Device to Device streaming en dispositivos móviles

Iván Gulyk y Noel José Algora Igual

FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo de fin de grado del Grado en Ingeniería Informática

Director: Simon Pickin

Curso 2018/2019

Este documento ha sido maquetado con LaTeX

Agradecimientos

Queremos agradecer toda la ayuda, apoyo y ánimos recibidos por parte de nuestros familiares, compañeros y amigos.

También agradecemos a todos los profesores que nos dieron formación y nos proporcionaron conocimientos día a día a lo largo de nuestra etapa universitaria.

Especial agradecimiento a nuestro tutor, Simon Pickin, por ofrecernos este proyecto y por toda la colaboración y apoyo que nos ha dado en todo momento.

Por último, queremos dar las gracias al creador de la librería libstreaming, que nos ha ahorrado mucho trabajo, y sin la cual posiblemente no habríamos podido llevar a cabo este proyecto.

Palabras Clave

- WiFI Direct
- Android
- Libstreaming
- Streaming
- RTSP
- Ad hoc
- P2P

Keywords

- WiFi Direct
- Android
- Libstreaming
- Streaming
- RTSP
- Ad hoc
- P2P

Índice

Resumen

El objetivo principal de este proyecto ha sido desarrollar una aplicación para dispositivos móviles capaz de crear una red distribuida, y de retransmitir vídeo en streaming directamente desde la cámara y micrófono de uno u varios dispositivos de la red. La aplicación propuesta debía poder crear una red infinita por interconexión de dispositivos cercanos, todo esto sin usar la infraestructura de red de los operadores de telecomunicaciones.

En el transcurso del proyecto mostramos que, debido a las limitaciones en la implementación de la tecnología disponible, actualmente no es posible lograr del todo este objetivo. Por esta razón, desarrollamos una aplicación con funcionalidad reducida, pero capaz de transmitir y reproducir streaming a través de una pequeña red distribuida de dispositivos conectados mediante la tecnología WiFi Direct. La aplicación permite distribuir el streaming por medio de *multihopping*, de tal forma que vaya pasando de un dispositivo a otro en cadena hasta llegar a su destino. Sin embargo el estado actual de la tecnología implementada en los dispositivos móviles solo nos permite realizar dos saltos, es decir, conectar tres dispositivos en cadena.

El código fuente de nuestro proyecto esta alojado en un repositorio publico en GitHub en la siguiente URL:

https://github.com/ivangulyk/TFG

Summary

The main objective of this project has been to develop an application for mobile devices able to create a distributed network, and of retransmitting streaming video directly from the camera and microphone of one or several devices in the network. The proposed application should be able to create an infinite network by interconnecting nearby devices, all without using the network infrastructure of telecommunication operators.

In the course of the project we showed that, due to limitations in the implementation of the available technology, it is currently not possible to achieve this objective in its entirety. For this reason, we developed an application with reduced functionality, but capable of transmitting and playing streaming through a small distributed network of connected devices using WiFi Direct technology. The application allows to distribute the streaming by means of textit multihopping, in such a way that it goes from one device to another in chain until reaching its destination. However, the current state of technology implemented in mobile devices only allows us to perform two jumps, that is, connect three devices in chain.

The source code of our project is hosted in a public repository on GitHub at the following URL:

https://github.com/ivangulyk/TFG

1 Introducción

1.1 Motivación

Hoy en día, en la mayoría de los casos, dependemos totalmente de la conexión a Internet a la hora de conectarnos con otros dispositivos a través de nuestras aplicaciones. En este proyecto buscamos investigar los diferentes usos que se pueden conseguir con aplicaciones en las que tengamos la posibilidad de conectar nuestros dispositivos a otros sin la necesidad de tener acceso a ninguna infraestructura de red física, en particular, sin usar la red de los operadores de telecomunicaciones.

Simplemente crear aplicaciones que sean capaces de conectarse con otro dispositivo cercano ya es algo realmente ventajoso, pero mas allá de conectar dos dispositivos cercanos entre sí, buscamos investigar cómo se podría crear una red *ad hoc* mediante esas conexiones. Una red *ad hoc* no depende de infraestructura externa como routers o puntos de acceso, y es una red descentralizada. En una red como esta, los dispositivos se conectan unos a otros de tal forma que nuevos dispositivos pueden unirse a la red creando una conexión con cualquier dispositivo que ya se encuentre en ella. Nuestra aplicacion crearia una red *ad hoc* en la que los datos pasan de dispositivo a dispositivo como si de una cadena se tratase hasta llegar a su destino. Mas concretamente se crearía una red WANET (Wireless Ad-hoc Network) o MANET (Mobile Ad-hoc Network), al ser esta una red ad hoc sobre dispositivos móviles con tecnologías inalámbricas.

En la actualidad prácticamente todo el mundo dispone de un dispositivo móvil y, teóricamente, en zonas con una cierta densidad de población se debería poder crear redes interconectando todos los dispositivos.

Con los nuevos avances de la tecnología – como son las tecnologías WiFi, que se encuentran en la mayoría de dispositivos – empezamos a disponer de nuevos medios para establecer este tipo de conexiones, abriéndonos un nuevo abanico de posibilidades. Creemos que es un área muy poco investigada para todo el potencial que nos ofrece como pueden ser este tipo de redes.

Mas específicamente, lo que buscamos es poder retransmitir streamings de vídeo que puedan ser capturados en directo por medio de la cámara y el micrófono de nuestro dispositivo, así como de vídeos ya grabados previamente, y distribuir estos streaming a través de nuestra propia red *ad hoc*. Algunos problemas que nos encontramos en una red MANET son el uso de técnicas de routing si se necesita establecer conexión directa entre dos dispositivos; pero para los fines planteados, se utilizaran técnicas de retransmisión de tipo *flooding* y *multihopping*, evitando así esos problemas.

Queremos realizar la retransmisión en streaming, en vez de una simple transferencia de ficheros de vídeo, dado que para algunos de los casos de uso que tenemos en mente se puede querer emitir el vídeo lo antes posible, y a veces no se podría enviar un fichero con el vídeo al finalizar la grabación. Secundariamente, como también puede ser útil, queremos que se puedan transmitir archivos.

Nos encontramos ligados a la conexión de Internet, pero en caso de una catástrofe natural se puede perder la infraestructura, y por tanto las conexiones a las redes convencionales. Se podría formar una red *ad hoc* para pasar vídeos y/o mensajes de ayuda de un dispositivo a otro hasta encontrar un dispositivo que si disponga de acceso a Internet, y de esta forma los datos enviados desde una zona sin conexión se publicarían en Internet y llegarían a su destino.

Otro de los posibles casos de uso para una aplicación como la que proponemos podría ser el dar la posibilidad de transmitir vídeos a dispositivos lejanos en países en los que se cometen crímenes contra los derechos humanos, de tal forma que el vídeo pasase de un dispositivo a otro saltándose el control y censura que países como estos establecen en las redes convencionales, además de dar la posibilidad de ocultar el origen de la transmisión, de tal forma que no se pueda identificar al denunciante. Por este motivo, además, la aplicación no deberá dejar trazas de los vídeos, ni en los móviles emisores ni en los que lo redistribuyen. Si la red es capaz de cruzar la zona de censura los vídeos podrían ser publicados en Internet con ayuda de organizaciones como Witness [1] que se dedican a promover la creación y custodia de vídeos de este tipo.

Pero habría que tener un especial cuidado para este caso, y en general, con el tema del "deepfake", ya que la inteligencia artificial está lo suficientemente desarrollada como para poder crear con su ayuda [2] vídeos falsos, la difusión de los cuales podría provocar escándalos y conflictos. Se está investigando sobre el uso tecnologías que puedan identificar deepfakes [3], pero esto es algo muy difícil.

Una tecnología como la que pensamos desarrollar es algo bastante novedoso, y como todo lo novedoso, en un principio existen muchos factores a tener en cuenta, muchos problemas que resolver, y agujeros de seguridad que tapar. La red sobre la que pensamos hacer streaming de vídeo es una red *ad hoc*, que no necesita utilizar ningún tipo de infraestructura de red externa (como puede ser un router o la infraestructura de los operadores de telecomunicaciones), y por tanto el contenido que pasa por ella no es controlado por nadie más que el emisor del mismo; por consiguiente, el propio emisor debe hacerse responsable de lo que emite en directo. Aquí nos surge un factor a tener en cuenta, que son las personas mal intencionadas, las cuales podrían transmitir cualquier tipo de vídeo, con contenido inapropi-

ado, como podría ser vídeos de pornografía infantil.

Con los avances de la tecnología nos encontramos también con el problema de que los vídeos pueden modificarse en transito, es decir, en la medida que pasan de un dispositivo a otro, y no únicamente desde el emisor, por lo que además es necesario hacer un filtrado en tiempo real. Esto junto con la retransmisión de contenido inapropiado es de vital importancia en el caso de uso por los defensores de los derechos humanos, dado que se pueden intentar utilizar este tipo de ataques para intentar desacreditar la red por la que se están distribuyendo, así como a los usuarios de esta.

1.2 Objetivos

El objetivo principal de este proyecto es diseñar e implementar una aplicación para dispositivos móviles capaz de conectar los dispositivos entre ellos sin utilizar la infraestructura de red de los operadores de telecomunicaciones. Una vez creado el enlace entre dispositivos, la aplicación debe poder transmitir vídeo en directo desde un móvil a otro, que debe a su vez ser capaz de recibir ese vídeo –y reproducirlo, si el usuario lo quiere– siendo capaz al mismo tiempo que recibe el vídeo de retransmitirlo a otros dispositivos a los que esté conectado, actuando de esta manera como un nodo transitorio entre dispositivos no conectados directamente entre ellos.

Para poder llevar a cabo el objetivo principal se debe:

1. Investigar sobre las tecnologías existentes tanto en el área de conexión entre dispositivos móviles –con WiFi Direct o WiFi Aware como posibles candidatos– como en el área de transmisión de vídeo en directo y su reproducción en móviles.
2. Escoger las tecnologías más adecuadas y aplicar los conocimientos obtenidos sobre esas tecnologías para desarrollar un proyecto práctico.
3. Diseñar e implementar la aplicación.
4. Detectar y estudiar los posibles problemas que podrían surgir con la funcionalidad que proporciona nuestra aplicación, como podría ser la transmisión de vídeos con contenido inapropiado. Proponer soluciones efectivas a estos problemas.

1.3 Plan de trabajo

Nombre de la tarea	Fecha de Inicio	Fecha final
Investigación	**20/07/18**	**30/10/18**
Tecnologías de conexión	20/07/18	24/08/18
Apps parecidas	25/08/18	28/08/18
Tecnologías de streaming	01/09/18	28/09/18
Tecnología de transporte	29/09/18	30/10/18
Desarrollo	**01/11/18**	**29/03/19**
Implementar la conexión de dispositivos	01/11/18	20/12/18
Generar datos(cámara y micrófono)	21/12/18	18/01/19
Implementar envío de datos	19/01/19	20/02/19
Implementar recepción, reproducción y reenvío de datos	21/02/19	29/03/19
Validación	**01/04/19**	**11/04/19**
Realizar pruebas unitarias	01/04/19	10/04/19
Pruebas con usuarios	11/04/19	11/04/19
Redacción de la memoria	**20/01/19**	**20/05/19**

Figure 1: Planificación

Para llevar a cabo el proyecto y conseguir los objetivos vamos a dividir la planificación en 4 grandes bloques, cada uno dividido en tareas, como se muestra en la Figura 3

A partir del esquema anterior hemos generado un diagrama de Gantt (Figura 4)

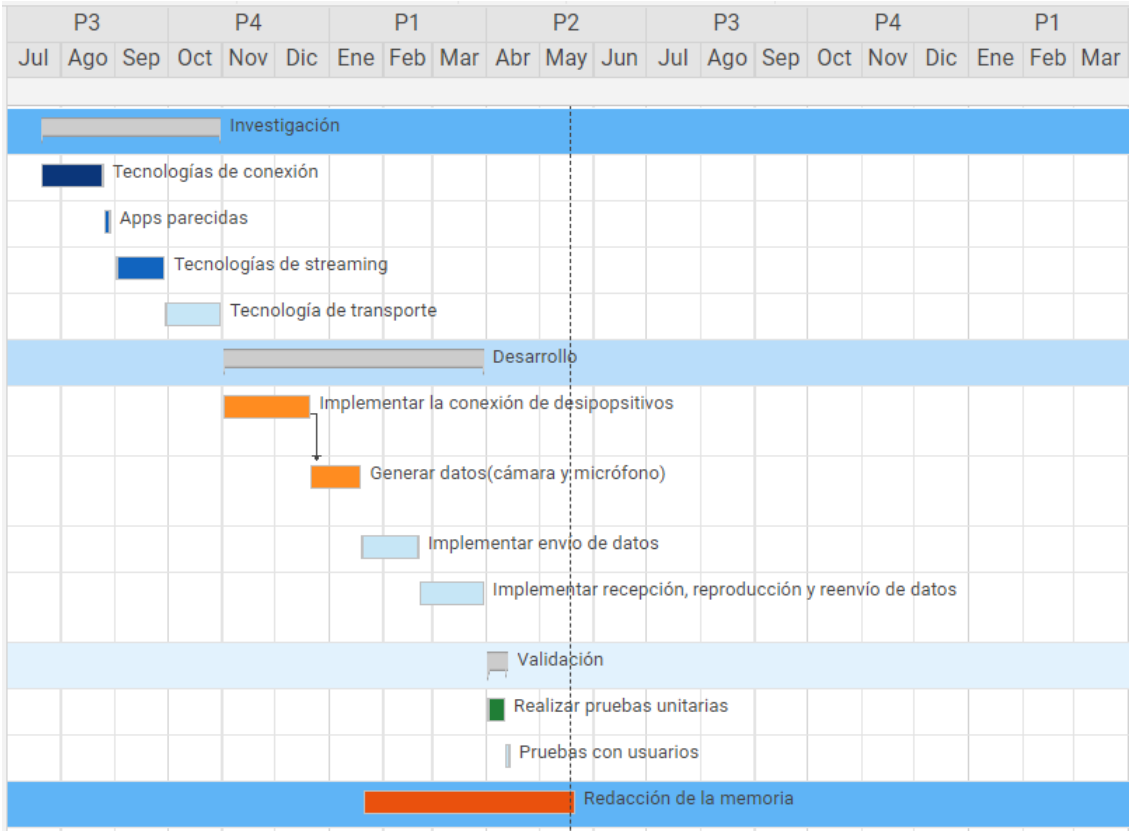

Figure 2: Diagrama de Gantt

1 Introduction

1.1 Motivation

Nowadays, in most cases, we depend totally on the Internet connection when connecting with other devices through our applications. In this project we seek to investigate the different uses that can be achieved with applications in which we have the possibility of connecting our devices to those of others without the need to have access to any physical network infrastructure, in particular, without using the network of telecommunication operators.

Even just to create an application capable of connecting to another nearby device is already really advantageous, but beyond connecting together two close devices, we seek to investigate how to create an *ad hoc* network through these connections. An *ad hoc* network does not depend on external infrastructure such as routers or access points, and is a decentralized network. In a network like this, devices connect to each other in such a way that new devices can join the network by creating a connection with any device already in it. Our application would create a *ad hoc* network in which the data passes from device to device, as if it were a chain, until reaching its destination. More specifically, a WANET network (Wireless Ad-hoc Network) or MANET (Mobile Ad-hoc Network) would be created, since this is an *ad hoc* network on mobile devices with wireless technologies.

Currently virtually everyone has a mobile device and, theoretically, in areas with a certain population density it should be able to create networks by interconnecting all devices.

With the new advances in technology – such as WiFi technologies, which are found in most devices – we began to have new means to establish this type of connection, opening a new range of possibilities. We believe that it is an area that has been little investigated in spite of the potential offered by such networks.

More specifically, what we seek is to be able to retransmit video streams that can be captured live through the camera and microphone of our device, as well as previously recorded videos, and distribute these streaming through our own network *ad hoc*. Some problems that we find in a MANET network are the use of routing techniques if we need to establish a direct connection between two devices; but for the proposed goals, relay techniques of type *flooding* and *multihopping* will be used, thus avoiding those problems.

We want to make a broadcast in streaming, instead of a simple transfer

of video files, since for some of the use cases that we have in mind you may want to broadcast the video as soon as possible, and sometimes you can not send a file with the video at the end of the recording. Secondarily, as it can also be useful, we want to be able to transmit files.

We are linked to the Internet connection, but in case of a natural catastrophe you can lose the infrastructure, and therefore the connection to conventional networks. An *ad hoc* network could be formed to pass videos and / or help messages from one device to another, until a device is found that has access to the Internet, and in this way the data sent from an area without connection will be published on the Internet and reach their destination.

Another possible use case for an application like the one we propose could be to give the possibility of transmitting videos to distant devices in countries where human rights crimes are committed, in such a way that the video would pass from one device to another bypassing the control and censorship that countries like these establish in conventional networks, in addition to giving the possibility of hiding the origin of the transmission, in such a way that the complainant can not be identified. For this reason, in addition, the application should not leave traces of the videos, neither in the issuing mobiles nor in those that redistribute them. If the network is able to cross the censorship zone, the videos could be published on the Internet with the help of organizations such as Witness [1] that are dedicated to promoting the creation and custody of videos of this type.

But we should take special care for this case, and in general, with the topic of "deepfake", since artificial intelligence is sufficiently developed to be able to create with its help false videos, the diffusion of which could lead to scandals and conflicts. The use of technologies that can identify deepfakes [3] is being investigated, but this is very difficult issue.

A technology like the one we plan to develop is something quite new and, like every novelty, there are at the beginning many factors to be taken into account, many problems to be solved, and security holes to be covered. The network on which we intend to make video streaming is an *ad hoc* network, which does not need to use any type of external network infrastructure (such as a router or the infrastructure of telecommunications operators), and therefore the content that passes through it is not controlled by anyone other than the issuer; therefore, the issuer itself must be responsible for what it broadcasts live. Here there is a factor we have to take into account, the malicious people, which could transmit any type of video, with inappropriate content, such as videos of child pornography.

With the advances in technology, we are also faced with the problem that videos can be modified in transit, that is, as they pass from one de-

vice to another, and not only from the sender, so it is also necessary to make a filtering in real time. This, together with the transmission of inappropriate content, is of vital importance in the case of its use by human rights defenders, given that it is possible to try to use this type of attacks with the intention of discrediting the network through which they are being distributed, as well as the users themselves.

1.2 Goals

The main objective of this project is to design and implement an application for mobile devices capable of connecting the devices between them without using the network infrastructure of telecommunication operators. Once the link between devices is created, the application must be able to transmit live video from one mobile to another, which must in turn be able to receive that video –and play it, if the user wants it– and able, at the same tiem, to relay it to other devices to which it is connected, acting in this way as a transient node between devices not directly connected between them.

To be able to carry out the main objective you must:

1. Investigate existing technologies both in the area of connection between mobile devices – with WiFi Direct or WiFi Aware as possible candidates – as in the area of live video transmission and its reproduction on mobile phones.
2. Choose the most appropriate technologies, and apply the knowledge obtained on these technologies to develop a practical project.
3. Design and implement the application.
4. Detect and study the possible problems that could arise with the functionality provided by our application, such as the transmission of videos with inappropriate content. And to propose effective solutions to these problems.

1.3 Workplan

To carry out the project and achieve the objectives we will divide the planning into 4 large blocks each with their tasks as shown in the Figure 3

From the previous scheme we have generated a Gantt chart (Figure 4)

Nombre de la tarea	Fecha de Inicio	Fecha final
Investigación	**20/07/18**	**30/10/18**
Tecnologías de conexión	20/07/18	24/08/18
Apps parecidas	25/08/18	28/08/18
Tecnologías de streaming	01/09/18	28/09/18
Tecnología de transporte	29/09/18	30/10/18
Desarrollo	**01/11/18**	**29/03/19**
Implementar la conexión de dispositivos	01/11/18	20/12/18
Generar datos(cámara y micrófono)	21/12/18	18/01/19
Implementar envío de datos	19/01/19	20/02/19
Implementar recepción, reproducción y reenvío de datos	21/02/19	29/03/19
Validación	**01/04/19**	**11/04/19**
Realizar pruebas unitarias	01/04/19	10/04/19
Pruebas con usuarios	11/04/19	11/04/19
Redacción de la memoria	**20/01/19**	**20/05/19**

Figure 3: Planificación

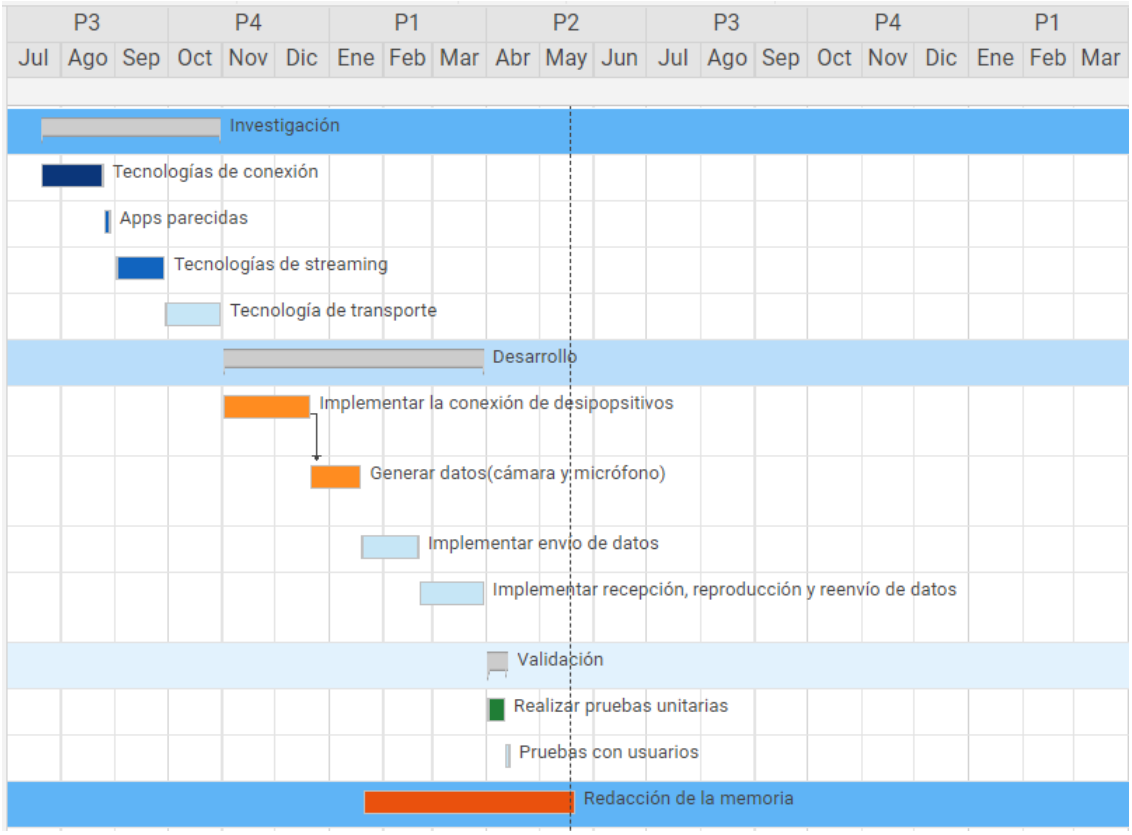

Figure 4: Diagrama de Gantt

2 Antecedentes

2.1 Streaming

Streaming consiste en crear un flujo sin interrupción de contenido multimedia través de una red de manera que un usuario puede reproducir lo retransmitido a la vez que se descarga. El streaming puede ser "live", en directo, o de audio y/o vídeo ya existentes. Normalmente cuando se retransmite audio y vídeo a la vez, estos se transmiten por separado en dos flujos distintos los cuales deben ir sincronizados por lo que requiere de protocolos que para controlarlo.

El streaming requiere de una conexión por lo menos de igual ancho de banda que la tasa de bits de la transmisión del servicio. También existe streaming de tasa de bits adaptable que detecta el ancho de banda de la conexión y la capacidad de la CPU de un usuario y ajusta la calidad de transmisión de multimedia en función de ello.

2.2 WiMAX

WiMax es una familia de protocolos de los niveles físico y enlace de datos que permite conectarse a una red a través de ondas electromagnéticas en las frecuencias de 2,5 a 5,8 GHz que pueden tener una cobertura de hasta 80 km.

Por su cobertura y rendimiento se suele usar conexiones en donde no es posible tener acceso a la red mediante redes tradicionales de pares de cobre o fibra óptica pero es necesaria una infraestructura bastante grande.

2.3 Bluetooth

El Bluetooth [4] es una tecnología que se usa para la transmisión inalámbrica de datos entre diferentes dispositivos que se encuentran a corta distancia, dentro de un radio de alcance que suele ser de diez metros. Consta de un conjunto de protocolos de nivel físico y de enlace pero también de niveles mas altos.

Esta tecnología transmite inalámbricamente datos a través de ondas de radio en las frecuencias de 2,4 GHz. Para ello, hace uso de las Redes Inalámbricas de Área Personal.

Los equipos deben encontrarse dentro de un radio de alcance corto, aunque puede variar en función del aparato. Los dispositivos Bluetooth se clasifican de la siguiente manera:

- Clase 1. Tienen una potencia máxima permitida de 100 mW y un alcance de 100 metros.
- Clase 2. Se caracterizan por tener un radio de alcance de entre 5 y 10 metros, dado que su potencia máxima permitida es de 2,5 mW. Estos son los más habituales.
- Clase 3. Cuentan con una potencia máxima de 1 mW y un alcance de, tan sólo, un metro.

Otro de los inconvenientes mas destacados de Bluetooth es el ancho de banda que tiene que es de unos 50 Mbps en Bluetooth 5.0, el más moderno a día de hoy. La transmisión de vídeo en directo podría llegar a ser muy lenta usando esta tecnología.

2.4 WiFi

WiFi es una familia de tecnologías para la transmisión inalámbrica de datos basada en la familia de protocolos IEEE 802.11 de las capas físico y enlace de datos que permite interconectar múltiples dispositivos.

WiFi utiliza distintos estándares para cifrar los datos transmitidos y brindarnos seguridad entre ellos el mas destacado y seguro es WPA2, aunque todavía se usan otros como WEP o WPA.

Los estándares de WiFi utilizan la banda de 2,4 GHz la cual tienen mayormente disponible, con una velocidad de 11 Mbit/s, 54 Mbit/s y 300 Mbit/se según el estándar. El Bluetooth también trabaja en la banda de 2,4 GHz y por ello se actualizo el estándar del mismo para evitar los conflictos con el uso simultaneo.

El alcance máximo teórico de WiFi es de 200m aproximadamente pero un alcance típico es de 30m aproximadamente en interiores y de entre 50m-100m en exteriores. Las señales del espectro de radio utilizado por WiFi sufren atenuación si hay obstáculos en el LoS (Line of Sight) – es decir, en la línea recta entre emisor y receptor – la cuantía de está atenuación varía dependiendo del material del que está hecho el obstáculo.

2.5 WiFi Ad hoc Mode

WiFi Ad hoc Mode (WiFi IBSS) es uno de los modos de conectividad de WiFi Alliance. Permite realizar una conexión peer-to-peer sin un punto de acceso, es decir, los nodos son todos iguales. Todos los dispositivos conectados a este tipo de red actúan como cliente y punto de acceso al mismo tiempo.

Esta tecnología existe desde hace tiempo, está pensada, sobre todo, para comunicación entre dispositivos embebidos, pero apenas se le ha dado uso y es difícil encontrar información al respecto.

Tiene el mismo alcance y velocidad de transmisión que los estándares WiFi.

2.6 WiFi Direct

WiFi Direct [5] es una marca de certificación para dispositivos que permite que los dispositivos WiFi se conecten directamente, lo que hace que sea sencillo hacer cosas como imprimir, compartir, sincronizar y mostrar. Los productos que disponen de WiFi Direct pueden conectarse entre sí sin unirse a una red tradicional de hogar, oficina o punto de acceso. Se puede establecer una conexión uno a uno o un grupo de varios dispositivos pueden conectarse simultáneamente.

Los dispositivos WiFi Direct emiten una señal a otros dispositivos en el área, informándoles que se puede hacer una conexión. Los usuarios pueden ver los dispositivos disponibles y solicitar una conexión, o pueden recibir una invitación para conectarse a otro dispositivo. Cuando dos o más dispositivos WiFi Direct se conectan directamente, forman un Grupo. Cada grupo de WiFi Direct tiene un Propietario de Grupo que actúa como punto de acceso y los demás dispositivos son clientes.

WiFi direct y WiFi Ad hoc Mode no son lo mismo, la diferencia más significativa entre ellos es la seguridad. En las redes *ad hoc* de Windows, el nivel más alto de seguridad admitido es WEP en entornos de clientes mixtos en cambio WiFi Direct es compatible con WPA2. Otra diferencia es que los dispositivos WiFi Direct también pueden conectarse simultáneamente a las redes inalámbricas existentes. Además mejor descubrimiento de dispositivos también diferencian la conexión WiFi Direct de la WiFi Ad hoc Mode.

Tiene el mismo alcance y velocidad de transmisión que los estándares WiFi.

2.7 WiFi Aware

WiFi Aware [6] es un programa de certificación de WiFi Alliance que amplía la capacidad de WiFi con un descubrimiento rápido, conexión e intercambio de datos con otros dispositivos WiFi, sin la necesidad de una Infraestructura de red tradicional de WiFi, conexión a Internet. WiFi Aware establece conexiones peer-to-peer independientes basadas en la ubicación y las pref-

erencias del usuario.

WiFi Aware está continuamente descubriendo otros dispositivos dentro del rango de WiFi del dispositivo antes de la conexión, lo que facilita la búsqueda de información y servicios cercanos disponibles.

WiFi Aware está preparado para su adopción masiva, con soporte nativo disponible en el sistema operativo Android Oreo.

Características principales de WiFi Aware:

- Proporciona continuo descubrimiento device-to-device antes de la asociación.
- Permite el intercambio multidireccional y simultaneo de servicios o información.
- Funciona de forma dinámica, es decir, los grupos de dispositivos conectados a través de WiFi Aware mantienen su conexión incluso si un dispositivo se sale del alcance
- Funciona bien en interiores y en ambientes abarrotados.
- Permite una comunicación eficiente de dispositivo a dispositivo, cada flujo de datos está asociado con una aplicación o servicio específico
- Da al usuario el control de la privacidad. Opción de entrada o salida de la identidad a través del sistema operativo o la aplicación

Tiene el mismo alcance y velocidad de transmisión que los estándares WiFi.

2.8 TCP y UDP

TCP o Transmission Control Protocol, es un protocolo de la capa de transporte que proporciona un servicio orientado a conexión y fiable que confirma la llegada de los datos y en el orden correcto.

UDP o User Datagram Protocol, es un protocolo de la capa de transporte que proporciona un servicio no orientado a conexión y no fiable, esto quiere decir que se va a intentar por todos los medios que los datos lleguen, pero no lo garantiza. Este protocolo es más rápido pero si hace falta una conexión fiable y se usa UDP, las capas encima deben ocuparse de la detección de errores y de las retransmisiones.

En streaming, una aplicación de tiempo real aunque sea tiempo real blando, si la tasa de errores no es muy alta, la opción de simplemente degradar ligeramente el servicio al receptor del stream en caso de error es

mejor que la opción de hacer retransmisiones a coste de introducir más latencia. La calidad de servicio para el cliente será mejor. Por tanto, se suele transmitir streams sobre UDP en vez de sobre TCP.

2.9 HTTP Streaming

HTTP es un protocolo de la capa de aplicación que también puede ser usado para hacer streaming de vídeo.

Se caracteriza por ausencia de estado (en el servidor), lo cual quita carga al servidor y además facilita iniciar la reproducción en cualquier instante si no se trata de un stream en directo, puede atravesar cualquier firewall o servidor proxy debido a que usa el puerto 80, y permite hacer streaming con la tasa de bits adaptable (la calidad se adapta al ancho de banda del usuario) sin cargar de más trabajo al servidor debido a que la lógica de esta característica se encuentra en el cliente.

Otrea de las ventajas de HTTP es que se puede aprovechar la red de distribución de contenidos (CDN).

HTTP streaming es la solución más popular hoy en día, sobre todo para"live streaming" debido a la facilidad de la implementación de streaming de tasa de bits adaptable.

2.10 RTP y RTCP

RTP (Real-time Transport Protocol) [7] es un protocolo de la capa de transporte que proporciona las condiciones adecuadas para que aplicaciones que transmiten datos en tiempo real, tales como audio y vídeo, puedan efectuarse. Utiliza el protocolo RTCP[8] para enviar datos de control entre el emisor y receptor de la secuencia RTP. Los paquetes son enviados aproximadamente cada cinco segundos, y contienen datos que ayudan tanto a verificar la transmisión como a sincronizar los flujos de audio y vídeo utilizando las marcas de tiempo de los paquetes RTP.

Puede utilizarse por encima de los protocolos TCP o UDP. Este ultimo es más utilizado debido a que es más ligero y rápido que TCP y para servicios de streaming es lo mas importante aun que no garantice la llegada de todos datos dado que no es tan importante si se pierde una imagen del vídeo por ejemplo, el usuario ni si quiera se daría cuenta.

Estos dos protocolos son protocolos con estado lo que hace más difícil implementar streaming de tasa de bits adaptable por lo cual en la mayoría de los usos ha sido desplazado por HTTP Streaming.

2.11 RTSP

RTSP[9] (Real Time Streaming Protocol) es un protocolo, con estado en el servidor, de las capas de sesión y aplicación que establece y controla flujos sincronizados de datos, ya sean de audio o de vídeo. RTSP no se encarga de la transmisión de datos, la mayoría de los servidores y clientes RTSP utilizan el Protocolo de transporte en tiempo real (RTP) junto con el Protocolo de control en tiempo real (RTCP) para la entrega en la capa de transporte.

El hecho de que sea un protocolo con estado hace mas difícil la implementación de streaming de tasa de bits adaptable.

2.12 Aplicaciones parecidas

2.12.1 Aplicaciones para compartir archivos

Algunas de las aplicaciones que hemos encontrado que hagan uso de la tecnología WiFi Direct son: WiFi Shoot, SuperBeam, WiFiShare, HitcherNet, SHAREit, ShareLink, Zapya... estas aplicaciones nos brindan la oportunidad de compartir archivos de un dispositivo a otro, pero la relación es de uno a uno. Podemos ver que uno de los beneficios de transmitir los archivos por las nuevas tecnologías WiFi por encima de tecnologías mas antiguas como el bluetooth es el considerable aumento en la velocidad de transmisión permitiendo enviar archivos mucho mas grandes en menos tiempo. Algunas de las aplicaciones como SuperBeam implementan funcionalidades de autenticación por escaneo de códigos QR.

Ninguna de estas aplicaciones permiten compartir un archivo a varios dispositivos al mismo tiempo.

2.12.2 Cam Wimote

Cam Wimote [10] es una de las aplicaciones existentes que mas se parecen al proyecto que estamos desarrollando, esta aplicación permite controlar de forma remota a través de WiFi Direct la cámara de otro dispositivo, aunque tiene un enfoque diferente dado que se pretende que sea el dispositivo que se conecta quien controle la cámara y decida cuando hacer la foto. Esta aplicación solo da soporte a fotos y no a vídeos. Tiene soporte para que varios dispositivos de una misma red de WiFi Direct puedan conectarse a la cámara y hacer fotos.

2.12.3 RTSP Camera Server

RTSP Camera Server [11] es una aplicación que permite convertir nuestro dispositivo en un servidor para retransmitir streaming a otros dispositivos. Esta aplicación permite que varios clientes puedan tener servicio del mismo streaming que se esta emitiendo en directo desde la cámara y micrófono del servidor. Es la aplicación mas parecido al proyecto en cuanto al servicio de streaming que da soporte aunque esta aplicación solo permite actuar como servidor al que nos podemos conectar siendo necesario utilizar una aplicación cliente RTSP para conectarnos y reproducir como podría ser VLC o similares.

3 Elección de tecnologías

3.1 Android

Para el desarrollo de este proyecto dado que se busca utilizar una aplicación móvil se ha elegido desarrollarla para el sistema operativo Android a causa de que es el sistema operativo con la mayor tasa de utilización en los smartphones con un porcentaje aproximado al 74.5%[12] al comienzo del año 2019. De esta forma el desarrollo para la aplicación para Android tiene el alcance de llegar a muchos mas dispositivos así como la posibilidad de encontrar mucha mas documentación y ejemplos que si la desarrollásemos para otros sistemas operativos como iOS o WindowsPhone. Además Android da soporte con su API tanto a la tecnología de WiFi Direct como a WiFi Aware.

3.2 WiFi Direct

Hemos decido llevar el proyecto a cabo usando la Wifi Direct implementada en la API de Android dado que esta tecnología esta presente desde la versión 4.0 del sistema operativo lo que significa que la gran mayoría de los dispositivos tendrán acceso a ella mientras que por lo que hemos podido comprobar incluso aunque el soporte de Android para WiFi Aware esta presente desde la versión Oreo (8.0), apenas existen dispositivos en el mercado con el hardware necesario para poder utilizarla.

Nos habría gustado poder usar Wifi Aware porque pensamos que con esta nueva tecnología podríamos llevar el proyecto a cabo mucho mas fácilmente y con grandes ventajas técnicas pero debido a la imposibilidad de probar la aplicación con dispositivos reales hemos tenido que limitarnos a Wifi Direct. El proyecto podría utilizar la tecnología Wifi Aware en un futuro, de forma exclusiva o incluso combinándola con WiFi Direct para continuar dando servicio a los dispositivos que no la soporten todavía.

3.3 Java

Java es un lenguaje de programación orientado a objetos multiplataforma muy popular en la actualidad. Lo hemos elegido para el desarrollo de la aplicación dado que es un lenguaje de programación con el que ambos tenemos experiencia

y es además el lenguaje principal de desarrollo de aplicaciones en Android y por este motivo se puede encontrar mucha mas documentación y ejemplos que otros lenguajes mas nuevos como Kotlin.

3.4 RTP sobre UDP

Como protocolo de transmisión hemos escogido RTP sobre UDP ya que es un protocolo confiable orientado a la transmisión audio o vídeo en tiempo real. Junto con RTP utilizaremos el protocolo RTCP para el control y la sincronización de los flujos multimedia. Sobre UDP porque es un protocolo más ligero y rápido aun que menos fiable que TCP, pero para el servicio de streaming es mejor aun que podría haber perdida de datos que simplemente no se mostrarán al usuario y puede que éste ni se dé cuenta.

3.5 RTSP

Para establecer y controlar el flujo de datos entre servidor y cliente utilizamos RTSP. Este protocolo nos pareció el mas adecuado porque se adapta perfectamente a las necesidades de nuestro proyecto, sobre todo para la parte del servidor o emisor del streaming, ya que se encarga de controlar la conexión entre el servidor y el cliente y atender las peticiones del cliente aun que el cliente también puede atender algunas peticiones del servidor. Es un protocolo muy estandarizado y que múltiples reproductores multimedia pueden abrir como puede ser el reproductor integrado del sistema operativo Android.

Respecto al interés para nosotros de streaming con tasa de bits adaptable, si un flujo tiene que pasar de dispositivo en dispositivo por muchos dispositivos, el efecto de usar este tipo de streaming sería que después de X hops, la calidad del vídeo sea la que soporta el enlace peor. Por otro lado, el hecho de tener que parar la reproducción durante un tiempo a la espera de más datos (progressive download) en los enlaces peores es menos problema que perder calidad ya que, si los mecanismos de adaptación bajan demasiado la calidad (menos frames por segundo o menor resolución por frame) es posible que se pierdan detalles cruciales. Por tanto, en los usos previstos para nuestra aplicación, es mejor no usar streaming con tasa de bits adaptable.

Además, puesto que nuestra aplicación no se conecta a la red de los operadores de telecomunicaciones, usar HTTP streaming tampoco daría acceso a la red de distribución de contenido que existe para HTTP. Finalmente, dado que nuestra aplicación cae dentro de la categoría de “live streaming”, tampoco nos interesa facilitar el salto a cualquier punto de stream.

Por esas razones, no nos importa usar los protocolos RTSP y RTP/RTCP, aunque sean un poco anticuados para hacer streaming, en vez de el actualmente mucho más popular HTTP.

3.6 libstreaming

libstreaming [13] es una librería de código abierto con licencia Apache 2.0 para el sistema operativo Android que nos ofrece un API fácil de integrar en aplicaciones con el que crear streamings desde la cámara y micrófono de nuestro dispositivo utilizando el protocolo RTP sobre UDP. Mas allá de esto la librería también implementa parte del protocolo RTSP para el control de streamings tanto en su forma de servidor como la de cliente. Brinda soporte para múltiples codificadores multimedia como pueden ser H.264 para el vídeo y AAC para el audio. Hemos elegido esta librería para basarnos en el desarrollo de la aplicación dado a la gran versatilidad que nos aporta como un punto de arranque para todo el desarrollo necesario para el proyecto.

3.7 libVLC

LibVLC [14] es una potente librería para la gestión de contenido multimedia, puede ser integrada fácilmente en las aplicaciones y se pueden llegar a conseguir resultados como el famoso reproductor multimedia VLC [15] dado que este está basado en esta. Es una librería de código abierto con licencia LGPL, además posee una librería que ofrece una API sobre este core para Android con una licencia GPLv2, nosotros utilizaremos esta ultima en el proyecto.

Hemos elegido usar esta librería para la reproducción de los streaming de vídeo debido a su potencia, versatilidad y la facilidad de integración de la misma para la reproducción de un streaming desde un servidor RTSP.

3.8 Android Studio

Android Studio es el entorno de desarrollo oficial integrado (IDE) para desarrollo de aplicaciones para Android. Está disponible para los sistemas operativos Windows, Mac OS X y Linux. Brinda una gran cantidad de herramientas para ayudar en el desarrollo y aumentar la productividad a la hora de crear aplicaciones. Ofrece la posibilidad de crear las

interfaces gráficas de forma visual, así como la posibilidad de probar la aplicación en diferentes dispositivos virtuales con diferentes versiones de Android. Hemos usado este entorno para desarrollar la gran parte del código de nuestro proyecto.

3.9 Gradle

Gradle [16] es una herramienta que permite automatizar la construcción de proyectos, permite automatizar entre otras cosas la compilación y la validación del proyecto, además de gestionar las dependencias de forma recursiva. Está basada en Groovy, un lenguaje de programación muy parecido a Java y está incorporada en Android Studio.

3.10 Git y Github

Git es un software de control de versiones diseñado pensando en la eficiencia y el mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en los archivos y coordinar el trabajo que varias personas realizan sobre archivos compartidos. Gracias a Git podemos trabajar de forma colaborativa con el menor numero posible de conflictos. Es un software al que los dos ya estamos acostumbrados dado a su extendido uso y que es fundamental a la hora de trabajar en equipo el desarrollo de una aplicación.

Hemos usado un repositorio git alojado en Github, esta plataforma que aloja miles de repositorios es la plataforma líder en cuanto al alojamiento de repositorios de código siendo un referente del código abierto en la actualidad. Ofrece el alojamiento de repositorios de forma gratuita Para llevar el control de versiones del código de nuestro proyecto hemos usado un repositorio Git alojado en GitHub dado que es un sistema que nos es familiar y de uso muy extendido.

3.11 Trello

Trello [17] es un software de administración de proyectos con interfaz web que utiliza el método Kanban. Esta herramienta permite la creación de tableros y listas de tarjetas virtuales. Las tarjetas son el representativo de tareas pendi-

entes y/o nuevas ideas del proyecto, y junto con las listas se van organizando de tal forma que podemos saber en cada momento cual es el progreso de los objetivos a realizar y quien es la persona asignada a dicha tarea. Es una herramienta muy interactiva y con un amplio uso para la repartición de tareas en proyectos desarrollados con metodologías ágiles.

3.12 LaTeX y Overleaf

LaTeX[18] es un sistema de composición de textos que está orientado especialmente a la creación de documentos de alta calidad debido al gran abanico de posibilidades que nos ofrece es muy usado para la creación de documentos, artículos y libros científicos así como trabajos de fin de grado, máster o tesis doctorales. Permite la fácil modificación de los estilos de todo el documento al mismo tiempo que ofrece ayudas para la gestión de los índices y la bibliografía. LaTeX está organizado sobre TeX.

Hemos decidido usar LaTeX por encima de Microsoft Word dada la complejidad del documento y con el objetivo de obtener una memoria del proyecto mas profesional y de mejor calidad.

Como editor de LaTeX hemos usado Overleaf [19], una plataforma que ofrece la oportunidad del desarrollo de documentos sobre latex de forma colaborativa y centralizada de tal forma que varios participantes pueden estar modificando el documento simultáneamente sin conflictos desde cualquier ubicación en la que dispongan conexión a internet.

4 Especificación

Tras realizar la investigación sobre las posibles tecnologías que se pueden aplicar a nuestro proyecto y escoger las que mejor satisfacen las necesidades de éste procedemos a diseñar y especificar nuestra aplicación.

4.1 Requisitos

Para cumplir los objetivos que proponemos, nuestra aplicación se puede dividir en tres apartados que son:

- Conexiones peer-to-peer.
- Transmisión del streaming de vídeo y audio generado por la cámara y micrófono del dispositivo.
- Recepción y reproducción del streaming junto con retransmisión del mismo como nodo de transito.

4.1.1 Conexión

Lo primero que nuestra aplicación debe hacer es poder conectarse a otro dispositivo con las tecnologías seleccionadas en el apartado de elección de tecnologías. Con lo cual debe cumplir estos requisitos:

- Requisito funcional: Buscar dispositivos cercanos disponibles.
- Requisito funcional: Realizar una conexión y crear un enlace con otro dispositivo cercano.

4.1.2 Transmisión de datos

En segundo lugar, una vez establecida la conexión entre dos o mas dispositivos la aplicación debe:

- Requisito funcional: Mostrar al usuario que desea hacer streaming lo que ve su cámara.
- Requisito funcional: Preparar los datos de la cámara y el micrófono para poder ser enviados.
- Requisito funcional: Identificar al destinatario.
- Requisito funcional: Enviar los datos preparados, o cualquier archivo o mensaje de texto.

- Requisito funcional: No dejar trazas en el móvil emisor tras realizar un streaming de vídeo.

4.1.3 Recepción, interpretación y retransmisión de datos

Por ultimo, la aplicación debe poder manejar los datos recibidos y, por tanto, cumplir los siguientes requisitos:

- Requisito funcional: Recibir y tratar los datos recibidos.
- Requisito funcional: Reproducir el vídeo recibido si el usuario quiere.
- Requisito funcional: Retransmitir los datos recibidos a otros dispositivos conectados al mismo tiempo que se están recibiendo.

5 Diseño

5.1 Consideraciones generales de diseño

5.1.1 Funcionamiento de Wifi Direct

Cuando empezamos a estudiar el diseño de la aplicación para una red con WiFi Direct nos encontramos ante varias situaciones debido a su diseño en el cual las conexiones de WiFi Direct crean un grupo que es muy parecido a una pequeña red local.

Esta red se crea alrededor de uno de los dispositivos involucrados en la conexión, que será designado como Group Owner y actuará de forma similar a un punto de acceso como si se tratase de una red WiFi "corriente" en la que el router es el punto de acceso y centro de la red.

Una vez se ha establecido la conexión entre dos dispositivos y se ha creado este grupo, otros dispositivos pueden conectarse a la red, bien creando una conexión directa al Group Owner o ya sea conectándose a alguno de los dispositivos conectados a la red el cual nos redirigirá al Group Owner para conectarnos a él.

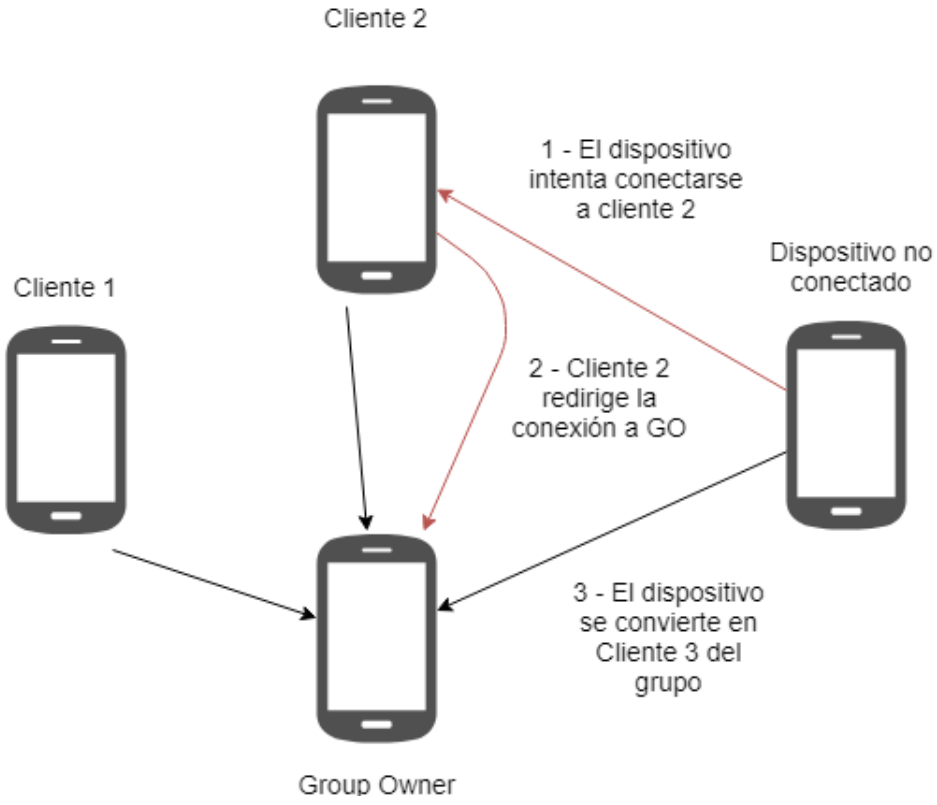

Figure 5: Conexiones WiFi Direct

Este proceso se puede observar en la Figura 5, pero para ello necesitaremos tener dentro de alcance al Group Owner o en caso contrario no podremos conectarnos a la red.

Cuando se crea esta conexión se lleva a cabo un proceso de negociación de la conexión mediante el cual se establecen ciertos aspectos sobre como será la red, quien será el Group Owner y las IPs de los dispositivos involucrados en la misma. Cuando dos dispositivos intentan conectarse se quedan en estado "invitado" hasta que se negocia y se autoriza la conexión. El proceso de conexión suele tardar entre unos 5 y 8 segundos.

5.1.2 Multihop y red *ad hoc* con WiFi Direct

Para crear una red *ad hoc* con WiFi Direct se necesita que un cliente que ya este conectado a un grupo sea capaz de conectarse a un segundo grupo de tal forma que este actué como nodo de enlace entre los dos grupos permitiendo conectar las redes que forman estos como podemos ver en la Figura 6.

Esto también permite que cualquier dispositivo se pueda unir a la red aunque se encuentre fuera del alcance del Group Owner: el nuevo dispositivo crearía un nuevo grupo en el que él mismo es el Group Owner y uno de los clientes de un grupo existente actuaría como nodo enlace conectado como cliente tanto al Group Owner del nuevo grupo, es decir el nuevo dispositivo, como al Group Owner del grupo existente.

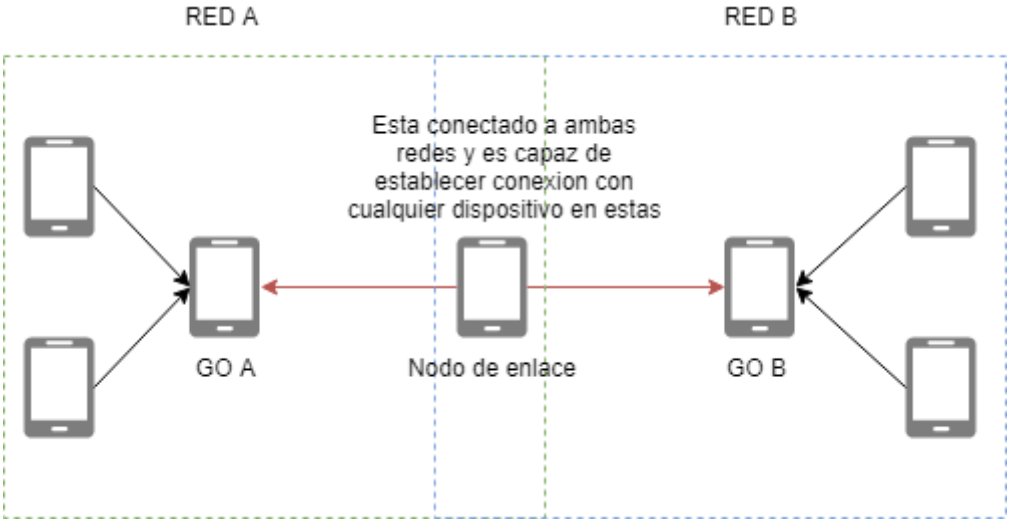

Figure 6: Múltiples redes WiFi Direct

Uno de los problemas con el que nos encontramos al enfrentarnos a unir dos redes de WiFi Direct es que cada red no se encuentra al alcance de la

otra. Solo el nodo de enlace tiene la posibilidad de acceder a ambas redes y es este el que tiene la responsabilidad de encaminar los paquetes para que lleguen a su destino. Por esta razón si se desea ser capaces de enviar paquetes directos entre dispositivos de diferentes redes se necesitan crear las rutas para el encaminamiento aunque para nuestro caso en concreto debido a que queremos retransmitir el streaming a todos los dispositivos de la red que lo acepten utilizaremos técnicas de tipo *flooding* y *multihoping*.

WiFi Direct nos otorga la posibilidad de que este Group Owner sea elegido al azar o bien que nosotros designemos quien va a ser el Group Owner o la prioridad que tiene a la hora de ser nombrado el Group Owner si no esta establecido. Esto puede ser realmente beneficioso a la hora de crear una red *ad hoc* con WiFi Direct ya que podríamos decidir observando los mapas de la red cuales deberían ser los Group Owners para optimizar la red.

También nos encontramos con que es posible que a través de la red de WiFi Direct los datos sean redireccionados hacia la red convencional de Internet si cualquiera de los dispositivos involucrados en la red tiene acceso las dos redes. El dispositivo conectado a ambas redes actuará como nodo de enlace por tanto podrá reenviar los datos a servidores externos o bien a otros clientes que se hayan conectado a el por la red de Internet. Este paso de datos entre la red de WiFi Direct y la red de Internet debe ser implementado por la aplicación que utilice WiFi Direct, así como el encaminamiento de estos paquetes.

5.1.3 Limitaciones de la implementación de WiFi Direct en Android

Cuando hemos investigado sobre como diseñar nuestra aplicación usando WiFi Direct nos hemos encontrado con una gran limitación en la implementación de la API de WiFi Direct que nos ofrece Android y que no nos ha permitido continuar el diseño y el desarrollo de la aplicación por el camino que deseábamos.

Esta limitación es que un dispositivo no puede mantener la membresía en varios grupos de WiFi Direct simultáneamente, esto nos condiciona a que no vamos a poder unir varias redes de WiFi Direct y por tanto no se podrá conseguir crear una red ad hoc para nuestra aplicación como se buscaba.

Según el articulo publicado por WiFi Alliance [20], que un dispositivo WiFi Direct sea capaz de pertenecer a varios grupos a la vez es una especificación opcional de implementar y el sistema operativo Android nunca la llego a implementar.

Algunas de las posibles soluciones que hemos encontrado son:

- Modificar SO Android:

  Android es un sistema operativo de código abierto pero aun así presenta ciertas limitación para los desarrolladores a la hora interactuar con algunos servicios o con el hardware.

  En algunos artículos [21] [22] se habla sobre la modificación de SO Android para cambiar ciertos comportamientos de WiFi Direct o incluso WiFi convencional, ya que los dos comparten el hardware, para luego combinar el uso de ambos y así poder permitir la creación de una red *ad hoc* creando distintas tablas de enrutamiento que no se pueden hacer en la capa de aplicación.

  Para ello se necesita tener un gran conocimiento sobre este sistema operativo y además, por lo general, se necesita hacer rooting del dispositivo para obtener los máximos privilegios y superar todas las limitaciones que los operadores de telefonía móvil y los fabricantes de hardware colocan.

  Debido a nuestra actual falta de conocimiento sobre el sistema operativo Android y que, según el articulo [21], la legalidad de las modificaciones de éste sigue siendo controvertida en varios países hemos decidido descartar esta opción.

- Saltar de red en red para actuar como nodo de transito:

  En un artículo sobre redes Multihop *ad hoc* con WiFi Direct [21] también se habla de utilizar un dispositivo como nodo de transito haciéndolo saltar de una red en otra, es decir, conectarlo a un grupo de WiFi Direct y tras recibir datos de ese grupo, desconectarlo de ese grupo y conectarlo a otro grupo cercano y transmitir los datos a ese grupo.

  Está solución no nos pareció viable ya que la conexión entre dispositivos mediante WiFi Direct es lenta, aproximadamente unos 5 segundos, lo que hace muy difícil ofrecer un servicio como el de streaming de vídeo que requiere la transmisión de una gran cantidad de datos y que el flujo de estos sea continuo.

5.1.4 Impacto de las limitaciones del WiFi Direct de Android

Debido a que no hemos podido encontrar ninguna solución realmente viable y digna de considerar a este problema vamos a tener que limitar nuestra aplicación a que se conecte únicamente a un grupo de WiFi Direct con un único GO y realice el streaming de vídeo por esta, lo cual va restringir el traspaso del stream fuera del limite de alcance de la señal del propietario del grupo.

Aunque no se pueda llevar a cabo el proyecto como estaba ideado en un primero momento por los motivos mencionados aun así se va a realizar un diseño en otras funcionalidades como puede ser el streaming de vídeo para que esta permita la creación de una red Multihop donde los dispositivos puedan ser nodos de transito del streaming de tal forma que se pueda dar el caso de actuar como nodo de enlace entre la red de WiFi Direct y la red convencional de Internet.

Además, el hecho de diseñar la funcionalidad del streaming de vídeo de forma que sea posible actuar como nodo de transito nos abre la posibilidad de, en un futuro, reimplementar el modulo de la aplicación que realiza las conexiones con WiFi Direct de tal forma que podamos migrarlas a una nueva tecnología como podría ser, por ejemplo, WiFi Aware y de esta forma ya si poder crear una red *ad hoc* de streaming cuando nos sea posible crear las conexiones básicas para crear este tipo de red.

Se va a diseñar el modulo de las conexiones con WiFi Direct de tal forma que este separado de toda la lógica de envió y recepción de datos de las conexiones de tal forma que sea fácil de implementar e integrar un nuevo modulo en el futuro con WiFi Aware o una nueva tecnología.

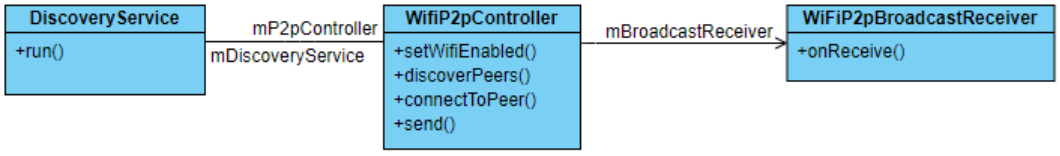

Figure 7: Diagrama de clases del modulo de WiFi Direct

Para ello, crearemos las clases que podemos ver en diagrama de la Figura 7 para gestionar las conexiones mediante el WifiDirectController que se apoyara en la clase WifiP2pBroadcastReceiver para recibir las notificaciones del sistema. Además crearemos una clase auxiliar para buscar y crear conexiones utilizando el controlador que será DiscoveryService.

5.2 Gestión de las conexiones

Para gestionar toda la interacción de las conexiones se va a crear un paquete con todas las clases necesarias para ello de tal forma que quede aislado del modulo de conexión de WiFi Direct.

Para ello hemos elegido utilizar la API de Java NIO (Non-blocking IO) la cual nos ofrece el uso de sockets que no son bloqueantes, gracias a esto con un solo hilo somos capaces de controlar múltiples conexiones frente al diseño de conexiones bloqueantes para la que necesitaríamos un hilo por conexión.

Es muy importante la optimización en la gestión de las conexiones, en este caso evitando el uso de un número excesivo de hilos, ya que estamos desarrollando una aplicación para dispositivos móviles y se debe intentar ahorrar al máximo el consumo de recursos y batería. Esta aplicación podría correr en segundo plano para actuar como nodo de transito de la red aunque el usuario no este interactuando con el dispositivo, por ejemplo, y esto no resulta atractivo para un usuario si la aplicación consume mucha batería.

La API nos ofrece los selectores los cuales son capaces de registrar múltiples canales que en nuestro caso serán los sockets y utilizarlos para realizar las operaciones correspondientes ya sean de escritura (envió de datos) o de lectura (recepción de datos).

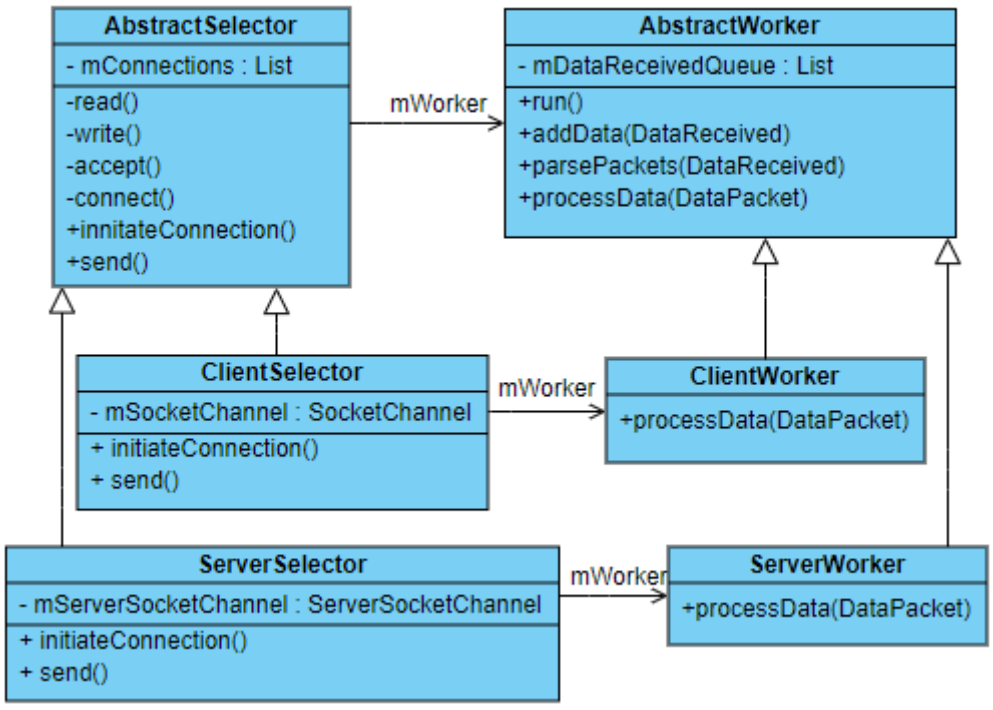

Figure 8: Diagrama de clases del paquete de conexiones

Para el uso de esta API vamos a implementar dos clases abstractas las cuales serán la base para las distintas implementaciones para actuar ya sea como cliente o servidor y estas son AbstractSelector y AbstractWorker.

El AbstractSelector será la clase principal e implementara toda la interacción entre los canales y el selector, esto incluye todas la operaciones básicas como son escribir, leer, aceptar una conexión y establecerla, así como gestionar todas las conexiones asociadas con nuestro selector.

La clase AbstractWorker es la clase encargada de hacer el tratamiento de los datos recibidos por AbstractSelector y se encarga de procesar los datos en un hilo separado al del selector para de esta forma agilizar las conexiones. Se agregarán los datos recibidos usando una cola y se brindará una función abstracta para que las subclases puedan gestionar los datos recibidos una vez estén completos.

Además, posteriormente crearemos las implementaciones correspondientes para cada una de estas clases para actuar como clientes y servidores de tal forma que podamos establecer y tratar las conexiones de forma diferente. Podemos ver en el diagrama de clases de la Figura 8 todas las implementaciones que realizaremos.

5.3 Streaming de Vídeo

5.3.1 Estado inicial de la librería

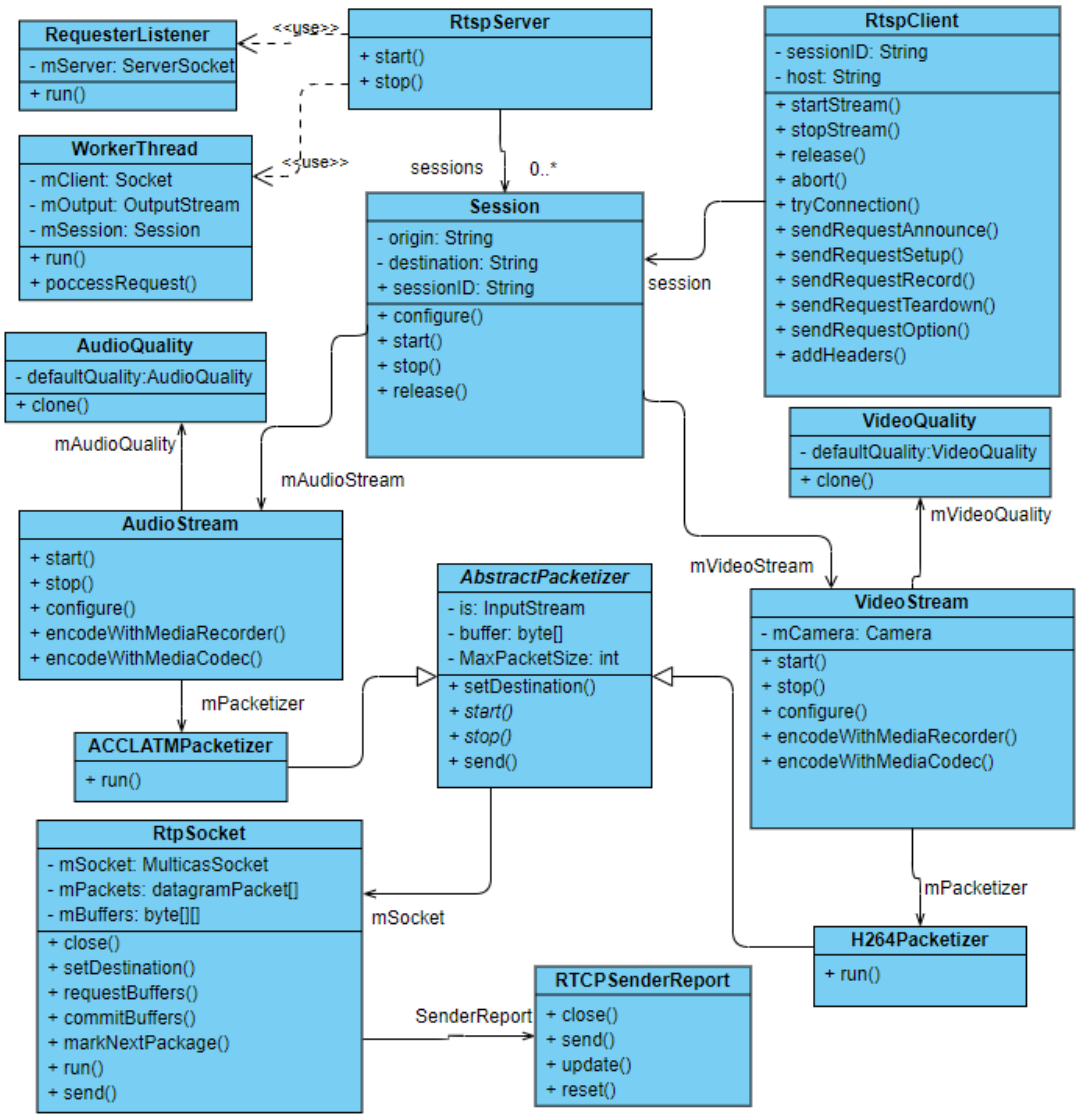

Figure 9: Diagrama de clases de la librería

Como ya hemos indicado en la sección de elección de tecnologías, para hacer streaming de vídeo vamos a usar libstreaming, una librería que hemos encontrado tras una larga tarea de investigación cuyo diagrama de clases del estado inicial presentamos en la Figura 9.

La librería esta orienta a poder crear flujos de vídeo y audio desde nuestra cámara y micrófono para después poder emitirlo mediante el protocolo

RTP. Esta librería nos brinda dos clases RtspServer y RtspClient que implementan el protocolo RTSP para llevar a cabo el control de estos flujos y poder o bien emitir a un servidor dado con el cliente o retransmitir a los clientes que se conecten a nuestro dispositivo con el servidor.

RtspClient trabaja con una sola sesión y esta preparado para enviar el streaming, que generamos a través de nuestra cámara y micrófono, a un servidor (ej. WoWza) mientras que RtspServer está a la escucha de la conexión de clientes y esta preparado para retransmitir el streaming que generamos, por cada cliente se crea una nueva sesión. Esta última clase solo implementa peticiones para retransmitir el streaming generado por el mismo servidor y no implementa peticiones para recibir stream de un cliente (RtspClient).

Cada sesión crea un AudioStream y un VideoStream. Estos obtienen datos de la cámara y el micrófono del dispositivo móvil, testean los codecs que van a usar y para tratar los datos utilizan los MediaCodecs. Cada AudioStream y VideoStream crearan un Packetizer dependiendo del formato en el que se quiere enviar los datos (AMR o ACC para audio y H264 o H263 para el vídeo). Los Packetizers son los encargados de empaquetar los datos y enviárselos al destinatario usando para esto la clase RtpSocket que los enviara usando el protocolo RTP junto con la ayuda de la clase RTCPSenderReport para enviar los datos del protocolo RTCP.

Esta librería tiene tres modos de hacer streaming dependiendo de la forma de codificar los datos:

- Usando MediaRecorder API
- Usando MediaCodec API y el método buffer-to-buffer que requiere Android 4.1 o superior
- Usando MediaCodec API y el método surface-to-buffer que requiere Android 4.3 o superior y solo está disponible para el flujo de la cámara.

Nosotros hemos probado los tres modos y hemos llegado a la conclusión de que el mejor es el de MediaCodec APi y el método buffer-to-buffer, ya que el modo MediaRocorder API directamente no funciona por usar pipes que están deprecados y MediaCodec API y el método surface-to-buffer solo funciona para el vídeo y además coge la imagen directamente de la surface, es decir , debe esperar que la imagen sea renderizada, por lo cual introduce un poco más de latencia.

5.3.2 Streaming a varios dispositivos simultáneamente

Cuando contemplamos los posibles escenarios a la hora de realizar el streaming de vídeo tanto en una red *ad hoc* donde es necesario hacer *multihopping* como en una red simple de WiFi Direct, en primer lugar tenemos las situaciones en la cual un dispositivo necesita retransmitir un streaming de vídeo a varios clientes simultáneamente. Esta situación se podría dar en primer lugar si se trata del Group Owner y el centro de la red, de tal forma que puede retransmitir a todos los dispositivos que están conectados a él, dando la posibilidad a que todos esos dispositivos puedan hacer *multihop* mas tarde y reenviar el streaming. Además, nos brinda la oportunidad de poder abastecer a toda una pequeña red de WiFi Direct creando conexiones directas con cada dispositivo. Podemos observar estas situaciones en la Figura 10.

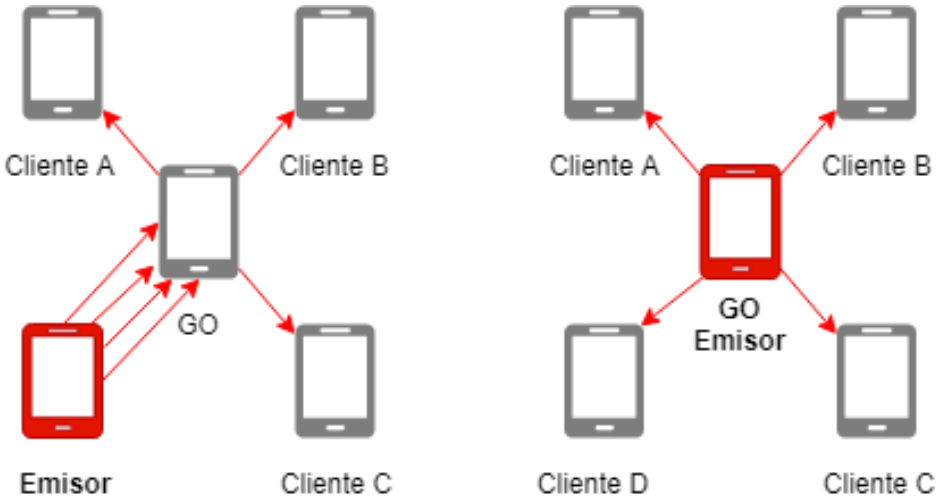

Figure 10: Escenarios de streaming a varios dispositivos simultáneamente

También se podría dar el caso de que un dispositivo, que es un nodo de enlace entre dos grupos de WiFi Direct, quiera retransmitir en directo. Teóricamente si el dispositivo emisor está conectado a las dos redes, debería ser capaz de retransmitir a todos los dispositivos en cualquiera de estas, estableciendo conexión directa con cada uno de los dispositivos de cada red como observamos en el diagrama de la Figura 11.

La funcionalidad de multistreaming es totalmente necesaria y ha sido nuestro primer problema a solucionar, esto es porque primero la aplicación debe ser capaz de retransmitir a varios dispositivos a la vez, en caso de estar conectados a varios, para que si cualquiera de ellos actúa como un nodo de enlace pueda retransmitir por medio de *multihopping* el streaming a los dispositivos del otro grupo de WiFi Direct.

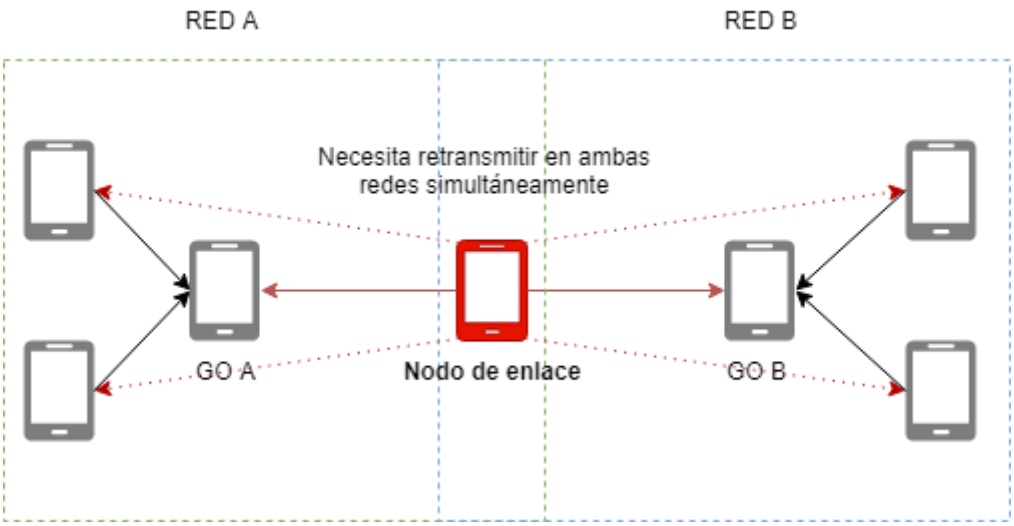

Figure 11: Escenarios de streaming a varios dispositivos simultáneamente

5.3.3 Problema en la retransmisión simultanea con la librería libstreaming

Nos encontramos con un serio problema de diseño en la librería libstreaming para implementar esta funcionalidad. Aunque esta diseñada para crear nuevas sesiones con cada conexión en el servidor, en el estado inicial de la librería era imposible dado que estas sesiones crean a su vez nuevos VideoStream y AudioStream, estas clases están diseñadas para interactuar directamente tanto con la cámara como con el micrófono respectivamente existiendo el gran problema que en Android estos periféricos no pueden ser usados mas de una vez al mismo tiempo siendo bloqueantes las instancias. Tanto VideoStream como AudioStream intentaban crear nuevos accesos a la cámara y al micrófono de tal forma que no podían acceder en el caso del audio o le "robaban" la cámara al VideoStream de la sesión anterior.

Esto es un claro problema de diseño en la librería ya que el servidor esta implementado de tal forma que pueda mantener sesiones simultaneas con varios clientes, pero no sirve de nada poder mantener estas sesiones si no somos capaces de retransmitir el streaming a los clientes de forma simultanea.

Para resolver este problema debemos modificar la librería y separar la cámara y el micrófono del VideoStream y AudioStream creando una clase intermedia entre los streams y los paquetizadores que maneje estos periféricos, recolecte los datos de estos y se los pase a cada paquetizador que los necesite.

5.3.4 Nodo de transito y *multihopping* de streamings con WiFi Direct

Para poder retransmitir el streaming en una red *ad hoc* encontramos en un segundo escenario en el que necesitamos que un dispositivo sea capaz de recibir y retransmitir el streaming al mismo tiempo para así actuar como un nodo de transito permitiéndonos crear una red Multihop. Cabe destacar que el dispositivo que actúa como nodo de transito también debería ser capaz de reproducir el streaming.

En el caso de una red sencilla de WiFi Direct, en el que un dispositivo ofrece el servicio de streaming y otros dispositivos de la red quieren recibirlo, el emisor podría enviar el stream a todos los potenciales receptores de forma simultánea, estableciendo conexiones directas con cada uno de ellos. Sin embargo, este procedimiento es poco escalable, dado que el streaming de los flujos de vídeo en alta calidad implica transferir grandes cantidades de datos. Sería más eficiente que el dispositivo que ofrece un servicio de streaming lo envie al GO y el GO se encargue de retransmitirlo al resto de sus clientes, tal como se ilustra en la Figura 12.

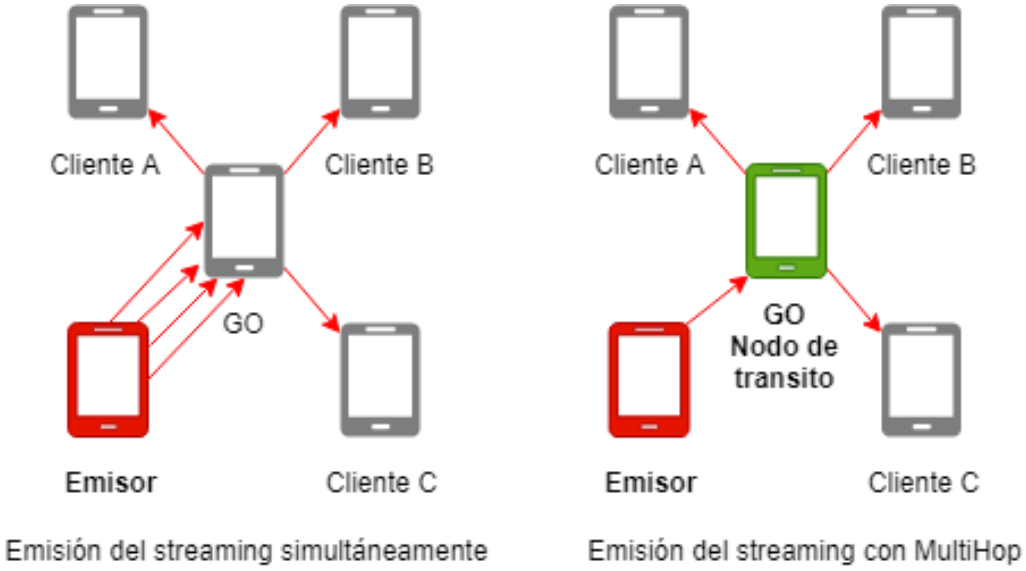

Figure 12: Escenarios streaming con *multihopping*

Pero ya seria totalmente necesario en el caso que un cliente estuviese conectado a una segunda red y queramos retransmitir el streaming de nuevo para distribuirlo por la nueva red. Si además esto se extiende a otros dispositivos que también actúan como nodos de enlace y nos conectan con otras redes ya nos permitiría distribuir el streaming por una red *ad hoc*.

Como podemos ver en la Figura 13 aquí no solo actuaria el Group Owner como nodo de transito sino que a su vez un cliente actúa como nodo de transito para el streaming y como nodo de enlace entre ambas redes. Cabe

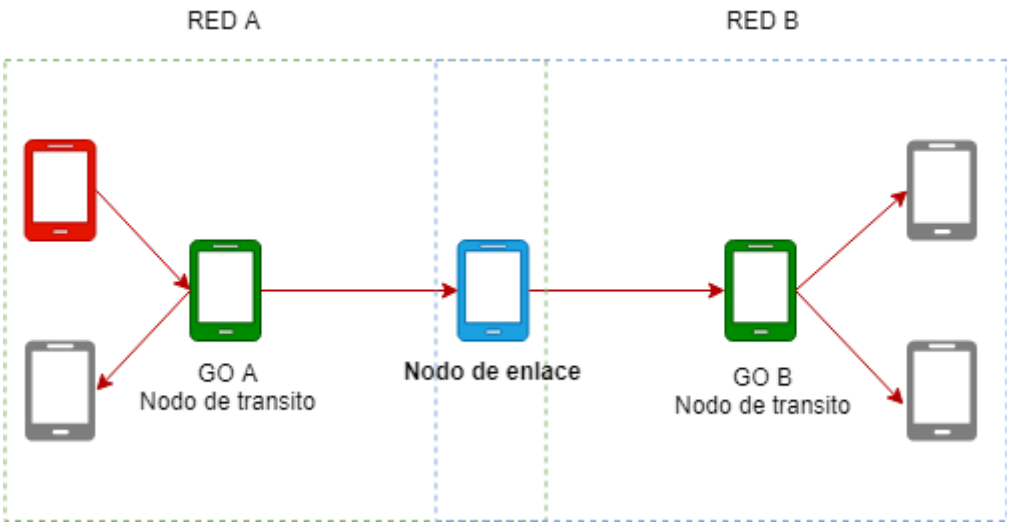

Figure 13: Escenarios streaming con *multihopping*

destacar que cualquier dispositivo conectado a la red de WiFi Direct que también disponga de una conexión a Internet podría actuar como nodo de enlace de ambas redes y permitir la retransmisión del streaming hacia las redes convencionales.

Cuando queremos ampliar un servicio como el de streaming de vídeo en dispositivos móviles algo en lo que debemos pensar además es en la carga que ponemos tanto sobre el dispositivo móvil como sobre las conexiones de la red, cuanto mas se expande la red mas importante es optimizar las conexiones y reducir la carga para la retransmisión.

5.3.5 Nodo de tránsito y *multihopping* con la librería libstreaming

Tal como se ha visto anteriormente, la librería libstreaming contiene una clase que implementa un cliente RTSP y una clase que implementa un servidor RTSP.

Según el protocolo RTSP [9], el cliente RTSP que quiere recibir (modalidad "reproducir") un stream de un servidor RTSP utiliza las directivas: DESCRIBE, SETUP, PLAY y TEARDOWN. Primero, el cliente envía la petición DESCRIBE a la que el servidor contesta con información de inicialización, a continuación el cliente debe realizar una petición SETUP, estableciendo el valor `play` para el parámetro `mode`, por cada pista multimedia que se desea recibir (normalmente, uno para el audio y otro para el vídeo), para finalmente enviar una petición PLAY indicando al servidor que se puede iniciar la transmisión. La petición TEARDOWN (una por stream, al igual que SETUP) termina la sesión y cierra el stream. Ver la

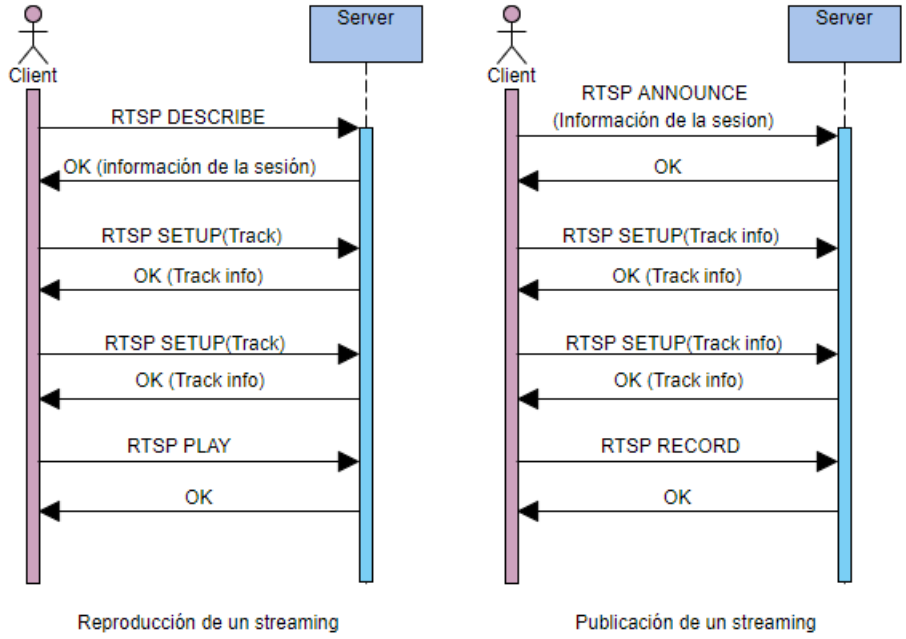

Figure 14: Secuencias de peticiones en RTSP

Figura 14 para una representación visual de este escenario.

Este es el escenario habitual cuando conectamos nuestro dispositivo con un cierto reproductor multimedia a un servidor RTSP para así poder visualizar el streaming en nuestro dispositivo.

Según el protocolo RTSP, el cliente que quiere enviar (modalidad "publicar") un stream a un servidor RTSP utiliza las directivas: ANNOUNCE, SETUP, RECORD y TEARDOWN. Primero, el cliente envía una petición ANNOUNCE que contiene información sobre los stream que se quiere enviar, a continuación debe realizar una petición SETUP, estableciendo el valor `receive` para el parámetro `mode`, por cada pista multimedia que se desea enviar (normalmente, uno para el audio y otro para el vídeo), para finalmente enviar una petición RECORD indicando que se va a iniciar la transmisión. La petición TEARDOWN (una por stream, al igual que SETUP) termina la sesión y cierra el stream. Ver la Figura 14 para una representación visual de este escenario.

Este es el escenario habitual cuando el cliente quiere ser el emisor del stream pero no quien lo distribuya, de esta forma envía el streaming a un servidor dado para que este mas tarde lo redistribuya o bien lo guarde en un fichero.

Cuando observamos las diferentes posibilidades para realizar *multihopping* con RTSP en el caso de la transmisión de un stream entre dispositivos A, B, C y D, es decir, de A a B, de B a C y de C a D, se podría hacer de

varias maneras:

La primera forma supone el uso del cliente y servidor en cada nodo de tránsito, usando siempre la modalidad "publicar":

1. Un cliente RTSP de A se conecta en modalidad publicar a un servidor RTSP de B
2. El servidor RTSP de B pasa el stream a un cliente RTSP de B
3. El cliente RTSP del dispositivo B se conecta en modalidad publicar a un servidor RTSP de C
4. El servidor RTSP de C pasa el stream a un cliente RTSP de C
5. El cliente RTSP de C se conecta en modalidad publicar a un servidor RTSP de D

La segunda forma supone el uso del cliente y servidor en cada nodo de tránsito, usando siempre la modalidad "reproducir":

1. Un cliente RTSP de B se conecta en modalidad reproducir al servidor RTSP de A
2. El cliente RTSP de B pasa el stream a un servidor RTSP de B
3. Un cliente RTSP de C se conecta en modalidad reproducir al servidor RTSP de B
4. El cliente RTSP de C pasa el stream a un servidor RTSP de C
5. Un cliente RTSP de D se conecta en modalidad reproducir al servidor RTSP de C

Y una tercera forma con cliente o servidor en cada nodo de tránsito y alternando entre las modalidades "publicar" y "reproducir":

1. Un cliente RTSP de A se conecta en modalidad publicar a un servidor RTSP de B
2. El cliente RTSP 1 de C se conecta en modalidad reproducir a este servidor RTSP de B
3. El cliente RTSP 1 de C pasa el stream a cliente RTSP 2 de C
4. El cliente RTSP 2 de C se conecta en modalidad publicar a un servidor RTSP de D

La mejor solución no es trivial en una aplicación como esta, para algunos de los casos de uso propuestos como el caso de la catástrofe natural y el caso de los derechos humanos por ejemplo queremos que el stream vaya pasando de un dispositivo a otro hasta que llegue a toda la red lo mas rápido posible y es de suponer que los dispositivos actuaran como nodos de transito sin interacción del usuario, por esto la primera forma en la que solo se utiliza la modalidad "publicar" puede parecer mejor, pero en el caso de los derechos humanos donde pueden existir usuarios malintencionados que intenten saturar la red o introducir vídeos con contenido no apropiado se podrían aprovechar de esto por lo que la segunda forma en la que siempre se utiliza la modalidad de "reproducir" puede ser una mejor aproximación dándonos mas posibilidad de decisión en el nodo de transito.

También cabe decir que un implementación mas completa donde se alternan como en la tercera propuesta es muy interesante, tiene la gran ventaja frente a la segunda de poder enviar con modalidad "publicar" el stream desde los nodos de transito, lo que permite enviar el stream a servidores a través de Internet. Además, en una alternativa donde se puedan alternar las modalidades de "publicar" y "reproducir" no siempre tendría que ser de la forma que hemos descrito y es posible que dependiendo de la configuración de la aplicación el transito del stream se efectué de formas muy diferentes, como podrían ser por ejemplo la primera donde siempre se envía utilizando la modalidad de "publicar" o bien la segunda forma donde siempre utilizamos la modalidad de "reproducir" dado que abrimos mucho el abanico de posibilidades.

En el caso de la implementación de WiFi Direct de Android, como se ha explicado en la Sección 5.1.3, solo se pueden hacer dos hops de manera satisfactoria. Para implementar estos dos hops, hemos decidido implementar los primeros dos pasos de la tercera opción (en vez de los primeros tres pasos de la primera o segunda opción).

5.3.6 Problemas para realizar *multihopping* con la librería libstreaming

El primero de los problemas que hemos encontrado con la librería libstreaming para realizar el *multihopping* es que, siendo una librería pensada solo para distribuir los streams generados por la cámara y del micrófono del propio dispositivo, solo implementa la modalidad de "publicar" en el cliente RTSP y es necesario que sea implementada en servidor RTSP también para que esté pueda recibir el stream que le envía el cliente.

El segundo problema que hemos encontrado con la librería libstreaming es que, además, la implementación de la modalidad de "reproducir" en el servidor RTSP de la librería va ligada a sesiones para reproducir un streaming que esta produciendo el dispositivo del servidor en tiempo real

desde la cámara y el micrófono, por lo que es necesario la reimplementación de las peticiones DESCRIBE, SETUP y PLAY cuando el contenido que se esta solicitando no proviene de los periféricos del dispositivo sino de un cliente que esta enviándonos un streaming a nuestro servidor en modalidad "publicar".

Esto además incluye que se tenemos que crear diferentes sesiones para cada tipo de cliente que se conecte al servidor:

- Session para aquellos clientes que desean reproducir el streaming producido por la camara y el micrófono en tiempo real. Esta clase ya esta implementada y apenas deberemos cambiar su implementación sino como se gestionan estas sesiones.
- ReceiveSession para aquellos cliente que desean publicar un streaming en el servidor para ser distribuido.
- RebroadcastSession para aquellos clientes que desean reproducir el streaming que se ha publicado en el servidor por un tercero.

Para retransmitir el streaming que se esta recibiendo en la ReceiveSession y transmitirlo al cliente de una RebroadcastSession crearemos unos servidores UDP que estarán recibiendo los datos de cada uno de los flujos multimedia (audio y vídeo) de una ReceiveSession y reenviándolos a todos los clientes que estén suscritos mediante una RebroadcastSession a esta primera sesión.

Con todo esto y debido a las grandes modificaciones que se deben hacer en el servidor RTSP tanto en las respuestas a las peticiones como en el tratamiento de estas hemos decidido utilizar nuestro paquete de conexiones y crear un nuevo servidor RTSP integrando las funcionalidades del RtspServer proporcionado por la librería y añadiendo las nuevas peticiones necesarias para recibir y retransmitir el streaming así como usar y diferenciar todas las sesiones.

De esta forma implementaremos la clase RTSPServerSelector y RTSPServerWorker para establecer las conexiones y procesar las peticiones RTSP, también implementaremos el UDPServerSelector y el EchoWorker para realizar la retransmisión de los paquetes RTP que se están recibiendo en los puertos UDP. Podemos observar el diagrama de estas clases en la Figura 15

Con esto seremos capaces de crear el primer salto, ya que un dispositivo será capaz de recibir el streaming y reenviarlo pero para la creación de un segundo salto seria necesario realizar mas cambios en el servidor RTSP y cliente RTSP.

Una primera forma seria extender la funcionalidad del servidor RTSP

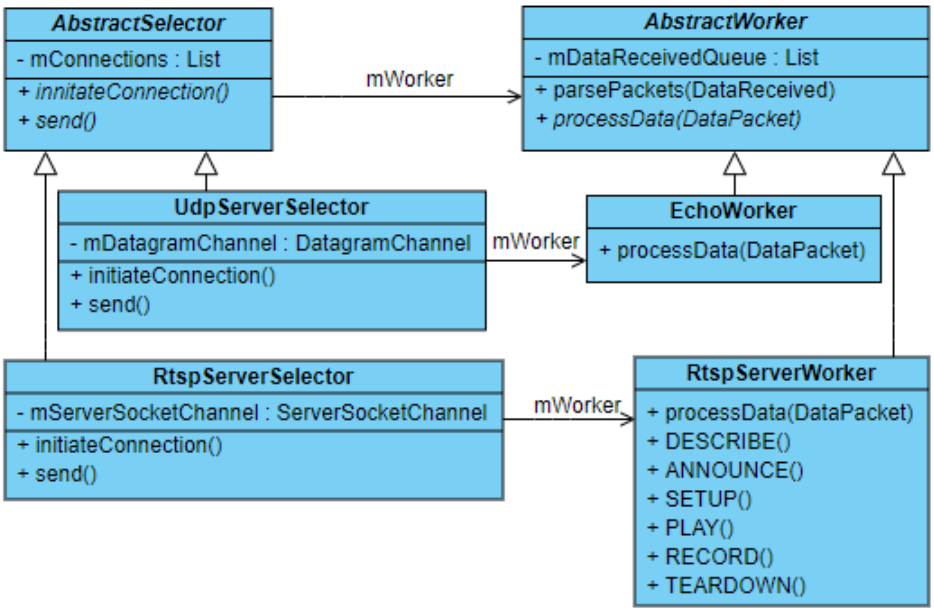

Figure 15: Diagrama de clases en el paquete de conexiones

para que sea capaz de traspasar el streaming a un cliente RTSP en modalidad "publicar" el cual retransmitirá el streaming que el servidor le proporciona. Esto nos permitiría llevar a cabo la primera opción descrita en la Sección 5.3.5.

Para llevar esto a cabo, además de los cambios necesarios en el servidor, para crear este cliente y traspasarle el streaming recibido es necesario realizar una gran cantidad de cambios en el cliente RTSP para poder retransmitir en modalidad "publicar" desde un streaming que no es generado por el propio dispositivo. Por ello es incluso mas práctica la creación de un nuevo cliente que se asociará con una RebroadcastSession en el servidor y utilizará los servidores UDP de la ReceiveSession para realizar el reenvió de datos.

Una segunda forma seria crear un nuevo cliente RTSP el cual fuese capaz de conectarse en modalidad "reproducir" a un servidor RTSP para recibir el streaming, generando una ReceiveSession que mas tarde se traspasará al servidor RTSP. Con esto podríamos llevar a cabo la segunda opción contemplada en la Sección 5.3.5.

Es necesario implementar un nuevo cliente por completo pero cabe decir que ya se han implementado tanto la ReceiveSession que debe generar como los servidores UDP que están asociados a esta. Por lo tanto sin demasiados cambios en el servidor RTSP usaremos este nuevo cliente RTSP para conectarnos a otro servidor y generar una nueva ReceiveSession, a continuación siguiendo el mismo proceso que acabamos de diseñar en esta

sección, el servidor será capaz de retransmitir el streaming de la ReceiveSession cuando un cliente se conecta al servidor.

Implementando uno u ambos cambios seremos capaces de realizar saltos infinitos, en el caso de implementar ambos tendremos mayor versatilidad a la hora de establecer como queremos que estos saltos se realicen dependiendo de las diferentes situaciones.

6 Implementación

6.1 Interfaz gráfica

Para implementar toda la funcionalidad de nuestra aplicación hemos tenido que dividirla en tres actividades, cada una con un rol especifico:

- MainActivity
- StreamActivity
- ViewStreamActivity

ViewStreamActivity es la actividad que se encarga de reproducir un streaming en una IP con ayuda de la librería libVLC. En esta actividad simplemente se fija el origen del stream (IP) y se inicia el MediaPlayer de la librería. Este MediaPlayer lo hemos tenido que configurar nosotros, lo cual al final no has costado bastante trabajo, porque la imagen que nos venía estaba girada y se veía en pequeño en una esquina.

StreamActivity es la actividad que inicia la retransmisión en directo, y dependiendo de si se trata de un cliente de un grupo de WiFi Direct o de su GO, se iniciará el RtspClient o RtspServer. Esta actividad simplemente muestra la vista de nuestra cámara trasera, y un botón para para o iniciar un streaming.

MainActivity es la actividad principal desde la cual se puede invocar a las otras dos. En esta actividad se manejan las conexiones WiFi Direct. Hemos dividido esta actividad en dos pestañas que son WIFI DEVICES y STREAMS AVAILABLE. Para poder fragmentarla en dos pestañas hemos creado las clases FragmeDevices, FragmentStreams y ViewPagerAdapter.

Independientemente de la pestaña en la nos encontremos, en la parte superior de la aplicación hemos puesto cuatro botones que sirven para encender o apagar WiFi, buscar dispositivos WiFi Direct cercanos, transmitir un archivo e iniciar streaming en directo que iniciará la actividad StreamActivity.

En la pestaña WIFI DEVICES 17 hemos implementado que se vea la información de nuestro dispositivo WiFi Direct y las lista de dispositivos WiFi Direct cercanos encontrados. Para implementar esta lista con todos los detalles de cada uno de los dispositivos hemos tenido que implementar la clase DeviceListAdapter. Pinchando en un elemento de la lista se iniciará la conexión entre dispositivos.

También, en esta pestaña , podemos intercambiar mensajes con otros dispositivos conectados. Esta funcionalidad la hemos implementado para

hacer nuestras pruebas durante el desarrollo y hemos decidido dejarla.

En un principio, esta pestaña solo la pensábamos mantear durante el desarrollo, ya que la idea era automatizar las conexiones entre dispositivos. Pero debido a que no hemos podido conseguirlo, finalmente se ha quedado como parte del proyecto.

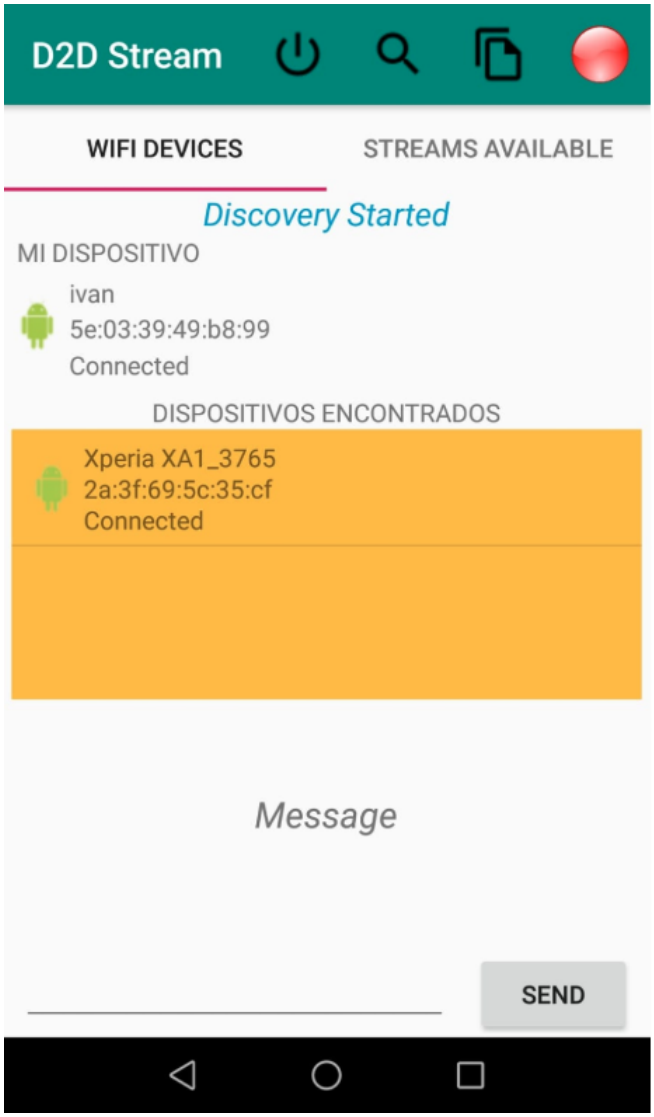

Figure 16: Nuestra aplicación en la pestaña WIFI DEVICES

En la pestaña STREAMS AVAILABLE cada vez que un dispositivo al que se está conectado, ya sea directamente o mediante un GO, inicie un streaming se notificará al fragmento de ello y éste mostrará en una lista la dirección desde la cual se está emitiendo y el nombre del stream.

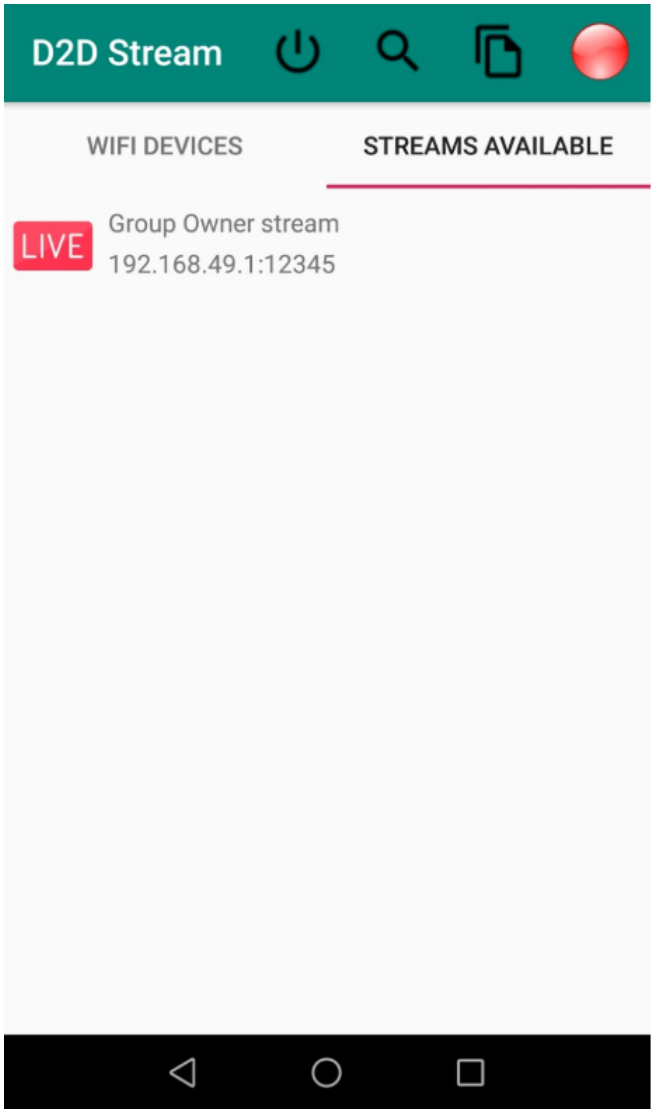

Figure 17: Nuestra aplicación en la pestaña STREAMS AVAILABLE con un streaming en en la lista

Pinchando en un elemento de la lista, se iniciará la actividad ViewStreamActivity que reproducirá la emisión correspondiente. Cuando se termina un stream se elimina el elemento correspondiente de la lista.

Para implementar la lista de streaming disponibles, y darle formato que queremos nosotros, hemos tenido que implementar las clases StreamListAdapter y StreamDetail.

6.2 Permisos

Para poder iniciar la búsqueda de dispositivos WiFi Direct cercanos es necesario que el usuario proporcione a la aplicación el permiso de acceso a la ubicación de dispositivo, al igual que cuando se desea iniciar retransmisión en directo se pedirá los permisos de acceso a la cámara y al micrófono del dispositivo. También tenemos la funcionalidad de mandar un archivo a través de la aplicación con lo cual, el usuario que reciba un archivo antes de poder guardarlo debe proporcionar el permiso de acceo al almacenamiento del dispositivo.

Para todo esto hemos implementado la clase Permissions que se encarga de manejar toda la lógica de lo permisos.

6.3 WiFi Direct

Implementamos la clase WifiP2pController la cual es la encargada de controlar todo lo relacionado con las conexiones de WiFi Direct. Además como es necesario que el Wifi del dispositivo se encuentre activado para poder usar WiFi Direct, también es capaz de activarlo o desactivarlo. Realiza las peticiones a la API de Android para buscar dispositivos a nuestro alcance que también se encuentren buscando emparejamiento de WiFi Direct y conectarnos a ellos. Estas peticiones son asíncronas y el sistema operativo nos responde enviándonos mensajes por medio de Intents a nuestra aplicación sobre el estado de las peticiones realizadas.

Por ello implementamos la clase WiFiP2pBroadcastReceiver la cual extiende de la clase BroadcastReceiver y será registrado en nuestra aplicación para estar constantemente escuchando si ha habido cambios en el estado de WiFi Direct y en el caso afirmativo, según el cambio de estado, se procede a efectuar una acción u otra.

Nuestra idea original era automatizar las conexiones entre dispositivos y así el usuario simplemente tendría que empezar streaming sin preocuparse de tener que conectarse a otro dispositivo por lo que hemos implementado en la clase DiscoveryService un hilo que se encarga de buscar dispositivos con WiFi Direct activo en la zona y se conecten a él. Debido a la gran cantidad de dispositivos que utilizan WiFi Direct como son las impresoras entre otros y que cuando nos conectamos a un dispositivo ya no nos podemos conectar a otro hemos decidido dejar la conexión para que se efectué de forma manual.

6.4 Paquete de conexiones

A continuación implementamos el paquete de conexiones para poder empezar a interactuar con los otros dispositivos a los que nos estábamos conectando mediante WiFi Direct. Para ello creamos las dos clases abstractas AbstractSelector y AbstractWorker junto con las clases auxiliares DataReceived, DataPacket y DataPacketBuilder.

DataReceived es la clase que empaqueta los datos recibidos junto con los datos de la conexión que los ha enviado para que puedan ser procesados por el Worker. Mientras que DataPacket es una clase auxiliar que hemos creado para identificar un paquete de nuestra aplicación, contiene información sobre el tipo de paquete, su longitud y otros parámetros específicos del tipo de paquete además de su contenido.

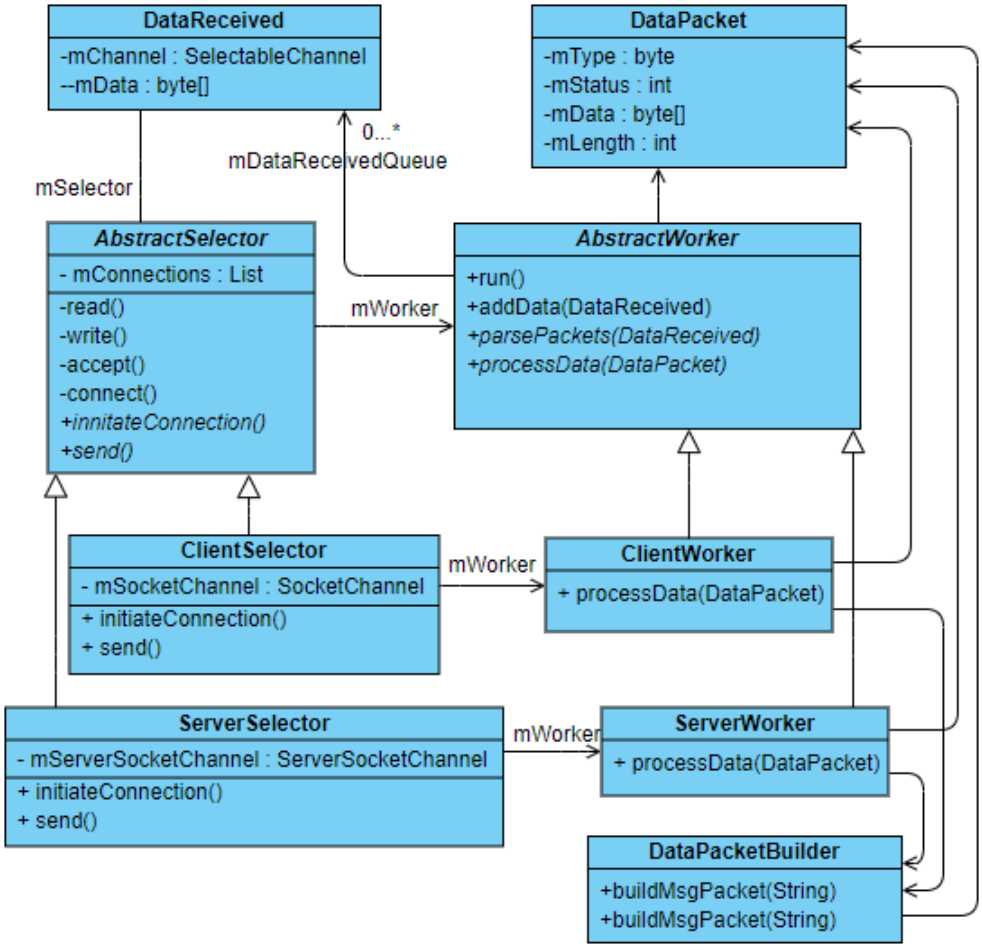

Figure 18: Diagrama de clases del paquete de conexiones

A través del DataPacket el Worker es capaz de saber cuando un paquete esta completo y separarlos si han llegado varios. DataPacketBuilder es la

clase encargada de generar los distintos tipos de DataPacket para enviarlos a los otros dispositivos y será utilizada tanto desde la interfaz grafica como desde los clientes y servidores.

A continuación creamos las implementaciones para el servidor siendo estas el ServerSelector y el ServerWorker. El ServerSelector se encarga de escuchar en el puerto seleccionado y aceptar las nuevas conexiones y el ServerWorker se encarga de procesar los datos recibidos para después reenviarlos a los otros dispositivos de la red y así distribuir el mensaje.

También creamos el ClientSelector y el ClientWorker, donde el primero establece una conexión con la dirección y el puerto dados y el segundo se encarga de procesar los datos recibidos pero esta vez sin reenviarlos a otros dispositivos.

Finalmente antes de empezar con el apartado de streaming creamos un nuevo tipo de paquete para ser capaces de enviar archivos junto con dos clases auxiliares FileHandler y FileSender, la primera es capaz de recibir y guardar un archivo contenido en un DataPacket y la segunda se encarga de leer un archivo, empaquetarlo y enviarlo mediante la conexión designada. De esta forma añadiendo al ClientWorker y ServerWorker el uso del FileHandler ya podíamos enviar archivos entre los dispositivos de la red.

6.5 Streaming

6.5.1 Streaming a múltiples dispositivos simultáneamente

Tras haber estudio la librería a fondo y haber investigado sobre las posibles soluciones para poder hacer streaming de vídeo a múltiples dispositivos simultáneamente, utilizando el RtspServer, sin grandes modificaciones no hemos podido dar con ninguna solución que funcionase. Entonces tuvimos que proponer e implementar una solución nuestra.

Como describimos en la Sección 5.3.2 es necesario conseguir que varios VideoStream puedan recibir los datos de la cámara, pero cada VideoStream intenta obtener una instancia de la cámara y solo es posible tener una instancia por aplicación. Lo mismo ocurre con los AudioStreams y el micrófono, por esto debemos desacoplar esa interacción tan directa entre las clases y los periféricos.

La primera solución que se nos ocurrió y que implementamos fue hacer la camera estática de manera que así todos los VideoStream pudieran compartir la misma instancia de la cámara, y lo mismo para el micrófono y los AudioStream. Estos eran capaces de obtener datos de la cámara y el micrófono simultáneamente, el problema era que cuando un VideoStream leía datos de la cámara y los enviaba, estos datos ya no estaban disponibles

para los otros VideoStream por lo que el resultado es que cada uno solo enviaba pequeños fragmentos de vídeo entrecortados.

El mismo problema descrito con la cámara se aplica al micrófono, es necesario obtener los datos de ambos de una forma externa y después distribuirla entre los distintos VideoStream y AudioStream. Desacoplar el uso de la cámara y el micrófono fue un trabajo bastante laborioso por lo ligados que estaban a las clases. Finalmente lo conseguimos mediante la siguiente solución aunque se han tenido que desactivar algunas interacciones que antes sí podían hacerse con los periféricos.

Por ello para aislar la recogida de datos de estas clases, planteamos implementar unas clases Dispatcher las cuales serán las encargadas de obtener los datos tanto de la cámara como del micrófono y repartirlo entre los distintos streamings activos. Para conseguir esto los streamings deberán suscribirse a estos Dispatchers cuando quieran comenzar la retransmisión.

Se comenzó implementando el Dispatcher para el vídeo, se han realizado dos implementaciones para este y esto es debido a que inicialmente se planteo una clase MediaCodecDispatcher, la cual debía ser capaz de leer los datos de la cámara y transferirlos a varios MediaCodec. La ventaja de esta solución era que para cada stream se podía tener una calidad distinta y además ya se podía hacer stream a más de un dispositivo.

Pero un gran inconveniente que nos echó para atrás con esta solución fue que cuantos más dispositivos se conectaban al RtspServer más lento empezaba a ir el vídeo en cada sesión y esto se debe a que por cada MediaCodec nuevo el móvil debía codificar la imagen y el audio, y no es un proceso ligero y requiere bastante computación por parte del procesador sobre todo con la parte de vídeo. Investigando hemos encontrado que los móviles mas potentes solo podía llegar a tener hasta 8 MediaCodecs trabajando al mismo tiempo. Finalmente tuvimos que descartar esta opción y pensar en algo mejor.

Las nuevas clases principales que creamos para la librería son AudioPacketizerDispatcher y VideoPacketizerDispatcher, estas se encargarán de leer los datos de los periféricos, codificarlos con un MediaCodec y traspasárselos a los Packetizers que se hayan suscrito a ellos. Cada AudioStream y VideoStream al iniciar el streaming llamaran a una función de su respectivo Dispatcher para que este comience a abastecerle con los datos multimedia.

Todos los Packetizer tanto de vídeo como de audio están diseñados para leer los datos de un MediaCodecInputStream, la cual es una clase que extiende de InputStream proporcionando así una interfaz estándar de lectura. Por ello hemos creado una nueva clase ByteBufferInputStream la cual también extiende de InputStream para así tener la misma interfaz de lectura que MediaCodecInputStream y poder reemplazarla. Esta clase es

la encargada de almacenar los datos para los distintos Packetizer.

Hemos creado una clase MediaCodecBufferReader la cual implementa un hilo para leer siempre que se disponga de nuevos datos de un periférico a través de su MediaCodecInputStream y copiarlos a todos los ByteBufferInputStream que tiene asignados en una lista. Cada Dispatcher al iniciarse creara un MediaCodecBufferReader para leer de la cámara o micrófono.

Ahora por cada nueva sesión las clases AudioStream y VideoStream deben suscribir sus Packetizers (son los encargados de crear los paquetes RTP y enviarlos) a los respectivos Dispatchers. Los Dispatchers crean un nuevo ByteBufferInputStream por cada Packetizer y lo asignan al mismo como fuente de los datos donde antes se le asignaba un MediaCodecInputStream.

Además se añade el ByteBufferInputStream a la lista del MediaCodecBufferReader de tal forma que este ultimo empieza a copiar los datos del MediaCodec en el ByteBufferInputStream y el Packetizer puede empezar a enviar los paquetes y de esta manera conseguimos que todos los Packetizer tengan los datos multimedia sin tener que acceder a la cámara y el micrófono múltiples veces.

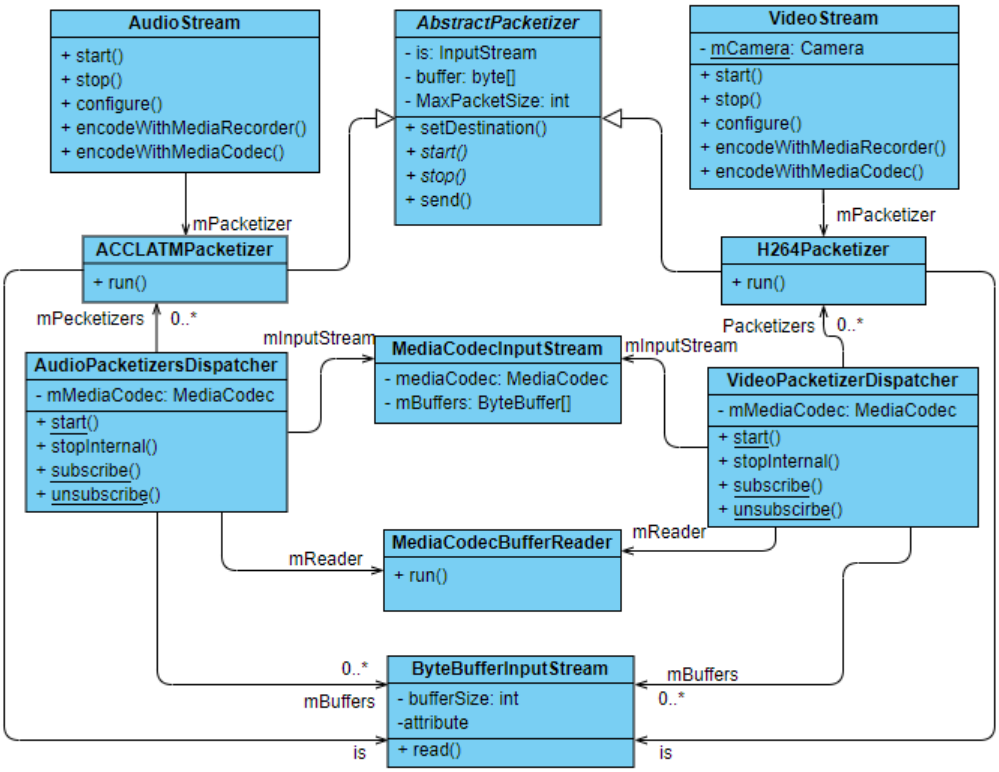

Figure 19: Diagrama de clases de la solución multistreaming

Cuando los Dispatchers se quedan sin Packetizers, es decir, no hay

clientes viendo el streaming en directo, se paran los MediaCodec y se cierran los distintos hilos que leen de estos para así liberar los recursos, a continuación se procede a terminar los Dispatchers, hasta que un nuevo cliente se conecte, evitando de esta manera gasto de recursos innecesario.

Con esta solución evitamos el problema de múltiples accesos a la cámara y el micrófono o creación de múltiples MediaCodecs y conseguimos hacer el streaming a múltiples dispositivos simultáneamente. El único inconveniente es que la transmisión en directo se hace a un calidad fija para todos los clientes aunque eso no es un problema para los casos de uso propuestos.

En la Figura 19 se muestra el diagrama de clases de esta solución. En el se muestran las nuevas clases nuevas y con las que estás interactúan. Para ver el diagrama inicial de la librería véase la Figura 9.

6.5.2 Retransmisión del streaming mediante MultiHop

Para llevar acabo la solucion propuesta el Sección 5.3.6, lo primero que hemos hecho ha sido implementar un nuevo servidor RTSP con nuestro paquete de conexiones basándonos en gran parte en el RtspServer de la librería, para ello creamos las clases RTSPServerSelector y RTSPWorker donde se controlan las conexiones y procesan las peticiones respectivamente.

Lo siguiente a implementar es la ReceiveSession y el servidor UDP para poder empezar a recibir la transmisión de un cliente en el servidor. Para ello, desarrollamos las nuevas clases UDPServerSelector y EchoWorker, donde el selector pone un socket UDP a la escucha en un puerto dado y permite que suscribamos direcciones y puertos a las que el EchoWorker reenviará cada vez que se reciban datos para de esta forma retransmitir los flujos multimedia que estamos recibiendo a otros clientes.

Para poder crear los servidores UDP para reenviar los flujos del streaming hemos realizado cambios tanto en el AbstractSelector como en el AbstractWorker, de tal forma que estos pudiesen trabajar tanto con los SocketChannel, que es la clase para los sockets TCP en NIO, como con los DatagramChannel, que son los designados para el protocolo UDP. Para ello era necesario modificar muchas de las operaciones básicas del AbstractSelector como las de recibir datos o enviarlos para identificar el tipo de Channel usado.

A continuación, hemos tenido que crear una implementación del AbstractSelector que es el UDPServerSelector el cual crea un servidor a la escucha en un puerto UDP y además es capaz de tener un listado de conexiones UDP a las que se enviaran los paquetes. También hemos desarrollado una implementación de la clase AbstractWorker que es EchoWorker. Este worker no procesa los paquetes y simplemente reenvía todos los datos que

le lleguen a todos los clientes que estén suscritos al UDPServerSelector. De esta forma podemos crear un servidor en el cual recibiremos los flujos multimedia y al cual podremos suscribir clientes que quieren recibir estos flujos para que sean reenviados.

Creamos una clase ReceiveSession la cual es capaz de contener la información de sesión y de cada pista multimedia creando nuevas TrackInfo para cada una, estas TrackInfo almacenan la descripción de los flujos multimedia a los que corresponden y son capaces de comenzar los UDPServerSelector en los puertos en lo que se van a recibir estas pistas para de esta forma poder empezar la recepción de los datos cuando se inicie la ReceiveSession.

Después es necesario reimplementar el procesamiento de peticiones del servidor para que procese las peticiones que realiza el cliente al publicar el streaming y pueda crear una nueva ReceiveSession, la primera petición a implementar es ANNOUNCE porque es con la que se inicia la sesión, en nuestro diseño para la gestión de múltiples streamings hemos decidido que la petición del ANNOUNCE debe ser llamada de forma que se identifica la sesión por la ruta de la petición (Ej: rtsp://192.168.1.49/customSession), de esta forma se crea una nueva ReceiveSession asociada a esa ruta y se almacenan las partes necesarias de la petición ANNOUNCE donde se describen la sesión y los flujos multimedia van a ser publicados para que luego mas tarde podamos crear las peticiones de retransmisión de forma correcta. Si todo es correcto el servidor almacena la sesión y procede a responder con un OK.

Después es necesario reimplementar la petición SETUP ya que esta solo esta disponible para mode=play y al publicar un cliente la envía con el parámetro mode=receive, como mapeamos las conexiones del servidor RTSP con las sesiones esto se resuelve de forma sencilla diferenciando que tipo de sesión tiene esa conexión para de esta forma si es una ReceiveSession invocar la nueva implementación. En esta nueva petición SETUP guardaremos los puertos a los que el cliente va a enviar el flujo multimedia de vídeo o audio y los asignaremos a la ReceiveSession asociada para mas tarde poder controlar las pistas.

Por ultimo para el flujo del servidor es necesario implementar la función RECORD donde se nos indica que se van a comenzar a enviar los flujos multimedia a los puertos establecidos en las peticiones anteriores, en este momento se inician los servidores UDP de los distintos TrackInfo de la ReceiveSession aunque todavía es necesario implementar la RebroadcastSession y las peticiones asociadas con ella para poder retransmitir el streaming que estamos recibiendo.

Primero creamos una RebroadcastSession la cual es capaz de almacenar la información del cliente que esta recibiendo el streaming así como de

cada uno de los flujos que se van a retransmitir desde la ReceiveSession que lleva asociada, es la sesión encargada de que los servidores UDP de la ReceiveSession comiencen o dejen de reenviar los datos al cliente asociado a la sesión. Esta sesión también utiliza la ayuda de una pequeña clase auxiliar RebroadcastTrackInfo donde almacena la información de cada flujo.

A continuación es necesario reimplementar cada una de las peticiones DESCRIBE, SETUP(mode=play) y PLAY, porque a pesar de que ya venían implementadas en el servidor proporcionado por la librería estas trabajan directamente con la sesión que crea al realizar un streaming desde la cámara y micrófono. Empezamos reimplementando la petición DESCRIBE, esta igual que el ANNOUNCE viene asociada con una ruta para identificar que ReceiveSession se desea reproducir, en caso de estar vacío se intentara crear una sesión para el streaming en tiempo real de nuestro dispositivo. En caso contrario se creara una RebroadcastSession y se la asociara con la ReceiveSession correspondiente, y se creara una nueva descripción de sesión utilizando los datos de los flujos almacenados anteriormente en la ReceiveSession que sera enviada al cliente.

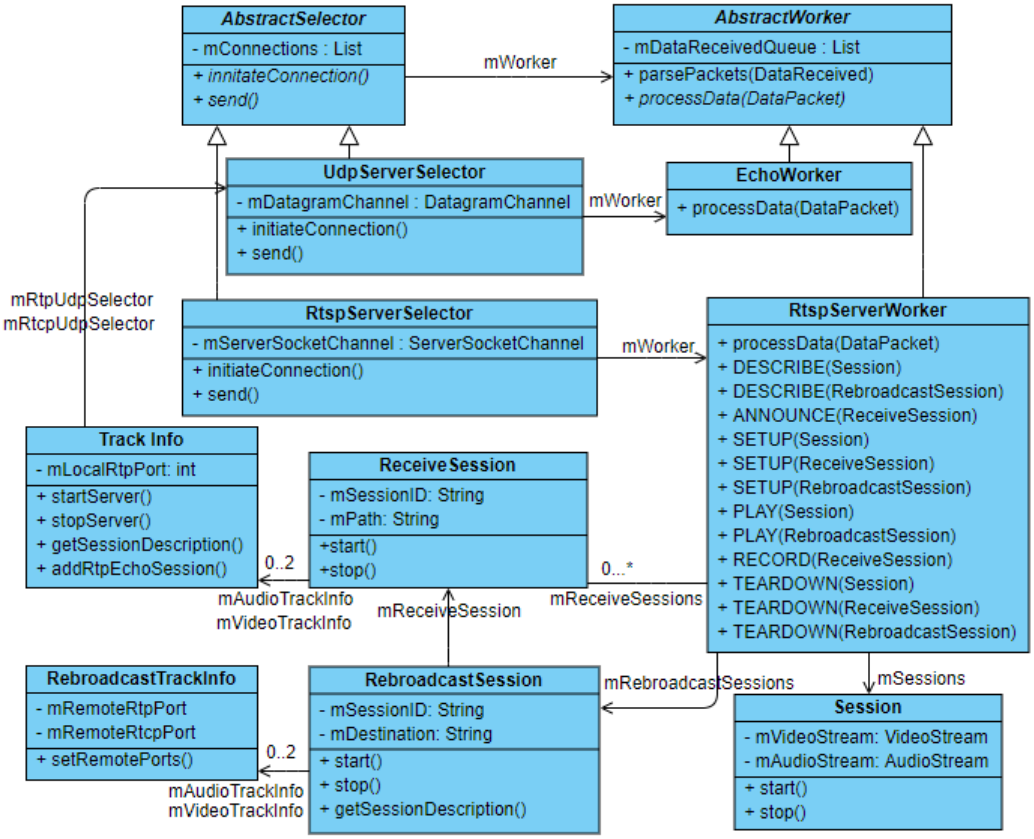

Figure 20: Diagrama de clases de la solución multistreaming

Después ha sido necesario reimplementar la petición SETUP cuando

es llamada por una RebroadcastSession, en esta petición estableceremos los puertos a los que enviaremos los flujos multimedia y configuraremos las RebroadcastTrackInfo de acuerdo con esto para posteriormente comenzar a reenviar el contenido. Finalmente cuando el cliente asociado a la RebroadcastSession realice la petición PLAY, una nueva implementación de esta petición suscribirá las direcciones y puertos asociadas con los flujos a los servidores UDP de la ReceiveSession de tal forma que el cliente comenzara a recibir el streaming.

También ha sido necesario reimplementar la petición TEARDOWN para cada una de las sesiones de tal forma que si la envía una RebroadcastSession esta cancele la suscripción a la ReceiveSession y deje de reenviar los flujos multimedia. En el caso de la ReceiveSession además de cerrar los servidores UDP dado que el cliente dejara de emitir el streaming ha de encargarse de cerrar todas las RebroadcastSessions que estaban asociadas a esa sesión pues se ha terminado el streaming que estaban visualizando.

Debido a nuestra limitación a una pequeña red de WiFi Direct por los motivos mencionados en la Sección 5.1.3, hemos establecido un sistema en el cual si el emisor del streaming es un cliente conectado al Group Owner, le enviará el streaming para que esté lo distribuya al resto de clientes y si el emisor es el Group Owner entonces este simplemente retransmitirá simultáneamente a todos los clientes.

7 Conclusión

La idea de crear redes *ad hoc* en dispositivos móviles que se encuentren cerca unos de otros sin tener que pasar por una red de los operadores de telecomunicaciones, y hacer streaming de vídeo en directo a través de dicha red, es algo ambicioso y difícil de llevar a cabo.

Nosotros en este proyecto hemos intentado por todos los medios posibles que esta idea se pudiese llevar a cabo analizando los distintos tipos de dispositivos móviles, cuáles serían los más adecuados, y qué sistema operativo sería la mejor opción para hacer posible esta idea.

Tras habernos decidido por los dispositivos móviles con sistema operativo Android, ya que éstos son los más comunes y los más usados, con un 85.9% del total de ventas de móviles en 2017 frente a un 16% de iOS y 0.1% otros, según un reportaje [23], nos hemos puesto a investigar las posibles tecnologías de conexión existentes que podríamos usar para la realización de este proyecto.

Tras la tarea de investigación dudábamos entre dos opciones posibles, WiFi Direct o WiFi Aware. Pero debido a que la segunda opción es algo bastante novedoso, la falta de la documentación sobre ésta, y la falta de disponibilidad de dispositivos con esta tecnología para la realización de pruebas durante el desarrollo, finalmente hemos optado por la primera.

Al estudiar WiFi Direct más a fondo descubrimos que es una tecnología cuya implementación en Android presenta ciertas limitaciones (explicadas en la Sección 5.1) las cuales nos impiden cumplir algunos de los objetivos importantes del proyecto, entre ellos el de creación de la red *ad hoc*. Propusimos algunas soluciones encontradas en distintos artículos acerca de la creación de redes *ad hoc* sobre WiFi Direct, pero ninguna nos pareció viable para este proyecto.

Estas limitaciones hacen que sólo podamos interconectar un grupo de dispositivos que se encuentren cerca unos de otros, y por tanto hacer el streaming de vídeo sobre una red pequeña.

Por otra parte, la emisión de vídeo en directo también fue un reto bastante grande que tuvimos que abordar. Para ello nos decantamos por el uso de los protocolos RTSP y RTP/RTCP en vez HTTP, a pesar de que HTTP streaming es el más popular actualmente, ya que las ventajas de HTTP streaming no son relevantes en nuestro caso, como explicamos en la sección 3.5.

Tras un proceso de investigación bastante largo, hemos encontrado e incluido en nuestro proyecto una librería de código abierto cuyas características y limitaciones exponemos detalladamente en la sección 5.3.

Concretamente, la principal limitación observada en la librería durante la fase de diseño, y explicada en el apartado 5.3.2, es que no se podía hacer streaming a múltiples dispositivos simultáneamente, debido a que cada retransmisión bloqueaba la cámara y el micrófono. Para solucionar este problema, hicimos cambios en la librería para que un dispositivo pueda emitir en directo a múltiples dispositivos a la vez accediendo una única vez a los periféricos. Explicamos en detalle este arreglo en el apartado 6.5.1 de la sección Implementación.

Aunque por las limitaciones de WiFi Direct no es posible la creación de la red *ad hoc*, en la sección 5.3.4 de Diseño hemos intentado simular el escenario de una red *ad hoc* donde el streaming se distribuye saltando de un dispositivo a otro, y estudiado las diferentes alternativas de las que disponíamos para ello.

Cuando planteamos simular este escenario con la librería nos encontramos con los problemas descritos en la sección 5.3.6. El problema es que la librería estaba únicamente pensada para poder retransmitir el streaming generado por el propio dispositivo. Por ello, hemos reimplementado el servidor, y desarrollado nuevas funcionalidades para poder recibir y enviar streamings, el desarrollo de las cuales se detalla en la sección 6.5.2. Esto nos ha permitido crear nodos de tránsito capaces de reenviar el streaming al mismo tiempo que lo reciben. De esta forma somos capaces de distribuir el streaming por nuestro grupo de WiFi Direct, saltando de un dispositivo a otro.

De esta manera nos acercamos más a la solución que deseábamos desde un principio, ya que de ser posible en un futuro la existencia de un "nodo de enlace" entre dos grupos WiFi Direct, o bien la posibilidad de crear una red *ad hoc* mediante otra nueva tecnología, podríamos reutilizar la solución que hemos desarrollado, y distribuir el streaming por la red utilizando este método.

Resumiendo, hemos desarrollado una aplicación desde la cual podemos crear conexiones WiFi Direct –limitando éstas a un solo grupo de WiFi Direct de dispositivos cercanos– a través de las cuales podemos hacer streaming de vídeo en directo en dos modalidades distintas. La primera modalidad es Multistreaming, consistente en realizar streaming simultáneamente a varios dispositivos. La segunda modalidad es Multihop, que consiste en que un cliente emite a un servidor, y este último se encarga que redistribuirlo a los dispositivos que están conectados con él. La aplicación también recibe un listado de streamings disponibles, y podemos reproducirlos directamente en el móvil pinchando sobre uno de nuestra elección. Para esta funcionalidad usamos otra librería, libvlc.

7 Conclusion

The idea of creating *ad hoc* networks on mobile devices that are close to each other without having to go through a network of telecommunication operators, and to stream live video through that network, is a goal ambitious and difficult to carry out.

In this project we have tried by all possible means to carry out this idea, by analyzing the different types of mobile devices, which would be the most appropriate, and which operating system would be the best option to make this idea possible.

After having decided on mobile devices with Android operating system, since these are the most common and most used, with 85.9% of total mobile sales in 2017 compared to 16% of iOS and 0.1% other , according to a report [23], we have started to investigate the possible existing connection technologies that we could use to carry out this project.

After the research task we were hesitating between two possible options, WiFi Direct or WiFi Aware. But because the second option is something quite new, the lack of documentation on it, and the lack of availability of devices with this technology for testing during development, we have finally opted for the first.

When studying WiFi Direct more thoroughly we discovered that it is a technology whose implementation in Android has certain limitations (explained in Section 5.1) which prevent us from fulfilling some of the important objectives of the project, among them the creation of the red *ad hoc*. We proposed some solutions found in different articles about the creation of networks *ad hoc* on WiFi Direct, but none of them seemed viable for this project.

These limitations mean that we can only interconnect a group of devices that are close to each other, and therefore wen can only make the video streaming over a small network.

On the other hand, the live video broadcast was also a very big challenge that we had to address. For this we have opted for the use of the RTSP and RTP / RTCP protocols instead of HTTP, although HTTP streaming is currently the most popular, since the advantages of HTTP streaming are not relevant in our case, as we explained in the section 3.5.

After a quite long research process, we have found and included in our project an open source library whose characteristics and limitations we expose in detail in the section 5.3.

Specifically, the main limitation observed in the library during the design

phase, and explained in the section 5.3.2, is that one cannot stream to multiple devices simultaneously, because each retransmission blocked the camera and microphone. To solve this problem, we made changes in the library so that a device can broadcast live to multiple devices at once by accessing the peripherals only once. We explain this arrangement in detail in the 6.5.1 section of the Implementation section.

Although due to the limitations of WiFi Direct it is not possible to create the *ad hoc* network, in the 5.3.4 section of Design we have tried to simulate the scenario of an *ad hoc* network where streaming is distributes jumping from one device to another, and studied the different alternatives that we had for it.

When we set out to simulate this scenario with the library, we find the problems described in the section 5.3.6. The problem is that the library was only designed to be able to retransmit the streaming generated by the device itself. For this reason, we have reimplemented the server, and developed new functionalities to receive and send streamings, the development of which is detailed in the section 6.5.2. This has allowed us to create transit nodes capable of forwarding the streaming at the same time they receive it. In this way we are able to distribute the streaming all along our WiFi Direct group, jumping from one device to another.

In this way we get closer to the solution we aimed at from the beginning, because if it were possible in the future the existence of a "link node" between two WiFi Direct groups, or the possibility of creating an *ad hoc* network by means of another new technology, it should be possible to reuse the solution we have developed, and distribute the streaming over the network using this method.

In short, we have developed an application from which we can create WiFi Direct connections - limiting these to a single group of WiFi Direct of nearby devices - through which we can stream live video in two different modes. The first modality is Multistreaming, consisting of simultaneous streaming to several devices. The second mode is Multihop, which consists of a client issuing to a server, where the latter is responsible for redistributing it to the devices that are connected to it. The application also receives a list of available streams, and we can reproduce them directly on the mobile by clicking on one of our choice. For this functionality we use another library, libvlc.

8 Future Work

Uno de nuestro principales objetivos fue la creación de una red *ad hoc*, el cual no hemos podido llevar a cabo con WiFi Direct. Por tanto una de las cosas que dejamos como trabajo a futuro es la investigación e implementación de un modulo con otras posibles tecnologías de conexión, como por ejemplo WiFi Aware.

Otro tema muy importante que se deja como trabajo a futuro es la implementación de medidas de seguridad para los vídeos falsos ("deepfakes"), los cuales pueden ser modificados en tiempo real mientras pasan de un dispositivo a otro, por lo cual nosotros también deberemos efectuar un análisis de éstos en tiempo real, así como medidas para el filtrado de vídeos con contenido inapropiado que pudieran ser introducido. Otro de los posibles ataques a tener en cuenta es el de nodos que intenten interrumpir las retransmisiones de los streamings, así como nodos que se dediquen a intentan rastrear los streamings en busca del emisor original.

Otra posibilidad que podemos destacar para su estudio más adelante es la de incluir en nuestra aplicación la opción de retransmitir vídeo a un servidor a través de redes convencionales, integrándolo con aplicaciones externas; un ejemplo para los casos de uso propuestos podría ser la integración con aplicaciones de Witness[1].

Asimismo, se podrían incluir en la aplicación varias opciones, como son la de modificar la calidad de vídeo a la que estamos retransmitiendo, cambiar entre la cámara trasera y la frontal, y la posibilidad de asignar un titulo al streaming, o que se muestren la fecha y la hora de la transmisión en una esquina.

Algo mas a tener en cuenta es que se debería llevar a cabo una validación mas consistente del proyecto, incluyendo tests unitarios que verifiquen automáticamente que el comportamiento de la aplicación es correcto. En una aplicación como ésta es muy difícil llevar a cabo este tipo de validación ya que la aplicación debe correr en múltiples dispositivos al mismo tiempo conectándose por "redes no convencionales", motivo por el cual también se dan problemas a la hora de emular estos dispositivos, y muchas veces el testeo tiene que llevarse a cabo con dispositivos reales.

Sería interesante que la aplicación sea capaz de identificar a otros dispositivos que tengan instalada la aplicación, con la posibilidad de conectarse a ellos.

9 Aportación de los participantes

En esta sección vamos a resumir los detalles de las aportaciones de cada uno de los miembros del equipo durante el desarrollo de nuestra aplicación Device to Device Streaming.

En general, el proyecto se ha realizado conjuntamente, participando ambos en todo en todos los apartados y fases del proyecto, aunque cada participante ha aportado más en unos aspectos o en otros.

Para la repartición del trabajo de forma equitativa además de seguir una metodología ágil hemos utilizado el método Kanban junto con reuniones semanales programadas tanto entre nosotros como con el director del trabajo Simon Pickin.

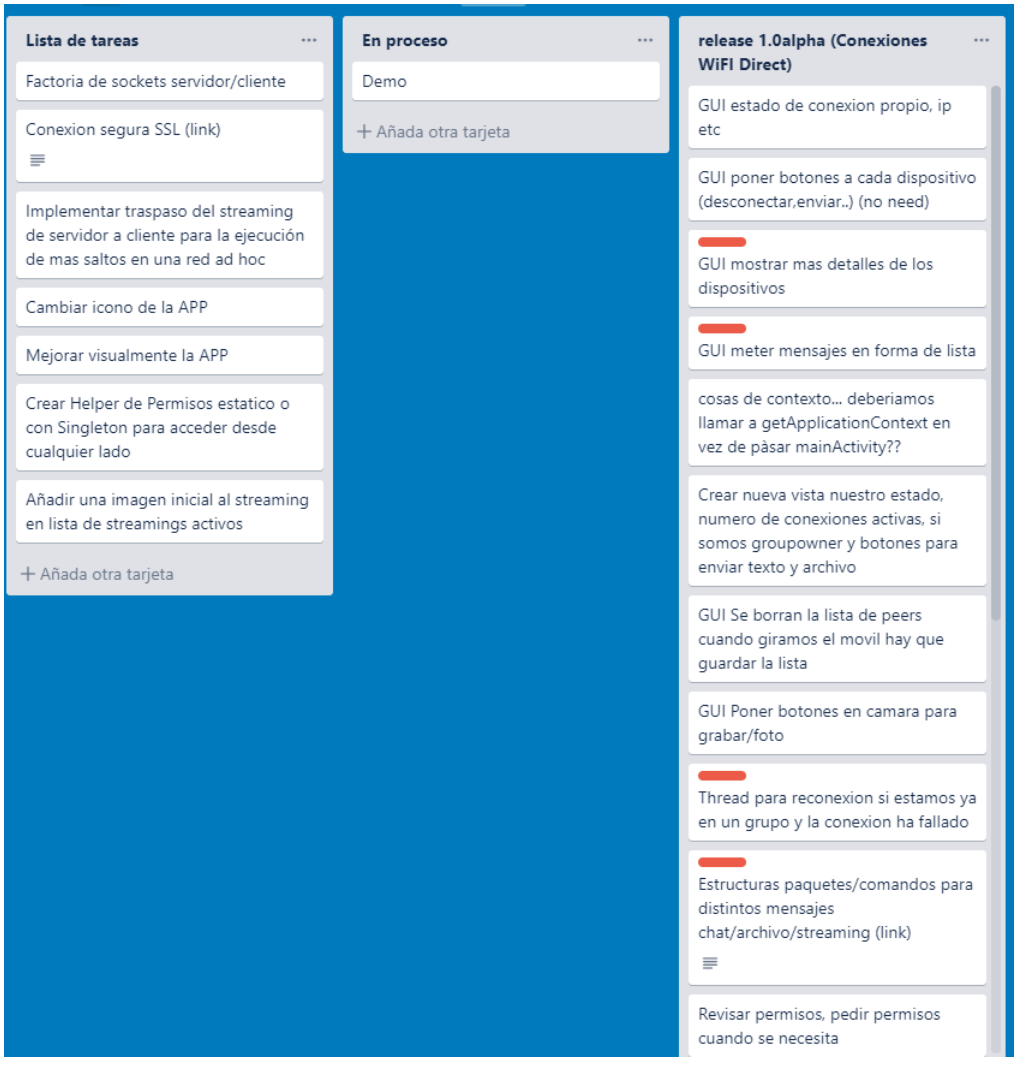

Figure 21: Kanban de Trello del proyecto hasta release 1.0

Hemos utilizado la aplicación web Trello [17] para usar el método Kanban, la cual permite crear tableros y tarjetas virtuales, siendo cada una de estas una tarea pendiente, en proceso o completada.

En la Figura 21 y la Figura 22 se puede observar como ha quedado el tablero al final del proyecto. En las figuras se puede observar que hemos partido el proyecto en dos sprints, uno dedicado a las conexiones WiFi Direct y otro orientado sobre todo a Strreaming de video.

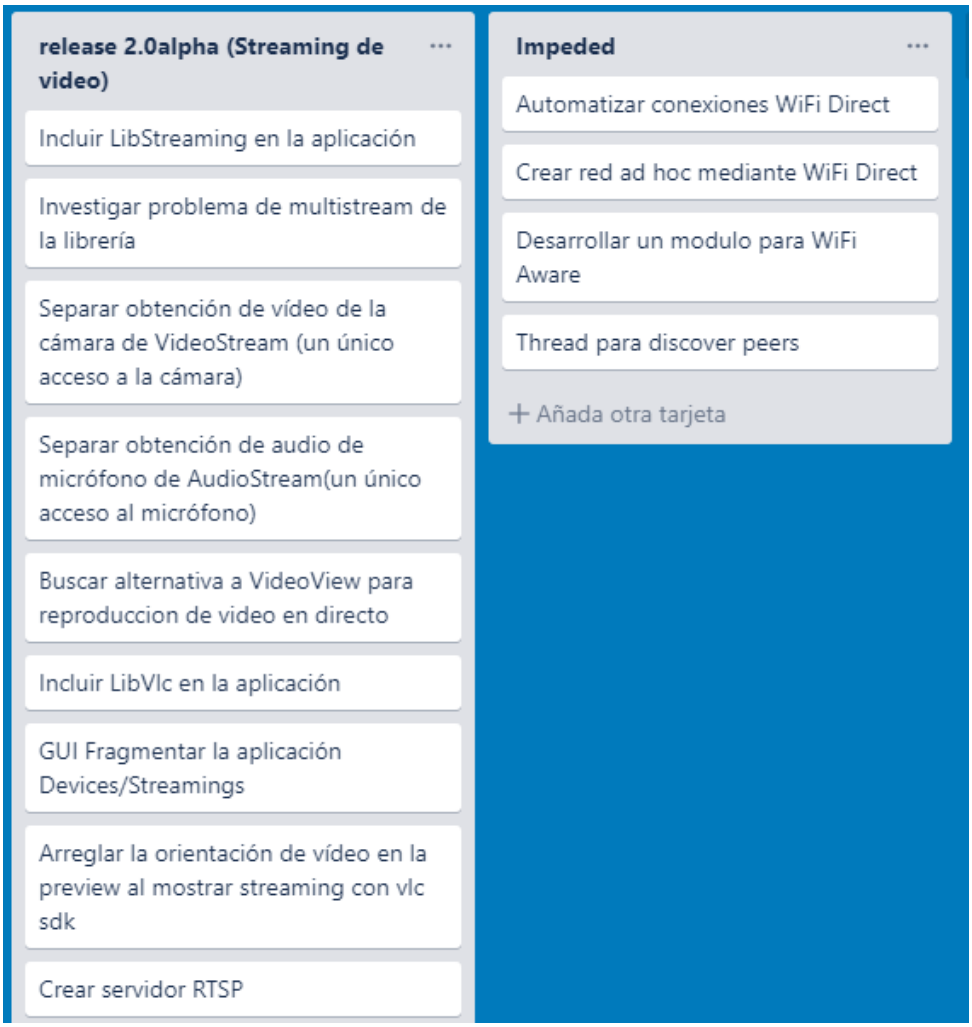

Figure 22: Kanban de Trello del proyecto

9.1 Noel José Algora Igual

9.1.1 Investigación

- Estudio de herramientas y lenguajes para aplicaciones Android: Android Studio, Java, XML y Apis de Android.
- Estudio de tecnologías para realizar conexiones peer-to-peer: WiFi Direct, WiFi Aware, WiFi AdHoc Mode, Bluetooth...
- Investigación sobre aplicaciones parecidas
- Estudio de tecnologías para la retransmisión de streamings multimedia: HTTP, RTSP, Adaptative bitrate.
- Selección de tecnologías e investigación exhaustiva de ellas tanto como de las librerías y APIs diponibles.

9.1.2 Desarrollo

- Implementación del modulo de conexión sobre WiFi Direct.
- Apoyo implementación de la interfaz grafica.
- Implementación del paquete de conexiones.
- Implementación Dispatchers y clases auxiliares.
- Implementación Dispatcher de Video.
- Implementación de *multihopping* para la libreria libstreaming.

9.1.3 Memoria

- Introducción
- Antecedentes y Elección de tecnologías
- Diseño e implementación
- Conclusión y Future Work
- Diagramas de clase para para los apartados de Multihop.
- Diagramas de dispositivos WiFi Direct y Multihop.

9.2 Ivan Gulyk

9.2.1 Investigación

- Estudio de herramientas y lenguajes para aplicaciones Android: Android Studio, Java, XML y Apis de Android.
- Estudio de tecnologías para realizar conexiones peer-to-peer: WiFi Direct, WiFi Aware, WiFi AdHoc Mode, Bluetooth...
- Investigación sobre aplicaciones parecidas
- Estudio de tecnologías para la retransmisión de streamings multimedia: HTTP, RTSP, Adaptative bitrate.
- Selección de tecnologías e investigación exhaustiva de ellas tanto como de las librerías y APIs diponibles.

9.2.2 Desarrollo

- Implementación del modulo de conexión sobre WiFi Direct.
- Implementación del modulo de los permisos.
- Implementación de la interfaz gráfica.
- Ayuda a la implementación de paquete de conexiones.
- Apoyo a la implementación de los Dispatchers.
- Implementación del AudioDispatcher.

9.2.3 Memoria

- Introducción
- Antecedentes y Elección de tecnologías
- Diseño e implementación
- Conclusión y Future Work
- Diagrama de clase para la librería libstreaming con la funcionalidad Multisesión
- Diagrama de clase de la librería libstreaming en su estado inicial

Bibliografía

[1] *WITNESS*. URL: https://witness.org/.

[2] PABLO GARRIDO HYEONGWOO KIM and Other. *Deep Video Portraits*. May 2018. URL: https://arxiv.org/pdf/1805.11714.pdf.

[3] McClatchy Washington Bureau Tim Johnson and Other. *DARPA Is Racing To Develop Tech That Can Identify Hoax Videos*. June 2018. URL: https://taskandpurpose.com/deepfakes-hoax-videos-darpa.

[4] *¿Qué es el Bluetooth y para qué sirve?* URL: http://www.valortop.com/blog/bluetooth.

[5] *Wi-Fi Direct*. URL: https://www.wi-fi.org/discover-wi-fi/wi-fi-direct.

[6] *Wi-Fi Aware*. URL: https://www.wi-fi.org/discover-wi-fi/wi-fi-aware.

[7] *RTP: A Transport Protocol for Real-Time Applications*. URL: https://tools.ietf.org/html/rfc3550.

[8] *Symmetric RTP / RTP Control Protocol (RTCP)*. URL: https://tools.ietf.org/html/rfc4961.

[9] *Real-Time Streaming Protocol Version 2.0 Internet Engineering Task Force (IETF)*. URL: https://tools.ietf.org/html/rfc7826.

[10] *Cam Wimote*. URL: https://play.google.com/store/apps/details?id=com.ftchan.android.camwimote&hl=es.

[11] MIV Dev. *RTSP Camera Server*. URL: https://play.google.com/store/apps/details?id=com.miv.rtspcamera&hl=es.

[12] *Statcounter Mobile Share*. URL: http://gs.statcounter.com/os-market-share/mobile/worldwide.

[13] *libstreaming*. URL: https://github.com/fyhertz/libstreaming.

[14] *LibVLC*. URL: https://wiki.videolan.org/LibVLC/.

[15] *VLC*. URL: https://www.videolan.org/.

[16] *Gradle*. URL: https://es.wikipedia.org/wiki/Gradle.

[17] *Trello*. URL: https://trello.com.

[18] *LaTeX*. URL: https://www.latex-project.org/.

[19] *Overleaf - Online LaTeX Editor*. URL: https://www.overleaf.com/.

[20] Wi-Fi Alliance. *Wi-Fi CERTIFIED Wi-Fi Direct*. Oct. 2010. URL: https://www.wi-fi.org/system/files/wp_Wi-Fi_Direct_20101025_Industry.pdf.

[21] C. Funai, C. Tapparello, and W. Heinzelman. "Enabling multi-hop ad hoc networks through WiFi Direct multi-group networking". In: *2017 International Conference on Computing, Networking and Communications (ICNC)*. Jan. 2017, pp. 491–497. DOI: 10.1109/ICCNC.2017.7876178. URL: https://ieeexplore.ieee.org/document/7876178.

[22] *Infrastructure-Less Communication Platform for Off-The-Shelf Android Smartphones*. Jan. 2018. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5876628/.

[23] Juan Antonio Pascual. *Android vs iPhone: la guerra de los smartphones en cifras*. July 2018. URL: https://computerhoy.com/reportajes/industria/android-vs-iphone-guerra-smartphones-cifras-271447.