UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Beamformer acústico

Implementación de un Beamformer acústico mediante array Ad-Hoc de smartphones.

Autor
Sergio Zapata Caparrós

Directores
Antonio Miguel Peinado Herreros
Ángel Manuel Gómez García

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Julio de 2022

Título del proyecto

Subtítulo del proyecto.

Autor
Nombre Apellido1 Apellido2 (alumno)

Directores
Nombre Apellido1 Apellido2 (tutor1)
Nombre Apellido1 Apellido2 (tutor2)

Título del Proyecto: Subtítulo del proyecto

Nombre Apellido1 Apellido2 (alumno)

Palabras clave: palabra_clave1, palabra_clave2, palabra_clave3, ...

Resumen

Poner aquí el resumen.

Project Title: Project Subtitle

First name, Family name (student)

Keywords: Keyword1, Keyword2, Keyword3, ...

Abstract

Write here the abstract in English.

Yo, **Nombre Apellido1 Apellido2**, alumno de la titulación TITULACIÓN de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI XXXXXXXXX, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Nombre Apellido1 Apellido2

Granada a X de mes de 201 .

D. **Nombre Apellido1 Apellido2 (tutor1)**, Profesor del Área de XXXX del Departamento YYYY de la Universidad de Granada.

D. **Nombre Apellido1 Apellido2 (tutor2)**, Profesor del Área de XXXX del Departamento YYYY de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Título del proyecto, Subtítulo del proyecto***, ha sido realizado bajo su supervisión por **Nombre Apellido1 Apellido2 (alumno)**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a X de mes de 201 .

Los directores:

Nombre Apellido1 Apellido2 (tutor1) **Nombre Apellido1 Apellido2 (tutor2)**

Agradecimientos

Poner aquí agradecimientos...

Índice general

Capítulo 1

Introducción

En este capítulo se realizará un fundamento teórico del concepto de "Array Signal Processing" y del método de "Beamforming", además de la utilidad en la realidad de estos términos. A parte, se argumentará la elección del proyecto en concreto y se explicará la distribución del mismo.

1.1. Procesamiento de señales multicanal

El procesamiento de señales multicanal, también conocido como "Array Signal Processing" se interpreta como un procesamiento de las señales en el espacio y en el tiempo.

Conforme se va incrementando en frecuencia, las antenas necesitan unas dimensiones eléctricas mayores y se empiezan a alejar de las geometrías lineales. Para conformar la radiación y llegar a conseguir frentes de onda que puedan generar directividades elevadas y diagramas de radiación concretos, se presenta las llamadas 'Aperturas'. La respuesta procedente de una apertura es direccional, es decir, la señal recibida en la apertura depende de la dirección de llegada (DOA). El diagrama de radiación de una apertura lineal uniforme de longitud 'L' se muestra en la *Figura 1.1.*

Se puede apreciar que actúa como un filtro espacial (fenómeno clave para hacer beamforming), diferenciándose claramente el lóbulo principal de los demás, menos directivos.

Un array de sensores es, básicamente, una apertura conformada por un número de aperturas lineales, simulando de esta manera el comportamiento de una apertura lineal de longitud la suma de las longitudes de los elementos del array más la separación entre ellos. Por así decirlo, un array de sensores es una apertura muestreada en ciertas localizaciones. Un array lineal uniforme simula el comportamiento de una apertura lineal uniforme con una longitud

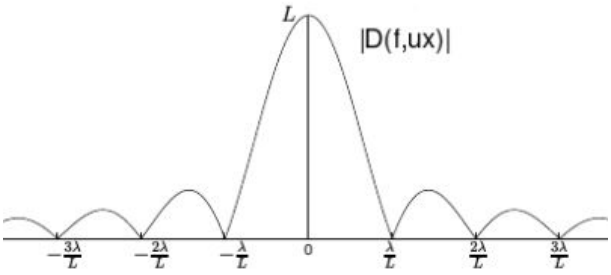

Figura 1.1: Directividad apertura

$L = dN$ donde d se corresponde a la distancia entre elementos y N la longitud de estos. En la Figura 1.2 se puede observar un array lineal uniforme orientado sobre el eje z. Variando la longitud del array con una longitud de onda fija, se pueden lograr diferentes patrones de directividad, como son el "endfire" (con el máximo de directividad en 90^{o}) y el "broadside" (con el máximo de directividad en 0^{o} ó 180^{o}.

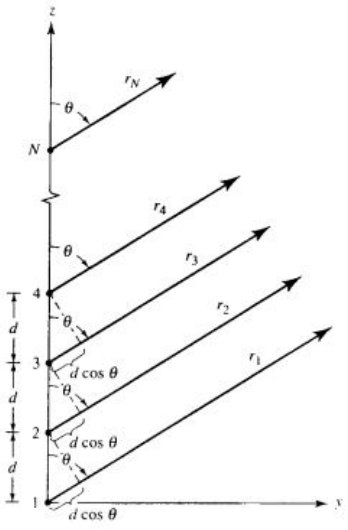

Figura 1.2: Agrupación de elementos

El tipo de array que se va a tratar en el pryecto será un array Ad-Hoc de smartphones. Las agrupaciones de micrófonos consisten en un número de micrófonos los cuales se combinan para filtrar espacialmente las ondas acústicas. La configuración geométrica del array permitirá filtrar las señales deseadas para diversas aplicaciones. Este filtrado espacial sigue normalmente algoritmos de beamforming, los cuales se detallan en la siguiente sección. Sin embargo, para llegar al filtrado espacial, las señales captadas por los

smartphones deben estar sincronizadas. Esto puede ser un problema, ya que el retardo producido desde que el primer móvil recibe la orden de grabar hasta que el último móvil recibe esta orden puede ser determinante. Además, las relojes de los móviles no se encuentran perfectamente sincronizados, por lo que se tendrá que implementar algún algoritmo de sincronización para mitigar estos efectos y llegar a realizar los filtros espaciales adecuadamente.

1.2. Concepto de Beamforming

El concepto de "Beamforming" agrupa las técnicas necesarias para la determinación de filtros específicos, los cuales tienen el fin de obtener un patrón de directividad con una forma y dirección determinadas.

Un "beamformer" se interpreta como un procesador, utilizado conjuntamente con una serie de sensores, para proporcionar una forma versátil de filtrado espacial. El objetivo es estimar la señal procedente de una dirección deseada, en un entorno ruidoso y en presencia de señales de interferencia. El "beamformer" actúa de forma similar a un filtro espacial, separando las señales que han sufrido una superposición en frecuencia, pero que se han originado en diferentes localizaciones espaciales.

El caso que se planteará en este proyecto corresponde a un beamforming acústico. Dentro de este caso, existen dos ramas principales: una relacionada con la extracción de la señal y otra relacionada con la localización de las fuentes sonoras.

1.2.1. Tipos de Beamformers

1.3. Impacto en la actualidad

El procesamiento de señales multicanal tiene múltiples aplicaciones. Algunas de ellas son: radioastronomía, radar, sismología, tomografía, comunicaciones inalámbricas, sonar y tratamiento de audio. Se explicará brevemente la actuación de este tipo de procesamiento para las comunicaciones celulares y para un punto de vista acústico.

1.3.1. Comunicaciones Inalámbricas

Las comunicaciones celulares en la actualidad emplean sectorización por células para proporcionar más frecuencias por área de cobertura.

Sin embargo, esta sectorización no fue suficiente para abarcar el creciente número de usuarios que solicitan recursos de frecuencia, por lo que, en

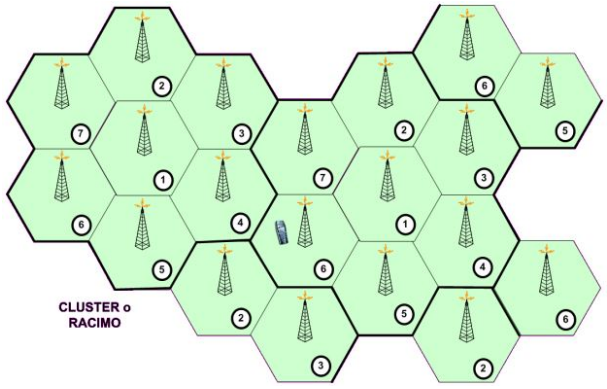

Figura 1.3: Sectorización celular

la última generación estandarizada de comunicaciones móviles (4G), se emplearon técnicas "MIMO" y "Smart Antennas. MIMO es un acrónimo que se refiere a "Multiple Input Multiple Output", por lo que se podrá utilizar una agrupación de antenas tanto en el emisor como en el receptor. Se puede entender mejor el concepto observando el canal del sistema correspondiente a la *Figura 1.2*. En concreto, es un canal MIMO 4x4.

Debido al uso de múltiples antenas, se pueden emplear las técnicas de multiplexado espacial y diversidad espacial. La primera de ellas consigue mejorar la ganancia mediante la transmisión de distintos flujos de información independientes a través de las múltiples antenas. Esto implica un aumento considerable de la tasa de bits respecto a los sistemas con una única antena. Con la diversidad espacial se incrementa la ganancia en codificación mediante la transmisión de secuencias redundantes por varias antenas.

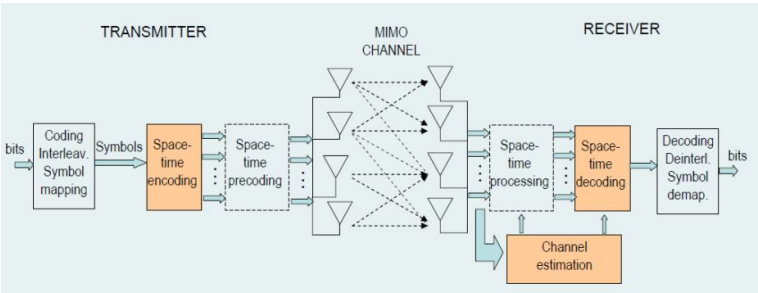

Figura 1.4: Sistema MIMO

En referencia a la introducción de las llamadas "Smart Antennas", son capaces de adaptar el patrón del haz de radiación con el propósito de mejorar la calidad de la señal deseada y minimizar el impacto de la señal interferencia. Esta adaptación implica el uso de beamforming. El beamformer puede ser adaptativo, el cual dirige el haz principal hacia la señal de interés, o con-

mutado, el cual posee una serie de patrones de radiación fijos. Este fenómeto es levemente ilustrado en la *Figura 1.3*.

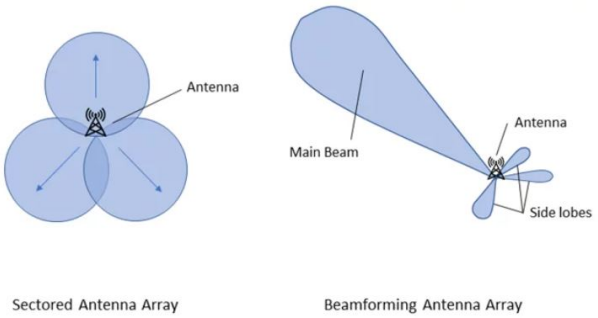

Figura 1.5: Beamforming en comunicaciones inalámbricas

1.3.2. Procesamiento acústico

En referencia un punto de vista acústico y manipulación de audio, se van a exponer dos técnicas distintas. La primera será diarización de locutores y la segunda será la localización de fuentes sonoras.

Diarización

La diarización entre locutores consiste en determinar los instantes de tiempo en los que cada locutor interviene, dada una señal de audio captada con array de micrófonos, como va a ser el caso. Se expone el ejemplo real de una sala de reunión con múltiples locutores y múltiples micrófonos distribuidos en la sala. Para aprovechar la capacidad de un array de múltiples micrófonos, se pueden utilizar técnicas de beamforming, ya explicadas en una sección anterior para realzar la señal de interés.

Antes de ningún proceso que abarque a todos los micrófonos, se debe aplicar un filtro de Wiener a cada canal individual para eliminar el ruido aditivo. Teniendo en cuenta el tiempo de llegada de cada señal y el número total de micrófonos, se implementa un algoritmo que implica correlación cruzada entre señales, obteniendo de esta manera el canal que proporciona una mejor calidad de señal. Tras aplicar beamforming, computando los retrasos entre cada canal, se propone usar el algoritmo de Viterbi, el cual se encarga de hallar el camino o secuencia con mayor probabilidad. El objetivo de este último paso es proporcionar una continuidad a la señal del locutor que está en ese momento hablando, estableciendo el retardo de la señal y filtrando la dirección del haz no deseada.

En la *Figura 1.4* se esquematiza el algoritmo de Viterbi para un ejemplo

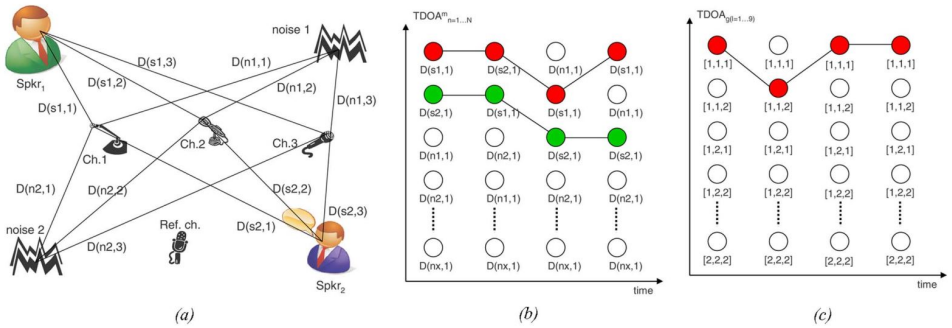

Figura 1.6: Decodificación de Viterbi

de dos locutores, dos fuentes de ruido y micrófonos distribuidos. En el primer paso (b) para cada canal individual, se obtiene los dos mejores caminos en función del tiempo. En el segundo paso (c) se seleccionan los retardos apropiados, considerando todas las posibles combinaciones de todos los canales. Se acaba eligiendo el mejor camino según los datos de distancias y correlación.
Nota: Las columnas de la representación trellis del algoritmo de Viterbi corresponden a la diferencia de tiempos de llegada obtenidos anteriormente.

Localización de fuentes

Técnicas de beamforming acústico son usadas para el propósito de localizar fuentes sonoras. Hay que destacar que, las condiciones ambientales pueden afectar considerablemente a la fiabilidad del beamforming. La reflexión y difracción del las ondas sonoras dan lugar a las llamadas "fuentes fantasma" o *ghost sources*. Estas *ghost sources* aparecen cuando el proceso de beamforming no está perfectamente adecuado para los fenómenos de propagaciones reales del entorno.

- Una aplicación interesante de beamforming es la relacionada con la identificación de fuentes en movimiento. En este caso, se debe considerar el efecto Doppler, por lo que se tendrá que compensar este fenómeno. El beamforming aplicado a objetos en movimiento se suele aplicar en el dominio del tiempo, ya que es más rápido para un número grande de micrófonos. Con las coordenadas del móvil estimadas y la frecuencia Doppler compensada, se procede con un beamforming "delay and sum", visto en secciones anteriores.

- Un campo en el que el beamforming ha supuesto una herramienta de mejora importante es en los experimentos llamados "Túneles de viento" o *Wind tunnels*. Estos experimentos se realizan con el propósito de

analizar el efecto del aire incidente en objetos. Se simula una situación real, enfocando una estructura como puede ser un avión, una aeronave o incluso edificaciones.

Figura 1.7: Configuración del array para un túnel de viento

Las ondas de sonido radiadas por las fuentes aero-acústicas ubicadas en la superficie del modelo, sufren refracción y scattering debido a entorno del experimento, por lo que deberá existir un algoritmo para compensar este fenómeno. La técnica de beamforming juega un papel importante al estimar las repercusiones acústicas que sufre el modelo y evitar posibles averías o incluso conseguir una mejoría en el comportamiento del modelo. En la *Figura 1.5* se muestra el experimento real (parte de la izquierda) y la disposición de los micrófonos (parte de la derecha).

- Por último, mencionar la aplicación de beamforming en interiores. Micrófonos empotrados son normalmente utilizados en entornos de interios, así como cabinas de avión o interiores de automóviles. De esta manera, se consigue identificar la ubicación de las fuentes sonoras y actuar frente a estas.

1.4. Justificación del proyecto

1.5. Distribución del proyecto

1.5.1. Requisitos de realización

1.5.2. Procedimiento

Aquí va una lista de lo que he hecho: servidor, app y algoritmo.

1.5.3. Entornos de trabajo

Java, AndroidStudio, Python...

Capítulo 2

Planteamiento del problema

Capítulo 3

Desarrollo del servidor externo

Tal y como se ha comentado, para crear la comunicación con los móviles y manejar las distintas órdenes correspondientes al objetivo de grabar de forma coherente en cada dispositivo, es necesaria la incorporación de un servidor externo.

3.1. Propósito principal

Lo primero, es aclarar la función que el servidor externo va a desempeñar. Su función será dar órdenes a la aplicación así como las de comienzo y finalización de la grabación. Además, será el encargado de, una vez recibidas las grabaciones procedentes de la aplicación, guardar los ficheros correspondientes. Por supuesto, será el encargado de habilitar un puerto disponible para cada dispositivo que se conecte.

3.1.1. Requisitos funcionales

A continuación se van a enumerar los parámetros funcionales a nivel de software que posee el servidor externo creado.

- Capacidad para manejar un número máximo de 10 clientes.
- La primera conexión de cada cliente se realiza en un mismo puerto común.
- Una vez realizada la primera conexión, el servidor maneja los puertos de conexión fija de los clientes, buscando puertos libres, de manera que tengan una asignación consecutiva (ej: nº puerto 5001, 5002, 5003...)

- Una vez conectados el número de clientes especificado, esperará la confirmación de la aplicación para enviar la orden de "comenzar a grabar"
- Se le podrá especificar la duración de la grabación en milisegundos.
- Una vez acabada la grabación por cada uno de los clientes, se guarda cada fichero de audio en formato de datos brutos (.raw) con la nomenclatura: "Device" + "nº cliente"
- El servidor muestra por pantalla las marcas de tiempo correspondientes al inicio y al final de la grabación de cada dispositivo o cliente.
- Posee una función la cual reproduce una señal de sincronización una vez hayan comenzado a grabar todos los dispositivos.

3.1.2. Requisitos no funcionales

En esta parte se va a exponer la calidad del software en sí mismo, con motivo de evaluar su funcionamiento.

- Es necesario especificar el número de conexiones o número de dispositivos que se van a conectar antes de iniciar el servidor.
- Una vez iniciado el servidor, espera indefinidamente (timeout) hasta que se realicen el número de conexiones especificadas.
- Los tiempos correspondientes a la comunicación entre los diversos dispositivos y el servidor son variables.
- Si a la hora de grabar se produce algún problema en uno de los dispositivos, los ficheros correspondientes a los demás dispositivos se guardarán correctamente, mientras que el fichero correspondiente al dispositivo erróneo, no se almacenará de forma correcta.
- Los tiempos de respuesta del servidor dentro de una zona de cobertura local rondan los 100 milisegundos.
- La comunicación del servidor con los distintos clientes, no se produce de una forma simultánea, sino que genera y recibe información de los dispositivos de uno en uno.
- Las marcas temporales devueltas por el servidor son relativas, ya que dependen de la sincronización de los relojes de cada dispositivo en concreto.
- Si se desea un tiempo de grabación alto, se deberá aumentar el "buffer" correspondiente al amacenamiento de bytes de los archivos de grabación.

3.2. Construcción del servidor

Como ya se ha comentado anteriormente, para la implementación del servidor se ha utilizado el entorno de "NeatBeans" debido a la facilidad que proporciona el lenguaje de programación "java" con los *sockets* para la comunicación cliente-servidor.

Se va a proporcionar una noción respecto al concepto de *socket* y la típica comunicación cliente-servidor. Acto seguido se verá en detalle cada funcionalidad implementada en el servidor .

3.2.1. Noción general y noción de *socket*

Para entender el manejo de los sockets, es necesario conocer la comunicación estándar cliente-servidor.

En una comunicación vía red, la información se desglosa en paquetes. La forma en la que están estructurados estos paquetes, la define el protocolo de transporte utilizado. En el caso que nos concierne, debido a que tenemos el propósito de entablar una comunicación orientada a conexión, se va a utilizar el protocolo TCP (Transfer Control Protocol), complementado con IP (Internet Protocol), formando TCP/IP. Este protocolo utiliza dos piezas claves de identificación: la dirección IP y un número de puerto.

Los términos que hacen referencia a los conceptos de *cliente* y de *servidor* son que el cliente debe iniciar la comunicación, mientras que el servidor espera pasivamente a este primer mensaje por parte del cliente; una vez recibe el servidor el mensaje del cliente, el servidor responde de forma coherente a la petición.

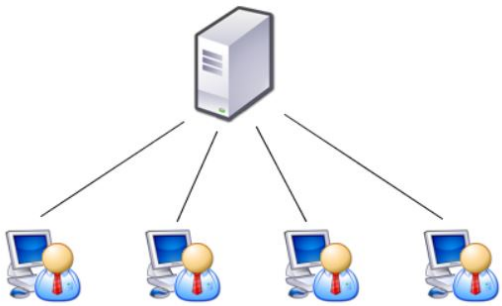

Figura 3.1: cliente-servidor

La distinción *cliente-servidor* es importante porque el cliente necesita conocer la dirección IP y el puerto del servidor, pero no viceversa.

En concepto de socket es abstracto. Se puede entender como un método

el cual permite la comunicación de aplicaciones en una misma red, pudiendo enviar y recibir datos a través del socket. El protocolo TCP/IP permite transmitir flujos de datos y a través de un socket, una aplicación es capaz de alcanzar un puerto disponible de la red. En la Figura 3.2 se esquematiza la posición de los sockets en el entorno de comunicación. El manejo de los

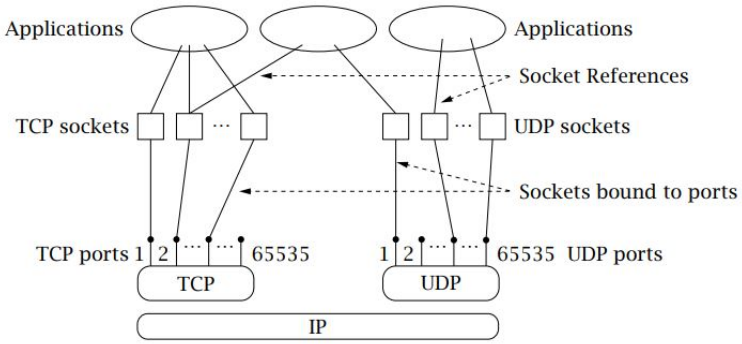

Figura 3.2: TCP sockets

sockets en Java lo podemos encontrar en su propia API.

Server Socket y Socket: Se deben inicializar tanto los sockets correspondientes al cliente como los correspondientes al servidor. El socket del servidor o "Server Socket" hace referencia al puerto al que debe conectar el cliente para mantener la comunicación con el servidor. El socket perteneciente al cliente o "Socket" es el cual se le asigna al cliente por el servidor una vez el cliente haya enviado una request ó petición mediante el Server Socket. El socket del cliente es de utilidad al propio servidor para manejar al cliente correspondiente y para inicializar este último se necesita además del número de puerto, la dirección IP del servidor.

En java, el socket procedente al servidor se inicializa de la siguiente manera:

$$ServerSocket\ server = new\ ServerSocket(Port)$$

El socket correspondiente al cliente:

$$Socket\ client = new\ Socket(IP,\ Port)$$

InputStream y OutputStream: La comunicación se realizará mediante flujos de entrada y de salida o también llamados *InputStream* y *OutputStream*. Para transmitir datos en forma de byte, más en específico,

se utilizan los flujos *DataOutputStream* y *DataInputStream*. Se corresponden al flujo de salida y al flujo de salida respectivamente. Desde el punto de vista del servidor, el *DataInputStream* va del cliente al servidor y el *DataOutputStream* va del servidor al cliente.

La sintaxis en java para inicializar estos puentes es la siguiente:

DataOutputStream output = new
DataOutputStream(client.getOutputStream())

DataInputStream input = new
DataOutputStream(client.getInputStream())

En la salida (*output*) se escribirán bytes para ser enviados y en la entrada (*input*) se leerán los bytes entrantes, valga la redundancia.

Teniendo una idea general de los que es un socket y cómo se utiliza, es trivial que en la aplicación a cual se comunica con el servidor poseerá sockets relacionados con el dispositivo en cuestión, para comunicarse correctamente con el servidor y su correspondiente puerto.

3.2.2. Implementación del servidor

Llegados a este punto, se procede a explicar la arquitectura del propio servidor creado para el proyecto.

Lo esperado es manejar 10 clientes, por lo que el servidor podrá crear y dirigir 10 sockets de clientes distintos. Aún así, se han determinado dos conexiones. La primera conexión será común para todos los dispositivos, es decir, debido a esta primera conexión se ha creado un socket de servidor y otro socket de cliente adicionales, los cuales serán estáticos a todos los clientes. Esto significa que cada dispositivo, por primera vez, se conectará a un mismo puerto. Una vez conectado el cliente a este puerto inicial, el servidor le proporcionará al cliente un nuevo puerto de conexión fija, en el cual se llevará a cabo el intercambio final de datos.

Para la selección de los números de puertos, hay que tener en cuenta cuáles están en uso y cuáles no, para que no haya ningún tipo de problema. Los puertos del 0 hasta el 1023 están reservados para el sistema operativo y los utilizan protocolos como FTP, DNS, SSH... Por lo que, para asegurar una correcta elección de número de puerto, se establece el puerto fijo de la primera conexión en el número de puerto 5000 y los puertos finales serán el 5001, 5002, 5003, ... 5010.

Además, para comprobar la disponibilidad del servidor y el número de puertos abiertos se ha hecho uso del comando *telnet* desde la terminal de Windows. Con este comando se comprueba fácilmente la conexión a un

puerto determinado conociendo la dirección IP de destino. La sintaxis del comando es la siguiente:

telnet[dominio ó dirección IP][puerto]

Si se llega a entablar la conexión, aparecerá una sección de pantalla vacía indicando que el puerto se encuentra disponible y si la conexión no se establece, se emitirá un mensaje de error lo cual indica que el puerto al que se referencia no está disponible o que, simplemente el servidor no se mantiene en escucha en ese puerto específico.

El servidor creado posee varias funciones, las cuales se enumeran y se explican a continuación:

- **Search4port**: Función encargada de devolver una lista de puertos disponibles para iniciar nuevas conexiones. Toma de argumentos de entrada un vector del tipo "booleano" donde se indica los puertos ocupados.
- **Recording**: Función encargada de enviar la orden de grabar a cada cliente. Una vez el servidor se encuentre en esta función, deberá recibir el "ACK" de todos los clientes para lanzar la orden de grabar. Toma de argumentos de entrada los sockets correspondientes al número de clientes.
- **Delay**: Esta función es la manejadora del tiempo de grabación. Simplemente manda a reposar el hilo principal durante un tiempo determinado, introducido en milisegundos.
- **stopRecording**: Como su propio nombre indica, es la función relacionada con el envío de la orden de parar la grabación. Similar a *Recording* con la diferencia de que se envía la orden opuesta y que el servidor no espera ningún "ACK" por parte de los clientes para emitir la orden correspondiente.
- **getTimeStamp**: Función que lee la marca de tiempo devuelta por cada móvil. Como argumento de entrada toma el socket de cada cliente, ya que la marca temporal de cada móvil proviene de la aplicación.
- **Time**: Si fuese necesario devolver una marca temporal "absoluta", tomada con el reloj del servidor, se puede llamar a esta función cuando se desee sin argumentos de entrada. Puede ser útil para comparar las demás marcas de tiempo "relativas" obtenidas para cada dispositivo.
- **serverImpulse**: Esta función se encarga de emitir un chirp inicial, justo cuando todos los dispositivos están grabando. De esta manera, con la señal inicial captada, se podrá llegar a un algoritmo de sincronización entre las grabaciones efectivo.

- **playChirp**: Para realizar un método de sincronización distinto al comentado en la función anterior, se puede llamar a esta función para emitir una orden de reproducir una señal "chirp" en cada uno de los móviles o clientes mientras se encuentran en estado activo de grabación, con el propósito obtener las señales grabadas sincronizadas.

- **saveFiles**: Por último, se llama a esta función. Su principal cometido es recibir los bytes procedentes de cada una de las grabaciones, almacenarlos en un buffer temporal, y guardarlos en la memoria de la máquina en la que se está ejecutando el servidor. Se debe ser coherente con el tiempo de la grabación y el tamaño del buffer temporal. Guarda las grabaciones como archivos de datos en bruto (.raw). Toma de argumentos de entrada el socket de cada cliente.

3.3. Comunicación

En esta sección se va a documentar en profundidad la comunicación producida entre el servidor y cada uno de los dispositivos.

Antes de nada, se debe tener claro el orden de procedimiento del servidor. Una vez iniciado el servidor, se presentan los siguientes flujos comunicativos:

1. El servidor se queda esperando hasta que recibe una petición de conexión de un cliente al puerto 5000.

2. Una vez aceptada esta primera conexión, el servidor busca un puerto libre, mediante la función *Search4port* y se lo devuelve al cliente para que realice una nueva conexión.

3. El cliente envía una nueva *request* al puerto indicado por el servidor. Al aceptar el servidor esta nueva conexión, el cliente finalmente se encuentra en el puerto adecuado para realizar la comunicación completa.

4. Ya conectados el número de clientes especificado, el dispositivo deberá enviar un mensaje de "READY", para dar a entender al servidor que está listo para grabar.

5. El servidor emite la orden de grabar ("START") y espera a que cada uno de los móviles le hagan llegar un "ACK" de confirmación por empezar a grabar. Adicionalmente, los móviles transmiten al servidor la marca temporal correspondiente al inicio de la grabación.

6. Pasados los segundos de grabación establecidos, el servidor emite la orden de parar la grabación ("STOP") y cada dispositivo devuelve de nuevo una marca temporal, relacionada con el final de la grabación.

7. Por último, cada smartphone envía al servidor el fichero de grabación resultante.

Con propósito a comprobar este flujo de transporte enumerado, se hará uso de la herramienta *WireShark*. El principal objetivo de esta herramienta es analizar el tráfico de una red. A parte, es ideal para estudiar cualquier tipo de comunicaciones y problemas de red.

Cabe anotar que, tras diversas pruebas, se ha obtenido un resultado más constante respecto a los tiempos de la red si se crea un punto de acceso inalámbrico en la máquina en la que corre el servidor y que todos los smartphones que deseen interactuar con el servidor, deberán conectarse a este punto de acceso creado. La comunicación mediante la Wi-Fi local presentaba muchos retardos inesperados, debido a que diversos dispositivos están conectados a la red, por lo que el tráfico de paquetes será mayor y, lo más desfavorable, con más rango de aleatoriedad. Además, disponiendo de un punto de acceso privado, el tráfico de la red se reducirá considerablemente a la hora de visualizar el flujo de transporte.

Mencionado esto, se presenta una captura del tráfico correspondiente al punto de acceso inalámbrico en la *Figura 3.3*. Se ha filtrado el tráfico de manera que solo se visualice el protocolo TCP.

Con los datos proporcionados por el tráfico de red, se traza un diagrama UML para una mejor visualización de la comunicación entre el servidor y un cliente.

Capítulo 4

Desarrollo de la aplicación

Tal y como se ha comentado anteriormente, para la comunicación entre el servidor y los respectivos smartphones y para que cada uno ejecute la grabación correctamente, es necesario la introducción de una aplicación Android en cada uno de los dispositivos. De esta manera se consigue la interconexión que se desea.

4.1. Introducción a la aplicación

La aplicación ha sido desarrollada en *Android Studio*. Se le ha especificado el nombre de *BeamRec*.

El propósito de la aplicación es cumplir los requisitos necesarios, esto es la conexión con el servidor, la correcta grabación y la debida transmisión del fichero de grabación. La interfaz de la aplicación está pensada para que sea lo más intuitiva posible, para conseguir únicamente la meta del proyecto que se está tratando.

Nada más abrir la aplicación, la interfaz resultante se muestra en la *Figura 4.1.* Como se puede apreciar, directamente proporciona la opción al usuario de conectar el dispositivo con el servidor externo. Una vez se pulse el botón de "CONNECT", el cliente enviará una petición de conexión al servidor especificado.

Al pulsar el botón de "CONNECT" y habiéndose conectado el dispositivo exitosamente al servidor, la siguiente interfaz o *activity* que el usuario visualiza se observa en la *Figura 4.2.*

En este caso, la aplicación únicamente proporciona la opción de grabar. Este botón se deberá pulsar una vez todos los móviles que se desean conectar al servidor hayan conseguido una conexión exitosa. Cuando se desee comenzar a grabar, una vez pulsado el botón, el cliente envía el correspon-

Figura 4.1: Interfaz de conexión

Figura 4.2: Interfaz de grabación

diente mensaje de que está listo para grabar hacia el servidor. Este último recibe el mensaje y emite la orden de grabar. Si se desea repetir la grabación, se deberá realizar desde el principio el proceso de conexión con el servidor externo. Será necesario que cada uno de los móviles estén conectados a la misma red a la que está conectado el servidor.

4.2. Funcionalidades

En esta parte se va a explicar las distintas funciones internas de la propia aplicación.

En total, se han creado 3 clases java en Android Studio:

- **ManActivity**: Es la clase principal. En ella se encuentra el diseño de la primera interfaz mostrada en la *Figura 4.1*. Al pulsar el usuario el botón, se hará una llamada a la siguiente clase con su nueva interfaz.

- **RecordClient**: Contiene el diseño de la segunda interfaz de la aplicación (*Figura 4.2*). Esta clase es la encargada de mantener la conexión con el servidor dada una dirección IP específica y el puerto común, el cual se ha determinado como el puerto número 5000 en la sección del servidor. Realiza todo el proceso de doble conexión al servidor en segundo plano y, una vez conectado el dispositivo al servidor, se cargará la interfaz correspondiente a la acción de grabar.

  Al pulsar el usuario el botón de "RECORD", se mandará el mensaje de "READY" al servidor y se esperará a la orden de empezar la grabación emitida por el servidor. La comunicación con el servidor desde la aplicación se realiza de manera similar que en el servidor: con el uso de *Sockets* y flujos de datos *DataInputStream* y *DataOutputStream*. Cuando la aplicación recibe la orden de grabar por el servidor, se hace la llamada a la clase *Recorder* y empieza la grabación en un nuevo hilo. En el hilo principal se procede al envío del "ACK" y de la marca de tiempo correspondiente al comienzo de la grabación por parte del dispositivo al servidor y se mantiene en espera hasta recibir la orden correspondiente a parar la grabación.

  Una vez recibida la orden de "STOP" procedente del servidor para parar la grabación, se hace la llamada correspondiente a la clase Recorder para dejar de grabar y se envía la marca de tiempo de finalización. Acto seguido, se procede a la manipulación del archivo. Mediante las clases *BufferedInputStream* y *BufferedInputStream* se almacenará la grabación en un buffer momentáneo que será enviado directamente al servidor.

Cabe destacar que, en esta clase también se definen los permisos necesarios para la correcta ejecución de la aplicación en el dispositivo Android. Estos son los permisos de utilización del micrófono y de la grabadora del smartphone.

- **Recorder**: Esta clase, como se ha mencionado en *RecordClient*, es la responsable de la propia acción de grabar. Se ejecuta en segundo plano y es capaz de ejecutar una grabación con una frecuencia de muestreo de 44100 Hz, la cual es la que soporta la mayoría de dispositivos actualmente, en un canal monofónico y una codificación del tipo PCM (Modulación de Pulsos Codificados) de 16 bits para que, de esta manera, no exista compresión alguna en el audio.

  Además, se implementa la desactivación del control de ganancia automático ó *AGC* y la cancelación de eco ó *AEC*, lo cual podrían ser factores perjudiciales para el propósito del proyecto, ya que se modificaría la señal automáticamente complicando procesos posteriores como es el de sincronización o beamforming.

  La clase *Recorder* proporciona dos métodos, uno de comienzo de la grabación, en el cual se irán almacenando paquetes de bytes en un buffer temporal, y el método relacionado con terminar la grabación.

Capítulo 5

Sincronización

Capítulo 6

Implementación del algoritmo de Beamforming

Capítulo 7

Análisis del modelo

Capítulo 8

Perspectiva de futuro

Aquí poner lo del 5G, massive MIMO etc Y ALGUNA COSA MÁS.

Capítulo 9

Conclusiones

Capítulo 10

Bibliografía

Capítulo 11

Apéndice