS

Curso Académico 20XX/20XX

Trabajo Fin de Grado/Máster

TÍTULO DEL TRABAJO EN MAYÚSCULAS

Autor : Nombre del Alumno/a

Tutor : Dr. Gregorio Robles

Trabajo Fin de Grado/Máster

Título del Trabajo con Letras Capitales para Sustantivos y Adjetivos

Autor : Nombre del Alumno/a

Tutor : Dr. Gregorio Robles Martínez

La defensa del presente Proyecto Fin de Carrera se realizó el día de de 20XX, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 20XX

Dedicado a

mi familia / mi abuelo / mi abuela

II

Agradecimientos

Aquí vienen los agradecimientos... Aunque está bien acordarse de la pareja, no hay que olvidarse de dar las gracias a tu madre, que aunque a veces no lo parezca disfrutará tanto de tus logros como tú... Además, la pareja quizás no sea para siempre, pero tu madre sí.

Resumen

Aquí viene un resumen del proyecto. Ha de constar de tres o cuatro párrafos, donde se presente de manera clara y concisa de qué va el proyecto. Han de quedar respondidas las siguientes preguntas:

- ¿De qué va este proyecto? ¿Cuál es su objetivo principal?
- ¿Cómo se ha realizado? ¿Qué tecnologías están involucradas?
- ¿En qué contexto se ha realizado el proyecto? ¿Es un proyecto dentro de un marco general?

Lo mejor es escribir el resumen al final.

Summary

Here comes a translation of the “Resumen” into English. Please, double check it for correct grammar and spelling. As it is the translation of the “Resumen”, which is supposed to be written at the end, this as well should be filled out just before submitting.

Índice general

1. Introducción **1**

2. Objetivos **3**

2.1. Objetivo general 3

2.2. Objetivos específicos 3

2.3. Planificación temporal 4

3. Estado del arte **7**

3.1. Python 7

3.2. RTP 8

3.3. SIP y SDP 9

3.4. MP3 9

3.5. Bitstring 10

3.6. Wireshark 10

4. Diseño e implementación **11**

4.1. Arquitectura general 11

4.2. SimpleRTP 12

4.2.1. RtpHeader 12

4.2.2. RtpPayloadMP3 16

4.2.3. send_rtp_packet 17

4.3. Programa cliente 18

5. Resultados y pruebas **19**

5.1. Resultados en Wireshark 19

5.2. Pruebas con alumnos 23

6. Conclusiones **27**

6.1. Consecución de objetivos 27

6.2. Conocimientos aplicados 28

6.3. Futuros trabajos 29

A. Manual de usuario **31**

A.0.1. Mayor nivel de abstracción 31

A.0.2. Nivel medio de abstracción 32

A.0.3. Menor nivel de abstracción 33

A.0.4. Wireshark 34

Índice de figuras

5.1. Envío de paquetes RTP en Wireshark. 19
5.2. Cabecera de un paquete RTP con valores por defecto 20
5.3. Otro envío de paquetes RTP. Se puede observar los primeros valores de timestamp y número de secuencia diferentes. 21
5.4. Cabecera de un paquete RTP con CSRCs indicados por el cliente 21
5.5. Cabecera de un paquete RTP con todos los valores asignados desde el programa cliente. 22
5.6. Cabecera de un paquete RTP con todos los valores asignados desde el programa cliente y con errores. 23
5.7. Uso por una alumna de SimpleRTP. 24
5.8. Captura de paquetes RTP enviados con SimpleRTP por una alumna. 25

A.1. Wireshark sin analizar paquetes RTP. 34
A.2. Casilla a activar para poder ver tráfico RTP. 35

Capítulo 1

Introducción

Este Trabajo Fin de Grado nace de la asignatura Protocolos para la Transmisión de Audio y Video en Internet (PTAVI) y la necesidad que había, para la p

Capítulo 2

Objetivos

2.1. Objetivo general

El objetivo de mi trabajo de fin de grado es crear una herramienta de uso didáctico sobre RTP en la que, con diferentes niveles de complejidad, se pueda enviar tráfico RTP con los parámetros que el usuario quiera definir. De esta manera se puede analizar el comportamiento del protocolo bajo diferentes escenarios, comprobando su funcionamiento y capacidades.

El trabajo busca encontrar la manera más amigable y sencilla para el usuario de usar el protocolo RTP para enviar archivos de audio MP3, y ahondar en funciones más complejas según se quiera buscar detalle. Para este propósito he decidido elegir el formato de programa como API.

2.2. Objetivos específicos

Para llevar a cabo el objetivo general del trabajo, he tenido que realizar diferentes objetivos específicos que en su conjunto han formado el objetivo final. Se pueden diferenciar diferentes fases, aquí listadas:

- Analizar los protocolos implicados en el programa, estudiando cómo están formados los paquetes RTP y MP3 en su cabecera y payload. Esta fase es necesaria para entender la función y necesidad de cada campo e identificar aquellos que son importantes para poder tratarlos en el programa.

- Crear una versión inicial del programa donde el objetivo es integrar los protocolos entre ellos para crear paquetes correctamente, basándose en parámetros introducidos por el usuario. Entender e implementar la manera de manejar los bits que componen la cabecera en Python.

- Implementar funciones según avanza el programa. Dar más accesibilidad según un modelo API e integrar más funciones a la hora de la utilización añadiendo campos de la cabecera modificables por el usuario.

- Verificar el programa con alumnos. Se les ha dado a probar a los alumnos de la asignatura de Protocolos para la transmisión de audio y video en Internet la posibilidad de probar el programa. Con los resultados y sus opiniones ha sido posible mejorar el programa.

- Mejorar el programa implementando funciones más sencillas para su uso en función de las pruebas con los alumnos y corregir errores detectados en estas pruebas.

2.3. Planificación temporal

Este trabajo de fin de grado empieza en julio de 2019 al plantearse una idea inicial y sus objetivos. Entre agosto y septiembre hago los fines de semana un trabajo de investigación sobre los protocolos y un planteamiento inicial de como plasmar la idea en un programa en Python. También llevo a cabo una busqueda de programas con un objetivo similar al mío pero sin resultados satisfactorios, por lo que procedo a buscar librerías que puedan ayudarme a empezar. La razón de no dedicar más tiempo en estos momentos es por trabajo.

Entre octubre y noviembre desarrollo el programa partiendo de una situación inicial con todos los datos recabados anteriormente. Al no haber encontrado programas con objetivos parecidos se hace más complicado el desarrollo al tener que escribir todas las funciones del programa sin base previa. En estos dos meses me encuentro con el tutor Gregorio Robles varias veces para revisar el avance y dirigir el programa hacia el objetivo final.

En diciembre el programa ya está acabado con las funcionalidad que se consideran necesarias y se da a probar a los alumnos para poder identificar fallos y puntos de mejora.

En enero de 2020, tras recibir respuesta de los alumnos, identifico en qué ha fallado el programa y en qué ha tenido éxito y procedo a mejorar el programa y a pulirlo en función a

esto.

Entre enero y febrero acabo de pulir y eliminar errores del programa y escribo la memoria del Trabajo de Fin de Grado, finalizando este mismo.

Capítulo 3

Estado del arte

3.1. Python

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código, creado a finales de los ochenta? por Guido van Rossum. Es un lenguaje de programación multiparadigma, lo que significa que la programación puede ser orientada a objetos, imperativa o funcional. Es un lenguaje donde se hace incapié en la legibilidad del código y en la sencillez, usando palabras donde otros lenguajes usan símbolos y utilizando indentación para diferenciar los diferentes bloques de código que incluyen clases, funciones, bucles o condiciones entre otras cosas.

Tiene principios que al cumplirlos el código se considera ”pythoniano”, entre los cuales destaco:

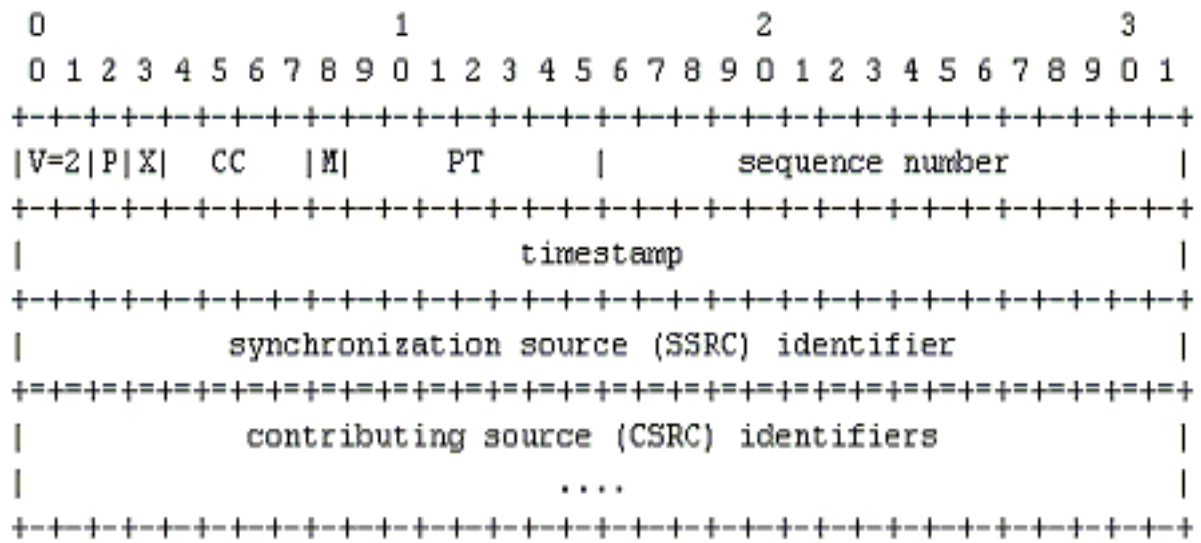
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Lo práctico gana a lo puro.
- Si la implementación es difícil de explicar, es una mala idea.

3.2. RTP

RTP o Real-time Transport Protocol es un protocolo a nivel de aplicación del modelo OSI utilizado para la transmisión de información audiovisual en tiempo real y end-to-end, descrito en la RFC 1889 como estándar en el año 1996 y actualizado en la RFC 3550 en 2003. Se usa principalmente junto al protocolo UDP de la capa de transporte del modelo OSI, que entre sus características no retransmite paquetes ni cuenta con control de flujo ni de aceptación, siendo un protocolo ideal para las aplicaciones el tiempo real al buscar la velocidad sin preocuparse de la fiabilidad. En las aplicaciones de streaming multimedia en tiempo real la pérdida de paquetes es tolerable debido a que la pérdida de un paquete en un paquete de audio es imperceptible con la ayuda de algoritmos para la corrección de fallos.

Una de las características de RTP es la de permitir diferentes formatos multimedia y permitir nuevos formatos sin tener que modificar el protocolo gracias a los perfiles y formatos de payload.

RTP funciona junto a un protocolo asociado a este llamado RTCP o Real-time Transport Control Protocol. Este se usa para enviar períodicamente información de control y parámetros de QoS o calidad de servicio y para sincronizar la sincronización entre las fuentes de flujos multimedia. RTCP consume poco ancho de banda comparado a RTP, alrededor de un 5 %.



3.3. SIP y SDP

Si bien este proyecto se centra en RTP y sus funcionalidades, caben destacar dos protocolos utilizados para la transmisión de contenido multimedia por RTP.

Las sesiones RTP donde se encuentran los flujos multimedia se inicializan mediante un protocolo de señalización. Entre ellos se encuentra SIP o Session Initiation Protocol que a su vez puede utilizar SDP o Session Description Protocol.

SIP (RFC 3261) es un protocolo utilizado para iniciar, mantener y finalizar sesiones multimedia en tiempo real. Es un protocolo basado en texto que comparte elementos de HTML y SMTP. Los diferentes end-points que utilizan SIP para comunicarse se llaman User Agents o UA, y actuan como UA Client al hacer una petición o como UA Server al responder a una petición. Otros agentes que actuan en la comunicación SIP son los Proxy Server, que actuan como intermediario y pueden hacer funciones de routing o seguridad, y los Registrar, que aceptan peticiones Register de los UA para mantener información sobre este incluyendo su dirección.

SDP (RFC 4566) es un formato para describir parámetros de sesión en una comunicación multimedia en tiempo real. Entre sus funciones se encuentran los anuncios e invitaciones de sesiones y la negociación de parámetros entre los end-points, que juntos forman el perfil de la sesión. Al igual que SIP, está diseñado para soportar nuevos formatos sin cambiar el formato.

3.4. MP3

MP3 o MPEG-1 Audio Layer III es un estándar de comprensión de audio definido en las ISO 11172-3 e ISO 13818-4 y uno de los formatos de audio común más usados en el mundo. Entre sus características se encuentra una fuerte compresión que causa pérdidas de información pero un menor tamaño de archivo. Las pérdidas provocadas por la compresión sin embargo no son perceptibles al oído humano gracias a modelos psicoacústicos.

Dentro de esta estándar los ficheros de audio pueden tener diferente velocidad de bit rate (kbits/s) y de velocidad de muestreo (Hz). Ambos parámetros se definen en la cabecera del paquete MP3.

3.5. Bitstring

Bitstring es una librería de Python diseñada para facilitar el análisis y procesado de datos en formato binario.

Bitstring define un objeto que se puede construir con variables con una gran variedad de tipos, desde integer big y little endian hasta string, hexadecimal, byte y un proprio binario.

Los objetos bitstring tienen diferentes funciones para unir, truncar, adjuntar y escribir el propio objeto y así modificar su contenido en bits. Estos objetos se pueden leer, buscar y reemplazar para poder tratarlos.

Todas estas características han resultado útiles para la lectura, manejo y creación de cabeceras en formato binario, tanto para las de RTP como para las de MP3.

3.6. Wireshark

Wireshark es un analizador de protocolos de red que permite analizar a muy bajo nivel todos los paquetes que pasan por la red, disponible para una gran cantidad de plataformas.

Tiene integrados una gran cantidad de protocolos en continuo aumento que permite detectarlos y analizar sus cabeceras y permite capturar tráfico en tiempo real o analizar capturas guardadas para un análisis posterior. También analiza exhaustivamente tráfico de VoIP (Voice over IP). Puede filtrar el tráfico en base a protocolos o a parámetros decididos por el usuario para tener una captura limpia solo con el tráfico que interesa.

Todas estas características son interesantes y han ayudado mucho a comprobar el funcionamiento del programa al poder ver en detalle los paquetes RTP enviados y sus cabeceras, comprobando que todos los campos que el usuario indicaba al llamar a las funciones se veían reflejados en tráfico real.

Capítulo 4

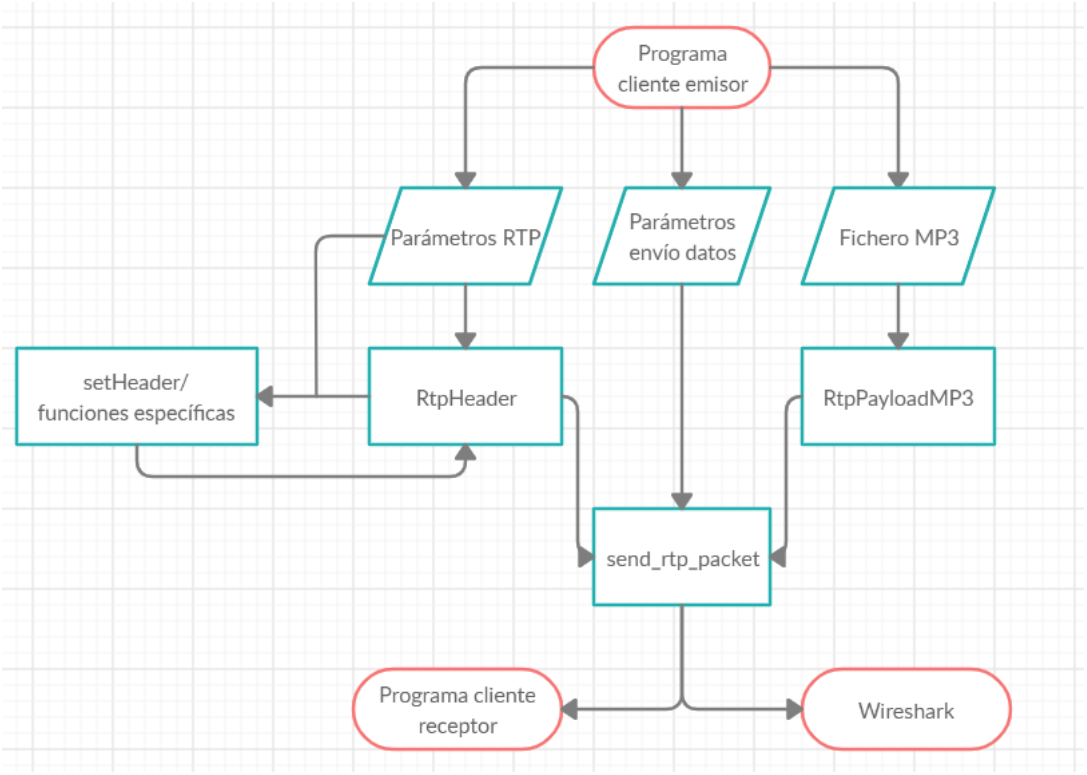
Diseño e implementación

4.1. Arquitectura general

El programa está desarrollado en un modelo API, lo que significa que sirve para dar funciones y procedimientos a otro software sin que este se tenga que preocupar del funcionamiento interno de las funciones utilizadas. Esta capa de abstracción hace que el uso sea más fácil para el usuario. En el caso de SimpleRTP, se ofrecen diferentes niveles de abstracción con el objetivo de poder profundizar en las opciones del programa.

El programa cliente emisor debe importar la librería SimpleRTP para poder abrir una conexión UDP y enviar paquetes RTP, mientras que un posible programa receptor que escuche en el puerto objetivo no necesita importarla.

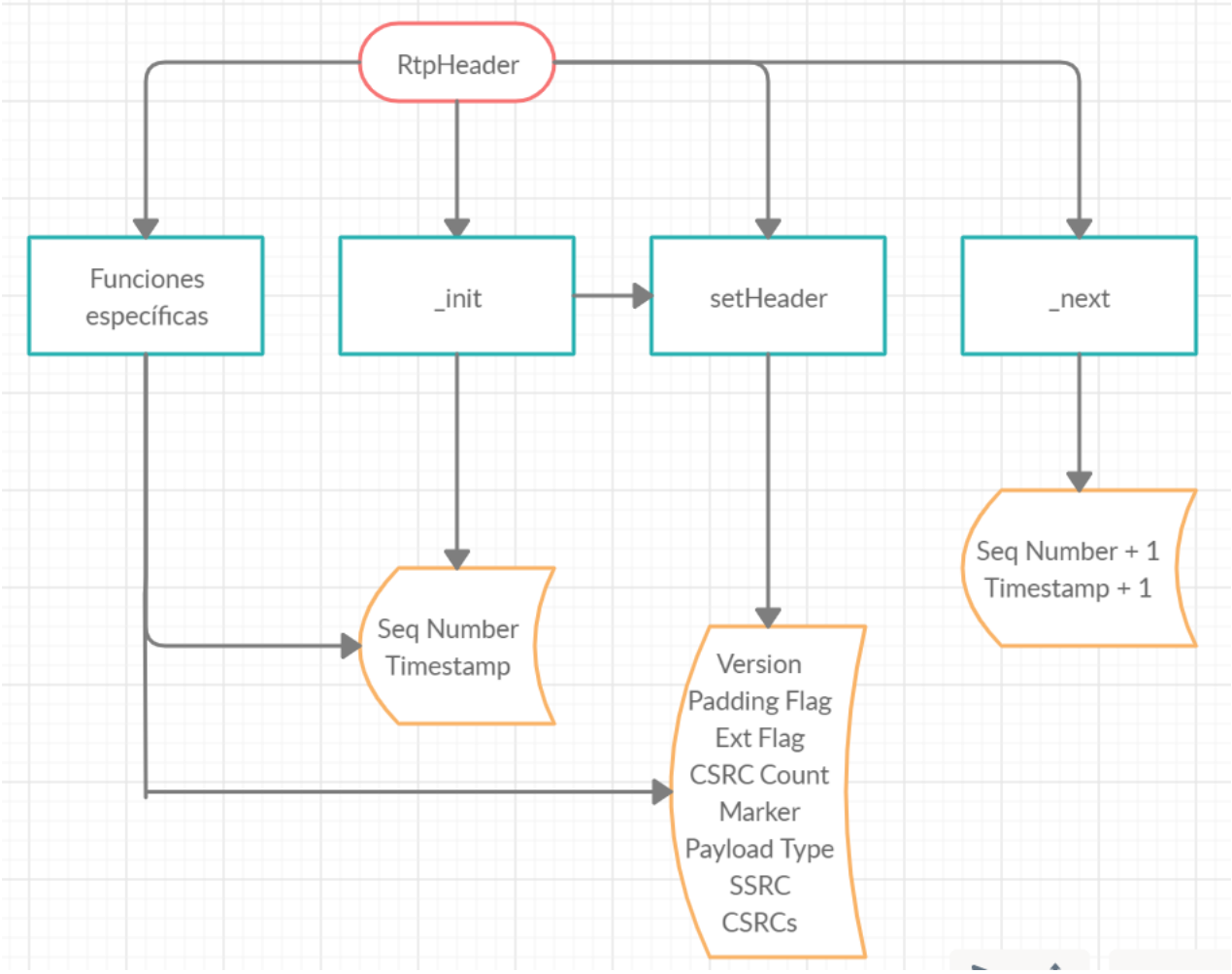
A continuación se muestra un diagrama del funcionamiento del programa con los diferentes agentes que participan en el proceso del envío de paquetes:



4.2. SimpleRTP

4.2.1. RtpHeader

El componente fundamental de este programa es la clase RtpHeader. Un objeto RtpHeader contiene todos los valores que forman una cabecera RTP y es indispensable crear un objeto para el envío de paquetes RTP. Tiene varias funciones con diversos propósitos que se detallarán a continuación. El diagrama que representa el flujo de funcionamiento de la clase es el siguiente:



Init

Al crearse el objeto se ejecuta la función Init:

```
def __init__(self, version=2, pad_flag=0, ext_flag=0, cc=0, marker=0, pay
        self.seq_number = random.randint(1, 10000)  # Aleatorio
        self.timestamp = random.randint(1, 10000)  # Aleatorio
        self.set_header(version, pad_flag, ext_flag, cc, marker, payload_
```

Todos los argumentos con los que se crea un objeto son opcionales y tienen valores por defecto, por lo que no es necesario para un uso básico usar ninguno. Por defecto se define un número de secuencia y timestamp aleatorios (impredecibles) para reforzar la seguridad y evitar ciertos ataques que se basan en estos valores. Estos dos valores se guardan en variables para su uso más adelante.

La función Init llama a la siguiente función que también puede llamar el usuario, set header. Su lógica detrás de esto es la con una sola línea y sin argumentos ni variables se pueda crear una cabecera RTP completa. Los argumentos que se hayan pasado al crear el objeto (si se han pasado, si no los valores por defecto) se utilizan para llamar a set header.

Set header

Esta función puede ser invocada por el usuario o al final de la función Init:

```
def set_header(self, version=2, pad_flag=0, ext_flag=0, cc=0, marker=0, p
        self.version = BitArray(uint = version, length = 2)
        self.pad_flag = BitArray(uint = pad_flag, length = 1)
        self.ext_flag = BitArray(uint = ext_flag, length = 1)
        self.cc = BitArray(uint = cc, length = 4)
        self.marker = BitArray(uint = marker, length = 1)
        self.payload_type = BitArray(uint = payload_type, length = 7)
        self.ssrc = BitArray(uint = ssrc, length = 32)
        self.csrc = BitArray()
```

Todos los valores que se pasen a set header se guardan en variables dentro del objeto. Cada variable es un campo de la cabecera RTP y se guardan con un tipo BitArray. Este tipo se importa de la biblioteca bitstring y pasado un valor de tipo integer y la longitud en bits se genera un número binario con ciertas características que le da el ser del tipo BitArray, entre ellas operaciones y poder unirse o separarse con otros BitArray.

Funciones específicas

A continuación las funciones específicas que modifican un solo campo de la cabecera RTP y se pueden invocar desde el cliente:

```
def setVersion(self, version):
        self.version = BitArray(uint = version, length = 2)


    def setPaddingFlag(self, pad_flag):
```

```
        self.pad_flag = BitArray(uint = pad_flag, length = 1)

    def setExtensionFlag(self, ext_flag):
        self.ext_flag = BitArray(uint = ext_flag, length = 1)

    def setCsrcCount(self, cc):
        self.cc = BitArray(uint = cc, length = 4)

    def setMarker(self, marker):
        self.marker = BitArray(uint = marker, length = 1)

    ...
```

La unica función algo diferente es la que guarda el valor de los identificadores CSRC. Cada valor CSRC forma parte de un array que es el argumento de la función, y uno a uno se convierten a valores binarios y se concatenan.

Next

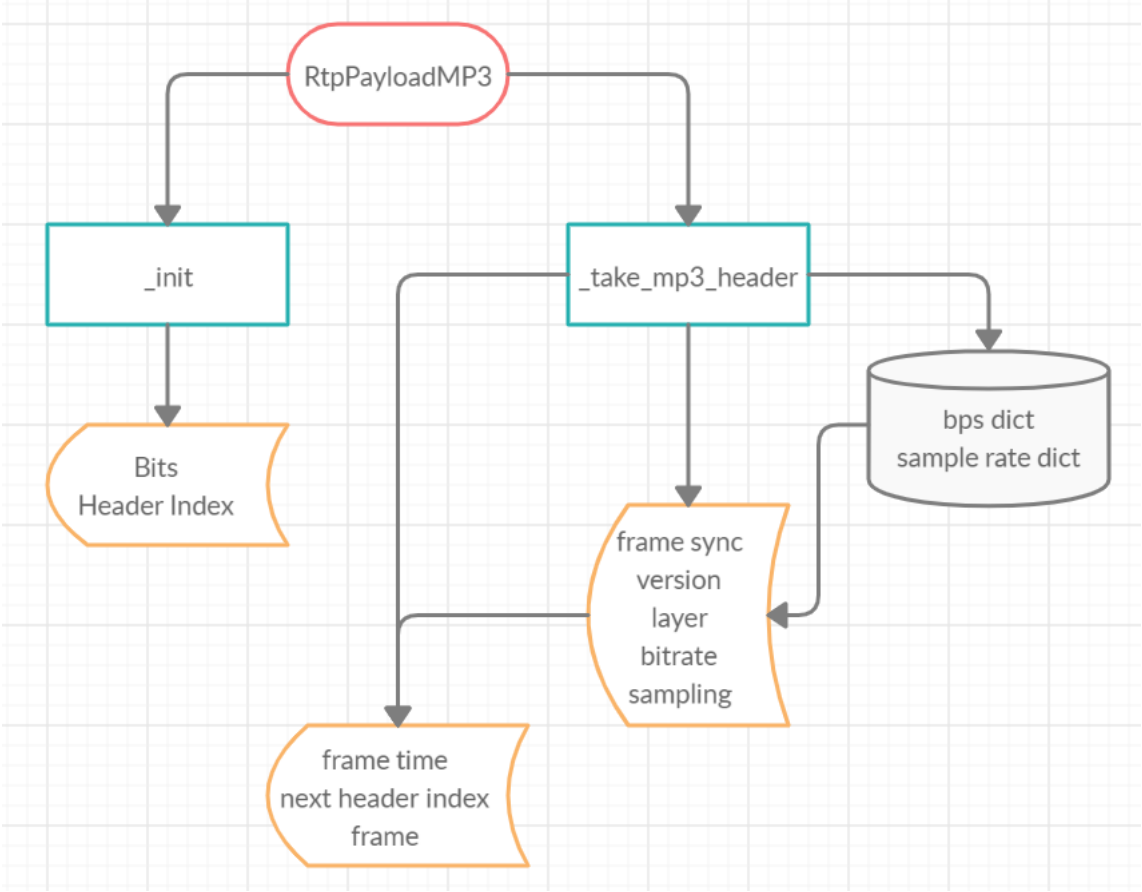
Esta función no se recomienda invocar por el usuario. Se invoca automáticamente cada vez que se envía un paquete RTP y aumenta el valor del número de secuencia en 1 y el valor del timestamp en función del tiempo de duración de un frame MP3 en milisegundos.

```
def _next(self, frameTimeMs):
        self.seq_number += 1
        # Calcular siguiente timestamp
        self.timestamp += int(8000 * (frameTimeMs/1000))
```

4.2.2. RtpPayloadMP3

Una vez tenemos en un objeto todo lo necesario para formar la cabecera de un paquete RTP, el siguiente paso es el payload o la carga útil del paquete. Aquí es donde se encuentra el paquete MP3. Un fichero MP3 es una sucesión continua de paquetes por lo que es necesario localizar la cabecera de cada paquete y analizarla para ver sus cualidades como el tiempo de muestreo o los bits por segundo.

La clase RtpPayloadMP3 tiene dos funciones. La primera, init, es la que se llama al crear un objeto de la clase y la segunda, llamada _take_mp3_frame es la encargada de analizar el paquete. El diagrama que representa el flujo de funcionamiento de la clase es el siguiente:



init

Al crear un objeto se llama a la función Init. El objetivo de esta función es manejar el fichero de audio MP3 que se le pasa, pasarlo a binario y encontrar el inicio de la primera cabecera:

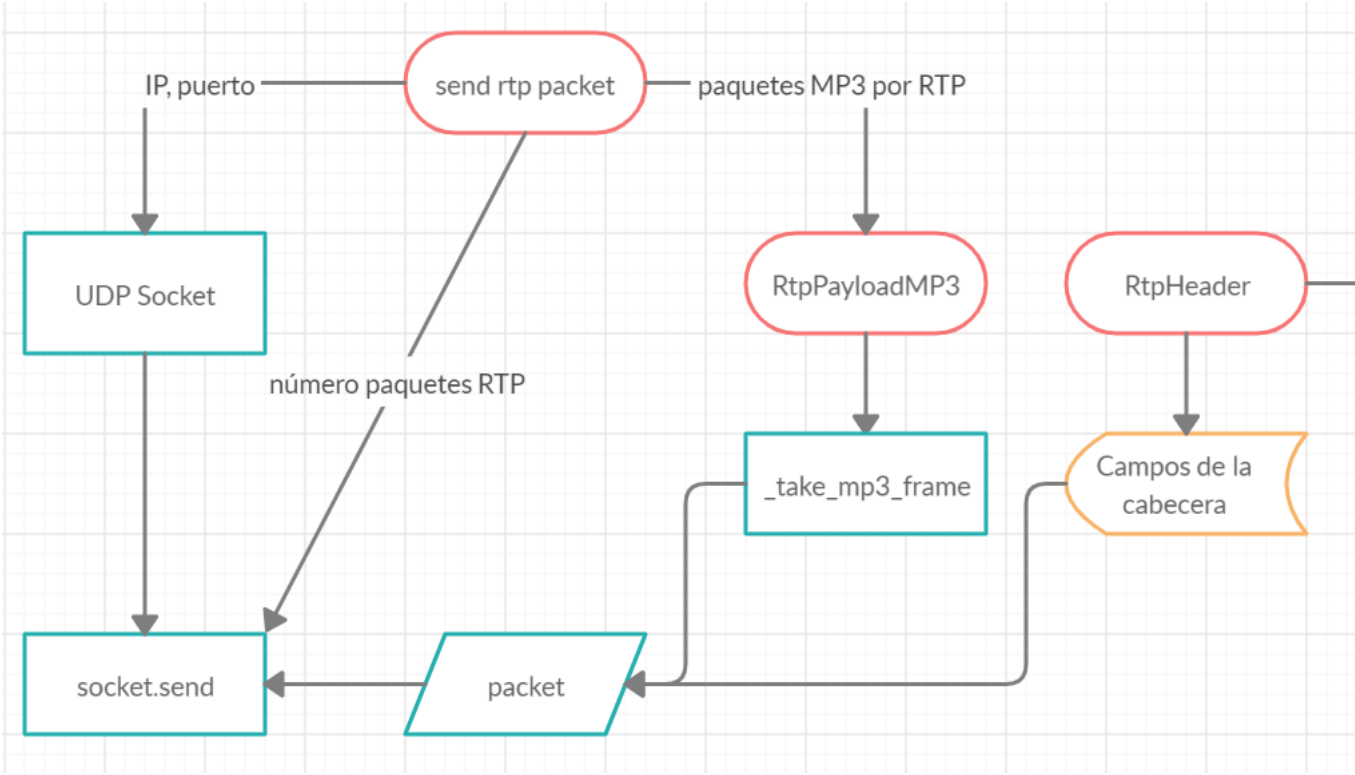
```
def __init__(self, file_path):
        with open(file_path, "rb") as file:
```

```
            bytes = file.read()
            self.bits = BitArray(bytes).bin
            self.header_index = self.bits.find('11111111111')
\end{verbatin}

El fichero se pasa a binario para poder trocearlo en paquetes e indexar e

\subsubsection{\_take\_mp3\_frame}

Cuando se invoca a esta función, se analiza el siguiente paquete que se v

El campo version (en inglés) de dos bits indica si nos encontramos ante u

Con estos valores podemos calcular el tamaño en bits del paquete, y por l

A continuación una parte de la función donde se ven los últimos pasos:

\begin{verbatim}
        frame_length = int(144 * 8 * (bps/sample_rate))
        # tiempo por frame en milisegundos
        self.frameTimeMs = int(144/sample_rate * 1000 * 8)
        next_mp3_header_index = self.header_index + frame_length
        self.frame = self.bits[self.header_index:next_mp3_header_index]
        self.header_index = next_mp3_header_index
```

4.2.3. send_rtp_packet

La función send_rtp_packet es la encargada de juntar las diferentes partes que componen el paquete RTP completo y enviar el paquete mediante un socket UDP. En esta función es donde se define el número de paquetes RTP que se quieren enviar y la cantidad de paquetes MP3 en cada paquete RTP.

El diagrama que representa el flujo de funcionamiento de la función es el siguiente:



Tras abrir el socket UDP con la dirección indicada, se empiezan a enviar paquetes RTP en bucle. El paquete que se envía se forma juntando los diferentes campos que contiene el objeto RtpHeader en orden y tras esto se juntan al final el número de paquetes MP3 indicados, que forman el payload del paquete final. Cada vez que se añade un paquete MP3 se llama a la función _take_mp3_frame que es la encargada de identificar el siguiente paquete.

Una vez formado el paquete se convierte a bytes y se envía. Después se aumentan los valores de número de secuencia y timestamp RTP para prepararse para enviar el siguiente paquete.

4.3. Programa cliente

Todo lo relativo al programa cliente lo he incluido en el capítulo A. Ahí se explica como utilizar SimpleRTP desde el punto de vista del cliente de diferentes formas según sea el objetivo. Todas las referencias a los niveles de abstracción en esta memoria se refieren a los niveles de abstracción del programa cliente que se pueden ver en el manual de usuario.

Capítulo 5

Resultados y pruebas

5.1. Resultados en Wireshark

El objetivo de SimpleRTP es el envío de paquetes RTP. Para comprobar que los paquetes se envian correctamente me ha ayudado la herramiente Wireshark. Con Wireshark es posible analizar los paquetes RTP en profundidad y comprobar que se ha formado el paquete correctamente con todos los campos correspondientes.

La primera prueba para verificar los resultados la realizo usando SimpleRTP de la manera más simple posible, utilizando el ejemplo de mayor nivel de abstracción de los apartados anteriores. Podemos ver el resultado en la figura 5.1.

Se puede observar que el primer paquete RTP tiene un número de secuencia y timestamp que no empiezan en 0, y que en cada paquete aumentan sus valores. Todos los paquetes tienen el mismo tamaño, ya que todos tienen en su payload el mismo número de paquetes MP3 y las cabeceras son iguales.

1	0.000000	127.0.0.1	127.0.0.1	RTP	2060 PT=MPEG-I/II Audio, SSRC=0x3E8, Seq=1636, Time=766
2	0.003597	127.0.0.1	127.0.0.1	RTP	2060 PT=MPEG-I/II Audio, SSRC=0x3E8, Seq=1637, Time=1054
3	0.009432	127.0.0.1	127.0.0.1	RTP	2060 PT=MPEG-I/II Audio, SSRC=0x3E8, Seq=1638, Time=1342
4	0.013083	127.0.0.1	127.0.0.1	RTP	2060 PT=MPEG-I/II Audio, SSRC=0x3E8, Seq=1639, Time=1630
5	0.019525	127.0.0.1	127.0.0.1	RTP	2060 PT=MPEG-I/II Audio, SSRC=0x3E8, Seq=1640, Time=1918
6	0.023243	127.0.0.1	127.0.0.1	RTP	2060 PT=MPEG-I/II Audio, SSRC=0x3E8, Seq=1641, Time=2206
7	0.026522	127.0.0.1	127.0.0.1	RTP	2060 PT=MPEG-I/II Audio, SSRC=0x3E8, Seq=1642, Time=2494
8	0.034621	127.0.0.1	127.0.0.1	RTP	2060 PT=MPEG-I/II Audio, SSRC=0x3E8, Seq=1643, Time=2782
9	0.038078	127.0.0.1	127.0.0.1	RTP	2060 PT=MPEG-I/II Audio, SSRC=0x3E8, Seq=1644, Time=3070
10	0.049500	127.0.0.1	127.0.0.1	RTP	2060 PT=MPEG-I/II Audio, SSRC=0x3E8, Seq=1645, Time=3358
11	0.054277	127.0.0.1	127.0.0.1	RTP	2060 PT=MPEG-I/II Audio, SSRC=0x3E8, Seq=1646, Time=3646
12	0.059967	127.0.0.1	127.0.0.1	RTP	2060 PT=MPEG-I/II Audio, SSRC=0x3E8, Seq=1647, Time=3934
13	0.063794	127.0.0.1	127.0.0.1	RTP	2060 PT=MPEG-I/II Audio, SSRC=0x3E8, Seq=1648, Time=4222

Figura 5.1: Envío de paquetes RTP en Wireshark.

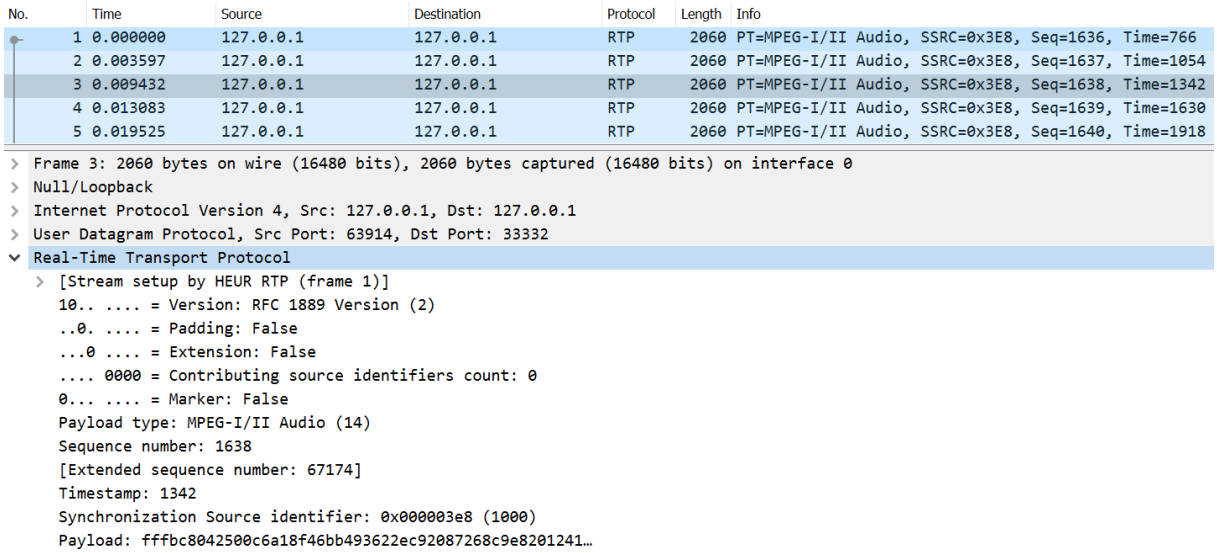


Figura 5.2: Cabecera de un paquete RTP con valores por defecto

Al enviar paquetes de esta manera, los valores que toma la cabecera son todos por defecto. Esto significa que tenemos los siguientes valores:

```
version = 2
pad_flag = 0
ext_flag = 0
cc = 0
marker =  0
payload_type = 14
ssrc = 1000
```

Podemos comprobar al analizar la cabecera RTP del paquete en Wireshark que todos los valores son correctos en la figura 5.2. Un primer indicio de que todo ha ido bien es que el propio Wireshark reconoce el protocolo RTP, por lo que el paquete parece estar bien formado. También se puede observar el tamaño del paquete. Este tamaño depende del payload y en este caso son 2 paquetes MP3 por RTp, que es el valor por defecto ya que no hemos indicado nada.

El siguiente paso es poner a prueba el programa enviando paquetes usando el ejemplo de nivel de abstracción medio. Aquí se hace uso de una función específica para un campo y se pasa un argumento también a la función set_header. También voy a cambiar el número de paquetes MP3 en cada paquete RTP a 3 para poder ver el cambio del tamaño del paquete final. Capturando

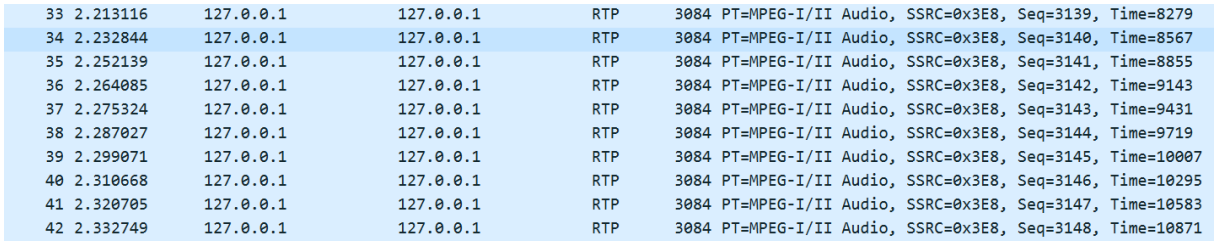


Figura 5.3: Otro envío de paquetes RTP. Se puede observar los primeros valores de timestamp y número de secuencia diferentes.

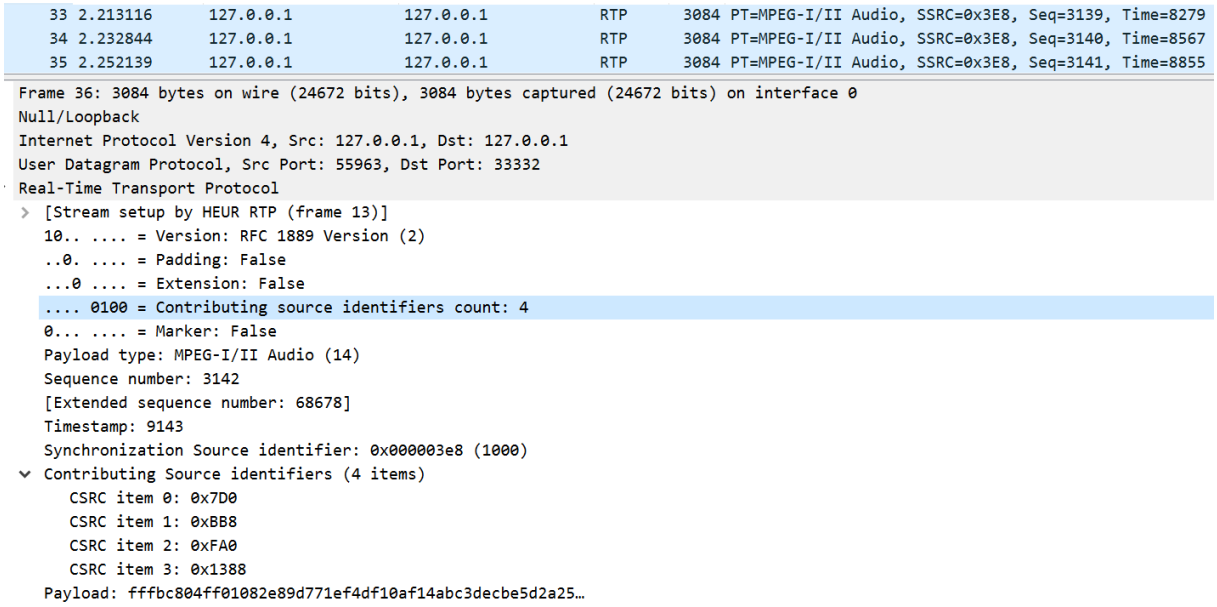


Figura 5.4: Cabecera de un paquete RTP con CSRCs indicados por el cliente

el tráfico obtenemos el siguiente resultado reflejado en las figuras 5.3 y 5.4.

Lo primero que se puede notar es que el tamaño de los paquetes ha cambiado. Esto no se debe solo al mayor payload, al haber 3 paquetes MP3 encapsulados en vez de 2, si no también a la inclusión de 4 CSRCs en la cabecera RTP que no estaban antes. Cada ID de un CSRC ocupa 32 bits o 4 bytes más en la cabecera.

En el detalle de la cabecera RTP, casi todos los campos siguen tomando los valores por defecto, menos el CC (ya que al final es la cantidad de CSRCs presentes en la cabecera) y los nuevos CSRCs. Podemos comprobar que todos los valores son correctos, ya que o son los indicados por el programa cliente o son los valores tomados por defecto. Los valores que toman los CSRCs se muestran en hexadecimal, pero son correctos al corresponderse a 2000, 3000, 4000 y 5000 en decimal respectivamente.

```
243 5.250218    127.0.0.1    127.0.0.1    RTP    1072 PT=MPEG-I/II Audio, SSRC=0x457, Seq=5449, Time=65269, Mark
244 5.255679    127.0.0.1    127.0.0.1    RTP    1072 PT=MPEG-I/II Audio, SSRC=0x457, Seq=5450, Time=65557, Mark
245 5.260221    127.0.0.1    127.0.0.1    RTP    1072 PT=MPEG-I/II Audio, SSRC=0x457, Seq=5451, Time=65845, Mark
246 5.263044    127.0.0.1    127.0.0.1    RTP    1072 PT=MPEG-I/II Audio, SSRC=0x457, Seq=5452, Time=66133, Mark
247 5.271452    127.0.0.1    127.0.0.1    RTP    1072 PT=MPEG-I/II Audio, SSRC=0x457, Seq=5453, Time=66421, Mark
248 5.274791    127.0.0.1    127.0.0.1    RTP    1072 PT=MPEG-I/II Audio, SSRC=0x457, Seq=5454, Time=66709, Mark

Real-Time Transport Protocol
  [Stream setup by HEUR RTP (frame 25)]
  10.. .... = Version: RFC 1889 Version (2)
  ..0. .... = Padding: False
  ...0 .... = Extension: False
  .... 0101 = Contributing source identifiers count: 5
  1... .... = Marker: True
  Payload type: MPEG-I/II Audio (14)
  Sequence number: 5456
  [Extended sequence number: 70992]
  Timestamp: 67285
  Synchronization Source identifier: 0x00000457 (1111)
  Contributing Source identifiers (5 items)
    CSRC item 0: 0x7D0
    CSRC item 1: 0xBB8
    CSRC item 2: 0xFA0
    CSRC item 3: 0x1388
    CSRC item 4: 0x1770
  Payload: fffbc804b40b97a280cec34f63f2eab05a05670c18dbb1fd…
```

Figura 5.5: Cabecera de un paquete RTP con todos los valores asignados desde el programa cliente.

Otra prueba es cambiando más campos a la cabecera RTP pasando valor a todos los campos, como en el ejemplo de menor nivel de abstracción, pero dejando algunos valores iguales que los de por defecto. Este último cambio lo hago en el último ejemplo ya que asignar valores sin sentido a los campos de la cabecera provocará errores. En la figura 5.5 podemos ver la cabecera con más cambios y con otro tamaño de paquete para comprobar los cambios de tamaño de payload, habiendo ahora un paquete MP3 encapsulado.

En esta captura se comprueba que todo sigue funcionando correctamente con los nuevos valores que hemos indicado. A continuación una última prueba en la que cambiamos todos los campos indiscriminadamente, sin atender realmente al significado que puede tener un 0 o un 1 en los indicadores de extensión o padding. El unico campo que no voy a cambiar para estos ejemplos es el de versión, porque Wireshark deja de reconocer el protocolo del paquete como RTP y no podemos analizar las cabeceras. El resultado de esta ejecución se ve en la figura 5.6.

En esta prueba podemos ver el problema desde el primer momento al indicar Wireshark que el paquete está mal formado. Sigue reconociendolo como un paquete RTP al no haber tocado la versión y por esto mismo entiende el resto de campos y concluye que algo hay mal. En este caso el cliente ha decidido indicar que los campos que indican que hay padding y extensión se activan, pero realmente la cabecera no tiene estos campos. Este ejemplo sirve para demostrar que se pueden provocar errores intencionadamente con SimpleRTP para ver como

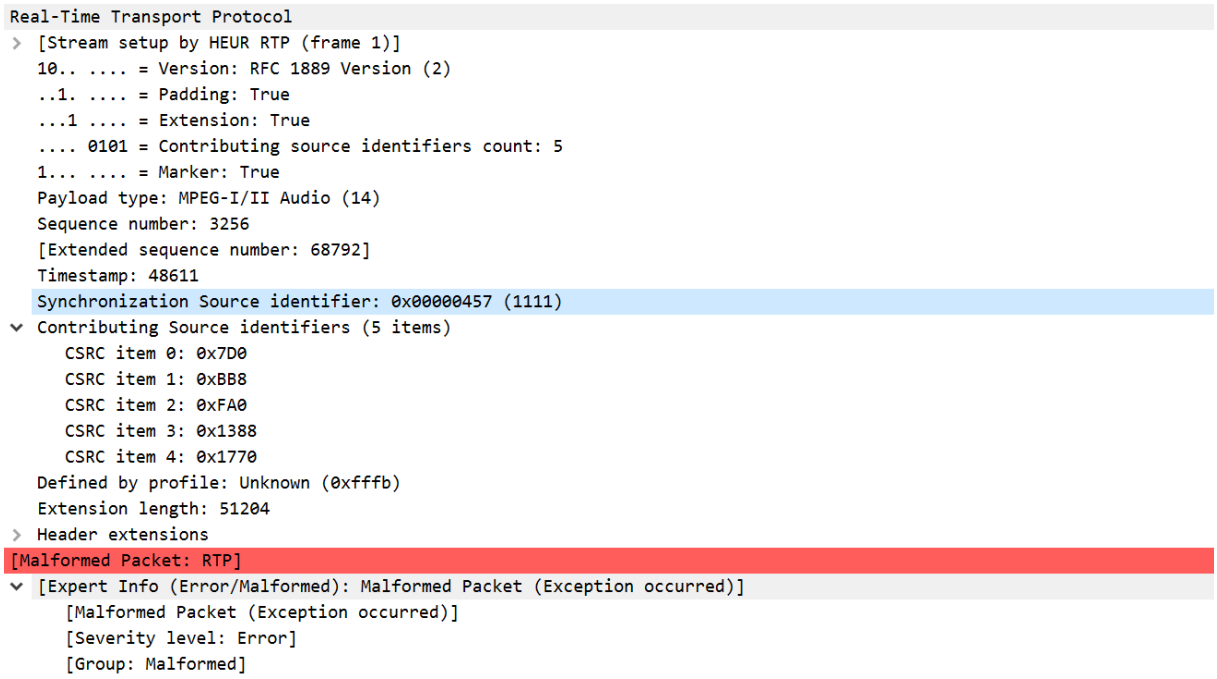


Figura 5.6: Cabecera de un paquete RTP con todos los valores asignados desde el programa cliente y con errores.

estos paquetes mal formados se comportan en la red.

5.2. Pruebas con alumnos

Para poner a prueba el programa, se le proporciono una versión no final a los alumnos de la asignatura de 'Protocolos para la transmisión de audio y video en Internet' del grado de Ingeniería en sistemas audiovisuales y multimedia. El objetivo era comprobar el grado de funcionamiento del programa expuesto a varias pruebas hechas por otras personas y analizar si el programa lograba su objetivo de ofrecer una manera fácil de usarse para propósitos educativos.

Tras dar un tiempo a los alumnos para integrar SimpleRTP en su práctica de la asignatura, se han podido identificar varios puntos a cambiar. Al analizar el código que usaron los alumnos, se pudo ver que no supieron entender como integrar este programa con el que ya tenían, ya que combinaban el método que utilizaban antes para enviar paquetes RTP con la biblioteca de SimpleRTP.

Debido a esto, trabajé en hacer niveles de complejidad menores que derivaron en el nivel de abstracción mayor que detallé en el apartado anterior. Con esto se busca que no sea nece-

```
def send_rtp(origen_ip, origen_puertortp):
    """Send multimedia content by RTP."""
    # Ejecutar y escuchar un string con lo que se ha de ejecutar en la shell
    aEjecutar = "./mp32rtp -i " + origen_ip + " -p " + origen_puertortp
    aEjecutar += " < " + AUDIO_PATH
    aEscuchar = "cvlc rtp://@" + origen_ip + ":" + origen_puertortp + '&'
    hcvlc = threading.Thread(target=os.system(aEscuchar))
    hcvlc.start()
    log.log_rtp(origen_ip, origen_puertortp, AUDIO_PATH)

    # envio RTP sin mp32rtp
    cabeceraRTP = simplertp.RtpHeader()
    csrc = [2000, 3000, 4000, 5000]
    cabeceraRTP.set_header(version=2, pad_flag=0, ext_flag=0, cc=4, marker=0, payload_type=90, ssrc=1000)
    cabeceraRTP.setCSRC(csrc)
    audio = simplertp.RtpPayloadMp3()
    audio.set_audio(AUDIO_PATH)
    numeroPaquetesRTP = 0
    paquetesMP3porRTP = 2
    ip = origen_ip
    port = origen_puertortp
    simplertp.send_rtp_packet(numeroPaquetesRTP, cabeceraRTP, audio, ip, int(port), paquetesMP3porRTP)

    return aejecutar
```

Figura 5.7: Uso por una alumna de SimpleRTP.

sario comprender en profundidad el programa ni el funcionamiento de RTP, haciendo lo más fácil posible enviar paquetes. Otros fallos identificados gracias a las capturas enviadas por una alumna eran fallos menores del programa que corregí gracias a las pruebas, como por ejemplo la inclusión de los diccionarios con diferentes bps y frecuencias de muestreo para todos los tipo de ficheros de MP3. Gracias a esto SimpleRTP soporta más tipos de ficheros MP3 y es más robusto frente a este tipo de errores. Se puede ver el uso de SimpleRTP junto al otro método en la figura 5.7.

Aún con los fallos que puedo encontrar esta alumna, el envío de paquetes RTP mediante este programa fue satisfactorio, como se puede observar en una captura en Wireshark donde se aprecia un paquete con los parámetros que indicó usando las funciones de la biblioteca SimpleRTP.

Por lo tanto, considero que el mayor problema fue entregar el programa con solo funcionalidades de nivel de abstracción bajo y que los alumnos no pudieron interpretar bien el funcionamiento del programa, al combinar el uso de este con el envío de paquetes de otras maneras. En cambio, pude comprobar el correcto funcionamiento de este al ver en la captura de Wireshark el envío de paquetes RTP mediante las funciones de SimpleRTP.

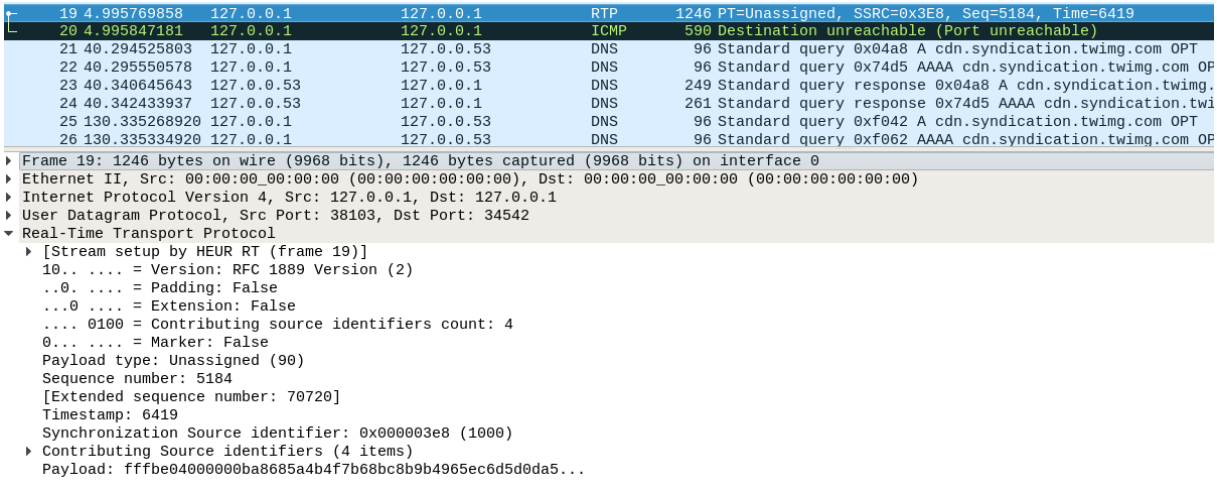


Figura 5.8: Captura de paquetes RTP enviados con SimpleRTP por una alumna.

Capítulo 6

Conclusiones

6.1. Consecución de objetivos

Es importante análizar si se han cumplido los objetivos del trabajo y en qué grado para concluir si se ha llevado a cabo satisfactoriamente. El fin para el que he hecho este programa es para ser usado por un estudiante o una persona que quiera poner a prueba el protocolo RTP para estudiarlo y comprenderlo de mejor manera y, si es posible, que se utilice en la asignatura de PTAVI para sustituir el programa que se estaba utilizando para el envío de RTP, ya que SimpleRTP ofrece mejores funcionalidades a la hora de aprender.

Respecto a este último punto, no será posible ver los verdaderos resultados hasta que se use en la asignatura el próximo curso y que se pueda ver su efectividad real con una mayor base de usuarios. Relacionado con los alumnos está la prueba que se hizo con los de este curso, a los que se les dió la posibilidad opcional de probar el programa en una práctica. Es cierto que esperaba una mayor participación, al haberlo utilizado solo dos alumnos, pero los resultados obtenidos no obstante me han servido de gran ayuda para corregir errores e identificar puntos a mejorar. Por lo tanto podría concluir que, aunque pueda parecer una baja participación, los resultados han sido útiles y considerar satisfactoria la prueba.

Los objetivos intermedios necesarios para llevar a cabo el trabajo han sido, como comenté en el capítulo 2, la adquisición de conocimiento teórico sobre los protocolos a tratar y el planteamiento desde cero de como programar lo que buscaba. Considero que estos objetivos los he llevado a cabo con éxito, ya que para poder enseñar algo y conseguir que se aprenda hay que entenderlo primero, y si la tarea de programación consigue que algo de mayor compleji-

dad se vuelva simple es porque he entendido a fondo como funcionan los diferentes factores involucrados.

Puedo concluir este apartado con que he logrado los objetivos propuestos pero con un rango de mejora que comento más adelante.

6.2. Conocimientos aplicados

Para llevar a cabo el programa SimpleRTP me han sido de gran ayuda varias asignaturas del Grado en Ingeniería en Sistemas Audiovisuales y Multimedia, que me han aportado el conocimienton necesario para poder plantear y desarrollar el programa y para solventar los problemas que hayan podido surgir durante el trascurso.

Sin duda la asignatura fundamental sobre la que se basa este trabajo es la asignatura Protocolos para la Transmisión de Audio y Video en Internet. Con esta asignatura conocí el protocolo RTP y los asociados como SIP y SDP, base del programa que he desarrollado y donde más he profundizado. Otro conocimiento fundamental que he adquirido en esta asignatura es el lenguaje de programación Python, importante no solo para este trabajo o la asignatura si no para el futuro al ser un lenguaje importante y con mucho potencial.

Aunque haya sido la asignatura sobre la que he basado el programa, las conocimientos de las asignaturas de los años anteriores son necesarios para llegar al nivel de conocimiento que requiere PTAVI. Las asignatura Informática I fue la puerta de entrada al mundo de la programación e Informática II me propuso retos más complicados que consiguieron que aumentase mucho mi nivel programando. Que todas las asignaturas de programación se impartiesen con diferentes lenguajes también me ayudó a saber adaptarme y no limitarme a uno solo.

Aún habiendo citado estas asignaturas en concreto, considero que todas las asignaturas del grado me han servido para desarrollarme como estudiante y como persona, y por lo tanto en el resultado de este Trabajo Fin de Grado. Hay asignaturas que han podido resultarme más sencillas por haber captado mi interés y coincidir con mis gustos y otras en las que he tenido que dedicar mucho más tiempo para poder absorber los conocimientos que quiere transmitir, pero al final todas de un modo u otro me han hecho enfrentarme a problemas que en su totalidad han constituido un reto que he conseguido superar. Mirando atrás desde que empecé el grado veo que cuando en su momento dudé con alguna asignatura, ahora veo su utilidad.

6.3. Futuros trabajos

Como he dicho el programa tiene un amplio rango de mejora, con funcionalidades adicionales que pueden dar más versatilidad y contenido al programa. Mi plan de futuro con este programa es poder mejorarlo según requiera la situación y cuando se puedan ver las necesidades cuando lo utilicen más alumnos.

- Entre las nuevas funcionalidades que se pueden incluir se encuentra todo lo relativo al protocolo RTCP, donde se pueden añadir funciones como las que pertenecen a la cabecera RTP para poder controlar todo el contenido del protocolo.

- También me gustaría añadir más codecs de audio. Si bien es una opción que investigué en su momento, hay varios protocolos que son de empresas donde la información pública es mucho más limitada y su inclusión en este proyecto excluí por las complicaciones que aportaba y porque el protocolo MP3 es ampliamente el más utilizado y suficiente para transmitir paquetes RTP, que es la función final del programa.

Todo se puede mejorar y estoy dispuesto a ello, empezando por las mejoras que ya he identificado y siguiendo por todas aquellas que identifiquen los usuarios, ya que las necesidades reales surgen cuando lo utilizan los clientes finales.

Apéndice A

Manual de usuario

El programa cliente puede tener cualquier propósito y diferentes implementaciones. Para el uso de SimpleRTP es necesario importar la biblioteca y también tener la biblioteca de bitstring.

Existen diferentes niveles de abstracción, desde el nivel más abstracto en el que no es necesario indicar nada sobre el protocolo RTP y solo hay que pasar un fichero MP3, hasta el nivel donde podemos elegir todos los campos de la cabecera RTP, el tamaño del payload del paquete y la cantidad de paquetes que queremos enviar.

A.0.1. Mayor nivel de abstracción

Un ejemplo de uso de SimpleRTP con el mayor nivel de abstracción es el siguiente:

```
cabeceraRTP = simplertp.RtpHeader()
audio = simplertp.RtpPayloadMp3('fichero.mp3')

ip = '127.0.0.1'
port = 33332

simplertp.send_rtp_packet(cabeceraRTP, audio, ip, port)
```

La primera línea crea una variable que contiene un objeto RtpHeader. Crear un objeto sin pasar ningún parámetro deja la cabecera con valores por defecto.

En la segunda también tenemos una variable que contiene un objeto, en este caso de la clase RtpPayloadMp3. Para crear este objeto es necesario pasar el path donde se encuentra el fichero

a enviar.

Las siguientes dos líneas son variables para indicar IP y puerto de destino donde se abrirá el socket UDP, por lo que no dependen directamente de SimpleRTP y simplemente se podrían incluir sus valores en la siguiente línea.

La última línea es donde se envían los paquetes. La función

```
send_rtp_packet
```

necesita mínimo todos los argumentos que hemos definido anteriormente: una objeto RtpHeader, un objeto RtpPayloadMp3 y una IP y puerto destino. Sin más argumentos opcionales, el tamaño de payload y el número de paquetes enviados toma un valor por defecto.

A.0.2. Nivel medio de abstracción

Entrando algo en detalle se pueden empezar a modificar campos de la cabecera RTP y el tamaño de los paquetes:

```
csrc = [2000, 3000, 4000, 5000]
cabeceraRTP = simplertp.RtpHeader(cc=len(csrc))
cabeceraRTP.setCSRC(csrc)

audio = simplertp.RtpPayloadMp3('archivo.mp3')

numeroPaquetesRTP = 0
paquetesMP3porRTP = 2

ip = '127.0.0.1'
port = 33332

simplertp.send_rtp_packet(cabeceraRTP, audio, ip, port, paquetesMP3p
```

En este caso al crear el objeto RtpHeader pasamos un argumento, el de la cuenta de número de fuentes contribuyentes. Se pueden pasar todos los argumentos que se quieran al crear el ob-

jeto dentro de los que existen, representando cada uno a un campo de la cabecera RTP. Cuando creamos el objeto, todos los argumentos que no se pasen toman valores por defecto.

Otra manera de definir campos de la cabecera es mediante funciones específicas para cada campo como setCSRC que define los identificadores de cada fuente contribuyente.

En la llamada a sendrtppacket se pasan dos nuevos argumentos, paquetesMP3porRTP y numeroPaquetesRTP. Esto permite elegir el número de paquetes MP3 que contiene un solo paquete RTP aumentando así el tamaño del payload RTP. El segundo argumento nos permite elegir cuantos paquetes RTP queremos enviar con la llamada. Esto permite no enviar todo el archivo y cambiar parámetros RTP y enviar más paquetes después si así se desea.

A.0.3. Menor nivel de abstracción

En el menor nivel de abstracción que ofrece SimpleRTP podemos modificar todos los campos de la cabecera RTP junto al tamaño del payload del paquete y los paquetes enviados.

```
cabeceraRTP = simplertp.RtpHeader()
csrc = [2000, 3000, 4000, 5000]
cabeceraRTP.set_header(version=2, pad_flag=0, ext_flag=0, cc=4, marke
cabeceraRTP.setCSRC(csrc)
audio = simplertp.RtpPayloadMp3('archivo.mp3')
numeroPaquetesRTP = 0
paquetesMP3porRTP = 2
ip = '127.0.0.1'
port = 33332
simplertp.send_rtp_packet(cabeceraRTP, audio, ip, port, paquetesMP3po
```

En este ejemplo se usa una nueva función llamada setheader. Esta función pertenece al objeto RtpHeader y permite cambiar en una línea todos los campos de la cabecera. Siguen estando las funciones para cambiar solamente campos específicos.

En este nivel por lo tanto se pueden modificar todos los campos de la cabecera RTP de diferentes maneras y controlar el tamaño del payload y el número de paquetes.

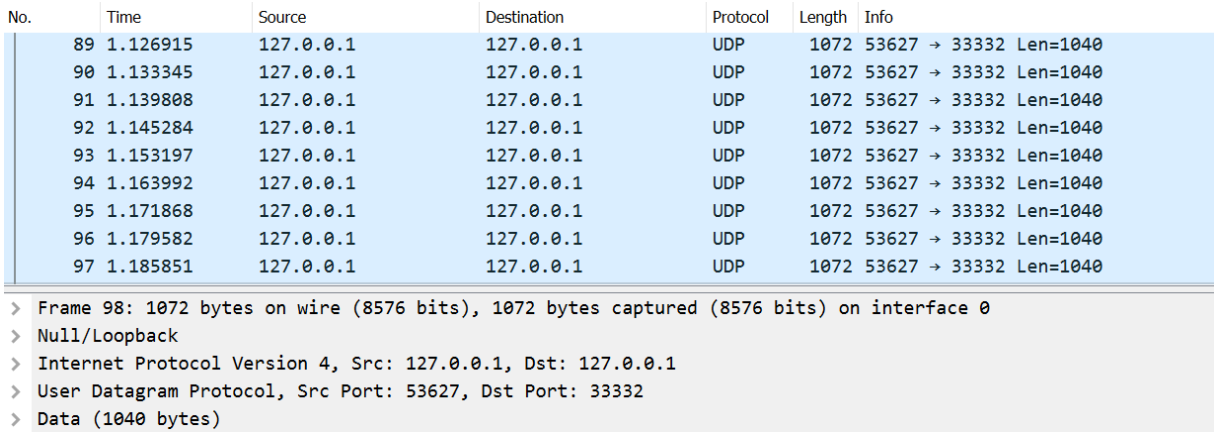


Figura A.1: Wireshark sin analizar paquetes RTP.

A.0.4. Wireshark

Para analizar el tráfico generado se entiende que el usuario tiene un conocimiento básico de Wireshark y sabe capturar paquetes y analizar el contenido de cada uno. Sin embargo, incluyo este apartado para explicar cómo ver tráfico RTP en Wireshark ya que normalmente esta opción está desactivada por defecto y por lo tanto no se podría ver la cabecera RTP con sus contenidos.

Por defecto enviando tráfico RTP veremos lo siguiente en Wireshark:

Como se puede observar, todo el tráfico se ve como UDP (el protocolo que utiliza el socket) pero nada de RTP. Si analizamos un paquete, vamos que contiene un gran payload pero lo clasifica como ”dataz no interpreta nada.

La solución a esto es la siguiente: desde la interfaz de Wireshark, ir a ’Analyze’ en la barra superior y después a ’Enabled protocols’. Ahí debemos buscar RTP y activar ’rtp_udp’ como podemos ver en la figura A.2.

Una vez activado, se podrán ver las capturas como en las imágenes de los apartados anteriores.

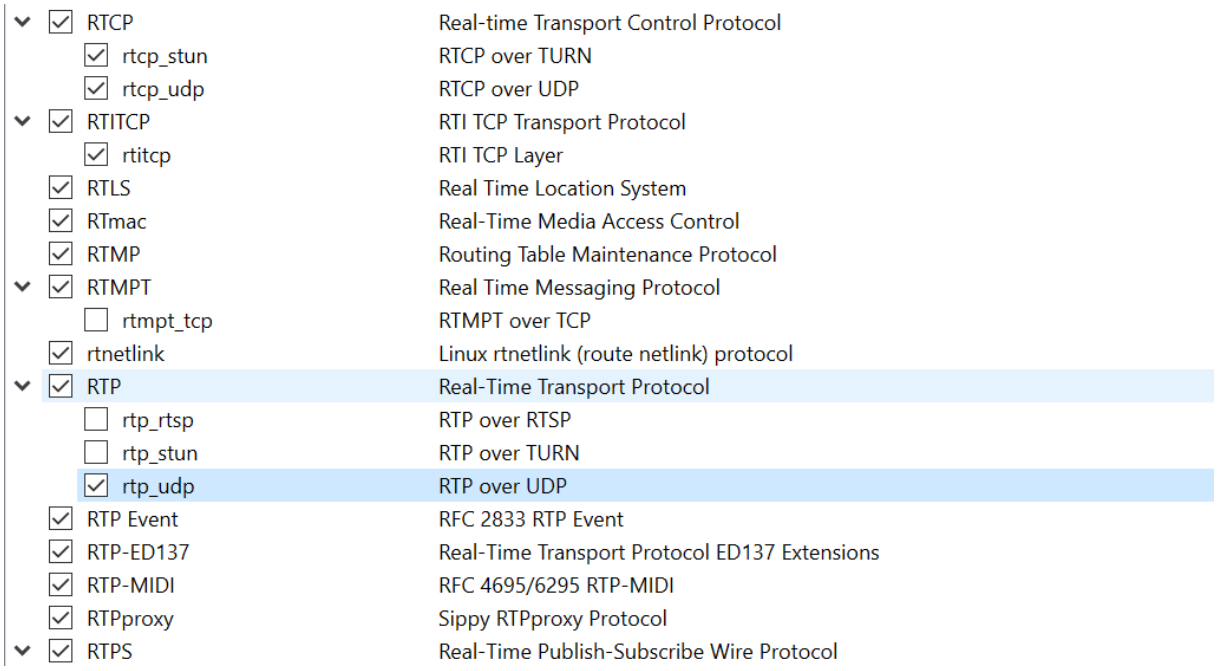


Figura A.2: Casilla a activar para poder ver tráfico RTP.