Universidad Complutense de Madrid
FACULTAD DE INFORMÁTICA

Grado en Ingeniería de Software

TRABAJO DE FIN DE GRADO

P2P Sharing:
Aplicación android P2P
para compartición de archivos

Autores:

Jesús Perucha Pérez

Josué Pradas Sacristán

Directores:

Simon Pickin

Pablo Rabanal

Junio 2018

I. Resumen

Desde el principio en el que leímos el concepto de P2P asociado a una aplicación móvil, en Android en este caso, pensamos en cómo a lo largo de nuestra vida hemos escuchado y usado aplicaciones para ordenador en las que descargamos archivos gracias a la tecnología P2P. El P2P y Android nos parecían unos campos muy interesantes en el que adentrarnos, primero para saber cómo funciona el concepto de P2P y luego para poder aprender a desarrollar una aplicación Android y quizás hasta publicarla en el store para compartir este trabajo con el mundo.

Por este motivo de difusión y aprendizaje académico, queremos que este TFG sea un punto de partida y quizás hasta una guía de referencia para futuros trabajos sobre P2P en los que haya que crear una aplicación en la que se quiera intercambiar archivos. Aunque hay muchas tecnologías que permiten el intercambio de archivos, creemos que el concepto de P2P, aun siendo muy antiguo, tiene recorrido y futuro a nivel tecnológico y sobre todo es un concepto que la mayoría de gente, aun siendo desconocedora de cómo funciona, asocia a intercambio libre de archivos, sin barreras o impedimentos, como una forma de luchar contra la censura que muchos gobiernos o instituciones han impuesto o podrían implementar.

Este TFG supone una inmersión dentro del mundo de las redes, pues aunque trate hacer una aplicación Android con tecnología P2P, el concepto de P2P no se puede entender si el concepto de red. Por ello hay que tener claros los distintos tipos de redes que hay y que aporta cada tipo. Además a nivel técnico han surgido algunos problemas que son inherentes a las estructuras y protocolos de red que se usan en el dia a dia, por ello hemos tenido que sumergirnos en ese mundo e investigar a fondo para salir con una solución concreta.

Se ha desarrollado una aplicación Android con la que dos personas pueden agregarse como amigos, subir los archivos que crean merecedores de compartir y ver los archivos que cada amigo suyo ha subido para poder descargarlos. No pretendemos que la aplicación Android sea un todo en uno que permita un intercambio muy eficiente, rápido y seguro, de tal

forma que se esté creando la una aplicación perfecta. Simplemente llegar a los objetivos propuestos y aprender en el proceso.

En definitiva, el proyecto muestra todo el proceso de aprendizaje realizado y como se ha pasado del desconocimiento de los temas investigados a generar una aplicación Android funcional que podemos usar para intercambiar archivos bajo cualquier tipo de conexión a internet.

Palabras Clave

P2P, Intercambio, Archivos, App, Android, WebRTC, PubNub

I. Abstract

From the moment we read about how P2P concept is associated to a mobile app, Android in this case, we thought in how throughout our life we have heard and used computer apps in which we download archives because of P2P technology. P2P and Android were fields very interesting for us to enter, first to know how the P2P concept works and then to learn how to develop an Android app and maybe publish it in the Play Store to share this work with the world.

For this reason of spreading and academic learning, we want this TFG to be a beginning point and maybe a reference guide for future works about P2P in which a sharing files application shall be created. Although there are tons of technology that allows to share files, we believe that the P2P concept, even being old enough, will continue being technologically developed in the future and above all is a concept that most people, even without knowing how it works, associates to a free files sharing, without barriers or hurdles, as a way of fighting against censorship that most governments or institutions have imposed o could implement.

This TFG entails an immersion into the net world, even though it is about making an Android app with P2P technology, the P2P concept cannot be understood without the net concept. Thus the existing types of nets must be clear and what can each kind of type be capable of. Furthermore in tech level, there have surged some problems that are inherent to the net structures and protocols that nowadays are used, thus we had to dive deep in this world and investigate to come out with an specific solution.

It has been developed an Android app in which two people can add themselves as friends, upload files that they think they should share and see the files that each one of their friends had upload, so they can download the shared files. We aren't aiming at a all in one Android app that allow us to share files very efficiently, fast and secure, so that we are creating the perfect app. We just want to achieve the proposed objectives and learn in the process.

Definitely, this project shows the whole process of learning accomplished and how we went from complete ignorance of the investigated topics to creating a functional Android app that we can use to share files under any kind of internet connection.

Keywords

P2P, Sharing, Files, App, Android, WebRTC, PubNub

II. Abreviaturas

API: Application Programming Interface.
DHCP: Dynamic Host Configuration Protocol.
DHT: Distributed Hash Table.
HTTP: Hypertext Transfer Protocol.
IaaS: Infrastructure as a Service.
ICE: Interactive Connectivity Establishment.
IGDP: Internet Gateway Device Protocol.
IoT: Internet of Things.
IP: Internet Protocol.
ISO: International Organization for Standardization.
IPv4: Protocolo de Internet versión 4.
IPv6: Protocol de Internet versión 6.
ISP: Internet Service Provider.
LAN: Local Area Network.
NAT: Network Address Translator.
NAT-PMP: NAT Port Mapping Protocol.
P2P: Peer to Peer.
PCP: Port Control Protocol.
SBC: Session Border Protocol.
SIP: Session Initiation Protocol.

SO: Sistema Operativo.

SSH: Secure Shell.

STUN: Session Traversal Utilities for NAT.

TCP: Transmission Control Protocol.

TFG: Trabajo de Fin de Grado.

TLS: Transfer Layout Security.

TURN: Traversal Using Relays around NAT.

UDP: User Datagram Protocol.

VoIP: Voz sobre Protocolo de Internet.

WebRTC: Web Real-Time Communications.

III. Indice

IV. Índice de figuras

1. Introducción

1.1. Motivación

La creación de una aplicación que haga uso de conexiones punto a punto para compartir ficheros es algo necesario hoy en día. Con los continuos ataques al anonimato en la red por parte de los gobiernos y empresas que solo buscan el lucro, la conexión directa entre peers aparece con más brillo que nunca para establecerse como un estándar en comunicaciones y compartición de archivos.

A todo el mundo le preocupa su privacidad, en mayor o menor medida. Cuando la privacidad se vulnera y se descubre que nuestros datos son visibles y tratados de cualquier manera por los gigantes empresariales y tecnológicos, es cuando surge la necesidad de la utilización de nuevos protocolos, o ya existentes para estandarizarse en nuestro día a día. Cada vez hay más aplicaciones, sobre todo de mensajería instantánea como Whatsapp, Telegram, etc, que ofrecen una encriptación extremo-a-extremo. Sin embargo, esto no termina de dar honor a la conexión punto a punto, o a las redes P2P que se basan en este principio, ya que se trata sólo de una encriptación, quedando registrados los mensajes en un servidor, o servidores, por los que circula y pudiendo ser descifrados de manera que se haría visible el contenido.

La creación de una aplicación que cree una red descentralizada P2P que de verdad cree una conexión directa entre nodos para la compartición de archivos y que estos no se queden alojados en ningún servidor surge casi como una necesidad, y este es uno de los principales motivos por el cual nos decantamos por este proyecto.

1.2. Objetivos

El objetivo prioritario de este proyecto es la creación de una aplicación Android que haga uso de la conexión punto a punto para la compartición de archivos de una manera segura y sin intermediarios que puedan acceder a nuestra información.

Con esto no se promete que la aplicación quede exenta de ataques que roben información como podría ser un ataque de intermediario (man-in-the-middle). Pues todo esto queda fuera del ámbito de la aplicación.

1.3. Estructura del documento

A continuación aparece la secuencia de capítulos, así como una breve descripción de su contenido:

- **Resumen:** Expone un breve resumen del trabajo realizado, algunas palabras clave que se utilizarán a lo largo del mismo y cómo se utilizarán en la memoria.

- **Índice Figuras:** Listado de todas las figuras y tablas que han sido utilizadas en este documento, junto con sus correspondientes números de página. Este índice aporta una vista previa de cómo se han usado los distintos ítems en el documento y sirve de ayuda para el lector a encontrar figuras específicas.

- **Abreviaturas:** Proporciona una lista ordenada alfabéticamente de abreviaturas que son importantes dentro del documento. Consultando esta lista, el lector puede localizar fácilmente las definiciones de las abreviaturas. También engloba los acrónimos.

- **Introducción:** Como primer capítulo del documento es el punto de partida del mismo y de la investigación previa a la puesta en marcha del proyecto. En él se

describe el tema elegido para realizar este Trabajo Fin de Grado, o lo que es lo mismo, la motivación que ha llevado a los miembros del grupo a realizar este proyecto. Seguidamente explica los objetivos que se desean alcanzar y para terminar se puede ver la estructura del documento para hacerse una idea de los capítulos contenidos en el mismo.

- **Redes P2P:** Este capítulo introduce las partes más importantes que caracterizan la investigación y documentación que se ha llevado a cabo en este Trabajo Fin de Grado antes de la fase de realización del mismo.

  Mediante este capítulo, se introduce al lector en el ámbito de las redes P2P, intentando explicar los conceptos que se han tenido que manejar a lo largo del proyecto, haciendo hincapié en los tipos de redes P2P que existen junto con sus características, ventajas e inconvenientes de las mismas, así como los protocolos para las conexiones en este tipo de redes. De igual manera, se habla de las tecnologías investigadas, así como del trabajo relacionado y consultado, como pueden ser otras aplicaciones similares en cuanto a funcionalidad a la que este proyecto quiere llegar, o una pequeña mirada atrás en el tiempo con aplicaciones que abrieron camino en este tipo de tecnología.

- **Desarrollo del proyecto:** Este capítulo presenta una visión global del desarrollo que ha sido necesario para el correcto funcionamiento del Trabajo Fin de Grado. Se comienza explicando la planificación realizada al inicio de este viaje, así como el análisis de requisitos básicos. Finalmente se explica el porqué de la elección tomada para realizar el desarrollo, explicando en detalle su configuración.

- **Aplicación Android:** Este capítulo contiene todo lo relacionado con la propia aplicación Android desarrollada, desde las primeras decisiones de diseño, pasando por los patrones de software implementados en la aplicación y por un

apartado de código relevante donde se indica lo que se cree que es el grueso de las “tripas” de la aplicación.

- **Resultados:** Finalmente, en el último apartado de este documento se presenta una discusión crítica y razonada sobre el resultado final del proyecto, así como las conclusiones finales y trabajo futuro por parte de los autores de este Trabajo Fin de Grado.

- **Apéndice:** Formado por dos apéndices: un manual de uso de la aplicación en el cual se explica a nivel de usuario el funcionamiento de la aplicación en sus diferentes pantallas para que el lector pueda entender y poder sacar mejor rendimiento de su uso. Para terminar, le sigue las contribuciones realizadas por los miembros del grupo al Trabajo Fin de Grado.

- **Bibliografía:** Para finalizar, listado de toda la documentación e información consultada para llevar a cabo la tarea de investigación para poder realizar este Trabajo Fin de Grado.

2. Redes P2P

La idea de conexión punto a punto surgió con el propio inicio de Internet al ser la manera más sencilla de establecer una conexión. Desde el inicio de las redes se han usado los sockets[1] como forma básica de conexión punto a punto, estando vigente a día de hoy en ejemplos como la conexión a la web que está reorientando su paradigma hacia WebSockets[2] dejando de lado la estructura cliente/servidor. Por todo esto, las conexiones punto a punto entre pares siempre van a existir ya que son la base de todas las nuevas formas de conexiones que se han desarrollado con el paso del tiempo.

La llegada de las redes locales (Local Area Network, en adelante LAN) y la implementación de seguridad del Firewall como muro entre el internet público y el privado,

ha complicado mucho más las conexiones punto a punto entre pares. La necesidad de implementar redes locales es debida al desbordamiento del número de direcciones IPv4 disponibles (4 billones), obligando a que los proveedores de direcciones IPv4 tengan que asignar la misma dirección IP a más de una máquina, haciendo necesaria una puerta de enlace (gateway) para enlazar una IP local con la IPv4 que el proveedor de internet nos ha asociado, dificultando la inicialmente sencilla conexión punto a punto teniendo que atravesar routers y firewalls para establecer la conexión.

Por culpa de la asignación de la misma dirección IP a más de una máquina, hay que diferenciar entre una dirección IP pública y una dirección IP privada.

La pública es el identificador de nuestra red desde el exterior, es decir, en el caso de una red residencial, la del router de nuestra casa, que es el que es visible desde fuera, mientras que la privada es la que identifica a cada uno de los dispositivos conectados a nuestra red, por lo tanto, cada una de las direcciones IP que el router asigna a nuestro ordenador, móvil, tablet o cualquier otro dispositivo que esté conectado a él. En el caso de redes residenciales, la pública suele ser una IP dinámica, es decir, puede cambiar cada poco tiempo, y de hecho así lo hace.

La IP pública del móvil puede cambiar con cada reconexión, es decir, cada vez que el móvil pierde conexión con el servidor del ISP, al momento de la reconexión puede que se le asigne una IP que puede ser la misma u otra nueva, necesitando un sistema de conexión punto a punto que sea estable y pueda pedir la nueva dirección IP cada vez que se produzca un cambio para no perder la conexión.

Desde entonces las conexiones punto a punto perdieron fuerza y sentido, ya que las direcciones bajo un mismo gateway estaban bajo una misma dirección IP pública y mediante una NAT (Network Address Translation, en adelante NAT), las respuestas son redirigidas a los distintos dispositivos dentro de la LAN, lo que nos ahorra muchas direcciones IP y nos protege del internet público mediante un firewall.

Sin embargo, las redes P2P nunca han dejado de estar de moda ya que ofrecen una conexión de punto a punto que nos aporta más seguridad al evitarse el uso de servidores que alojan nuestros datos, lo que siempre ha preocupado al usuario final. Tenemos ejemplos en PC de muchos tipos, desde los muy antiguos como Emule[3], que aunque usaba una conexión final punto a punto entre los ordenadores a compartir archivos, requiere de un servidor para permitir esa conexión entre peers, hasta los usados a dia de hoy como BitTorrent[4] (o sus múltiples derivados). Todos estos ejemplos están explicados en la sección de trabajo relacionado.

2.1. Tipos de redes P2P

Se denominan redes P2P a aquellas redes de ordenadores que siguiendo el típico esquema cliente-servidor, cada ordenador de la red hace al mismo tiempo de cliente y servidor. Esto es así ya que según la concepción de red P2P cada peer es igual en jerarquía y puede solicitar archivos o ser host de ellos.

Las redes P2P puede utilizarse para intercambiar archivos de audio, vídeo o software entre los nodos conectados, pero también permiten más cosas, como por ejemplo hacer más eficiente la transmisión de datos en tiempo real en telefonía VoIP (tecnología que hace posible que la señal de voz viaje a través de Internet mediante el Protocolo de Internet o IP).

Cuando queremos encontrar cual es la gran diferencia entre redes P2P, debemos atender a cómo está construida. Encontramos dos grandes diferencias referentes a las redes P2P: según su centralización y según su estructura.

Según su orden de centralización se pueden clasificar en:

- Redes P2P centralizadas: Todos los nodos están subyugados a un servidor central que maneja y dirige la comunicación entre nodos. Este servidor central conoce y administra todos los nodos conectados a él, teniendo así privilegios sobre los nodos. Este tipo de conexión es el más vulnerable ya que si se ataca al

servidor central y se tira la red dejará de estar disponible. Napster[5] usaba este modelo de red.

- Redes P2P descentralizadas: Este tipo de red posee agrupaciones de nodos en torno a un servidor que sirve como enrutador para los distintos nodos pero sin conocerlos. Este servidor tiene distintas funciones que ayudan a que la red funcione, pero no llega al nivel de poder sobre los nodos que tiene un servidor en una red centralizada. eMule es un ejemplo que utilizaba este tipo de red.

- Redes P2P distribuidas: Este tipo de red consiste en que cada peer es una parte por igual de la red, tanto compartiendo archivos como pudiendo solicitarlos a otros peers. Este tipo de red funciona como una gran "biblioteca" en común, ya que todos los archivos están disponibles aunque no estén situados en un mismo servidor o un mismo peer. Los nodos se disponen como una red neuronal de modo que cada nodo es compañero e igual a los otros. Es la más segura de las 3 redes según su orden de centralización, ya que si un nodo cae, la red puede seguir perfectamente funcionando. Este tipo de red era característica de Gnutella[6].

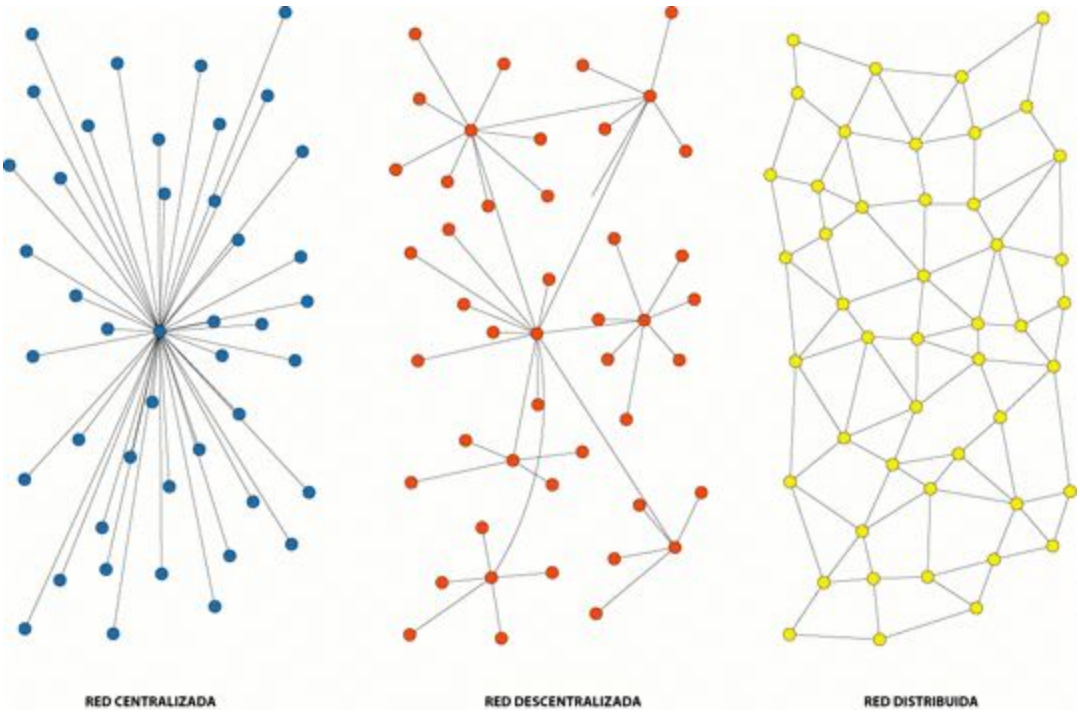

Figura 1: Tipos de redes P2P según su grado de centralización

Habiendo visto la clasificación por orden de centralización, las redes P2P pueden tener otro tipo de clasificación según su estructura. Este tipo de clasificación es el más típico y el más importante:

2.1.1 Redes P2P estructuradas

"Su principal característica es que en lugar de encargarse de organizar los nodos, se centra en la organización del contenido, agrupando el contenido similar en la red y creando una infraestructura que permite una búsqueda eficiente, entre otras cosas"[7].

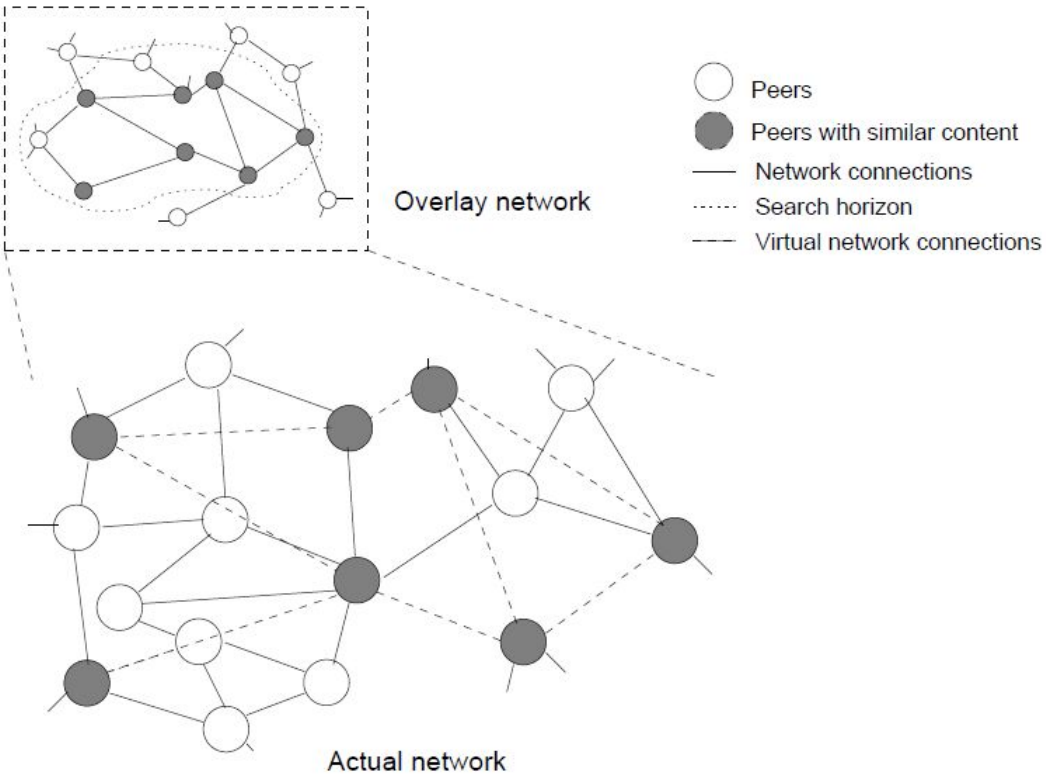

Figura 2: topología básica de red estructurada

La gran característica de las redes estructuradas es su búsqueda eficiente. Ya que vemos que organiza su contenido, un peer que requiera cierto archivo será redirigido a él de

una manera rápida y eficiente. Existen varios protocolos de búsqueda en esta red, pero el más conocido seguramente sea CHORD.

CHORD

Este protocolo hace uso de las DHT (Distributed Hash Tables) que “son un tipo de tablas hash, que almacenan pares (clave, valor) y permiten consultar el valor asociado a una clave, en las que los datos se almacenan de forma distribuida en una serie de nodos y proveen un servicio eficiente de búsqueda que permite encontrar el valor asociado a una clave.”[8]

De esta forma cuando un nodo entra en la red, recibe un id calculada mediante el hash SHA-1[9] quedando identificado dentro de la red de manera inequívoca. Cada nodo guarda información de la red en sí mismo en una tabla, de manera que puede identificar a otros nodos. De esta manera, cuando a un nodo le requieren una clave busca en su propia tabla si tiene esa clave y si no referencia a la clave más cercana a la requerida. Así, el nodo que genera la petición va, poco a poco, obteniendo un nodo más cercano al requerido hasta llegar al mismo.

2.1.2 Redes no estructuradas:

Una red no estructurada es una red en la que no existe una jerarquía, sino que todos los peers son iguales y se comunican entre ellos sin intermediarios. Si queremos encontrar un archivo en esta red, un peer va preguntando a sus conocidos por el archivo y así sucesivamente hasta encontrar a un peer que tenga el archivo y pueda compartirlo. Este tipo de búsqueda se conoce como “búsqueda por inundación”.

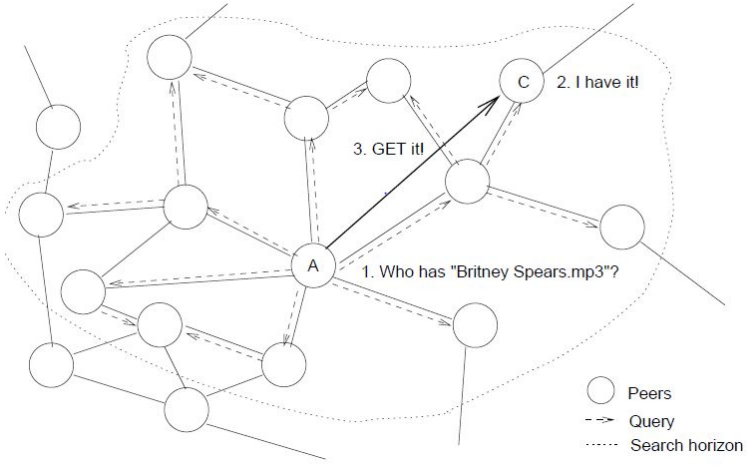

Figura 3: Topología básica de red no estructurada

Búsqueda Por Inundación

Dentro del uso de una red con estructura P2P es necesario saber quién de toda la red tiene la información que un peer puede necesitar. Por ello se desarrolló un algoritmo de búsqueda para este tipo de red conocido como “búsqueda por inundación.”

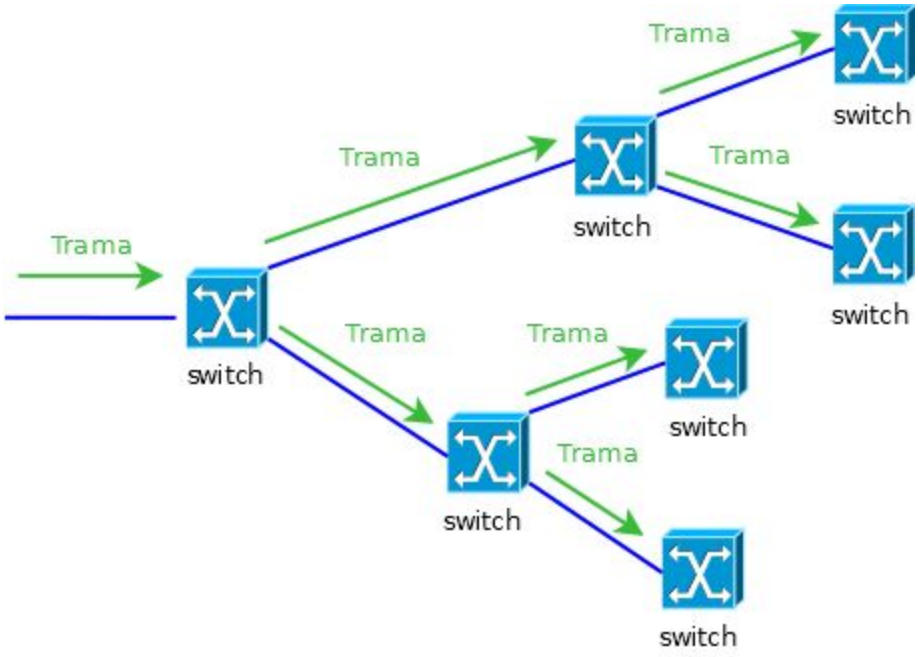

Figura 4: Funcionamiento de la búsqueda por inundación

El funcionamiento de este algoritmo se basa en la réplica de mensajes. Un nodo solicita a su peers conocidos ciertos archivos. Estos miran si poseen este archivo y en caso negativo, se replica el mensaje a todos sus peers conocidos. De esta manera se va transmitiendo el mensaje original sobre la solicitud de cierto archivo hasta que este llegue a un peer que disponga de dicha información o archivo. La transmisión del mensaje crece de esta manera exponencialmente, llegando en pocas iteraciones a un número elevado de peers. Dado este hecho, para no saturar la red de mensajes entre peers y así quitar el bando de ancha a la transmisión de archivos o información solicitada muchas veces, la búsqueda por inundación está limitada a un número determinado de iteraciones que se va restando cada vez

que el mensaje llega a un nuevo peer y las cuales si son pasadas el mensaje deja de propagarse.

El mensaje que envía el peer original solicitando su archivo suele contener su dirección de manera que el peer que tenga el archivo pueda contestarle.

2.2 Tecnologías investigadas

En base a todo lo anterior comentado en esta memoria y por culpa del actual modelo de conexión basado en routers que redirigen las distintas peticiones de los aparatos que se sitúan bajo su LAN, los cuales ofrecen seguridad mediante firewalls que rechazan peticiones externas y NAT que traducen las direcciones que llegan del exterior, la conexión punto a punto se ha hecho algo más complicado que lo que antaño era.

Por esto, fue necesario crear protocolos para crear túneles en el router (o bypass) de manera que podamos "saltarnos" estas restricciones que nos ofrece el actual modelo de conexión, todos estos protocolos de NAT transversal implican el uso de una tercera tecnología, ya sea en forma de servidor o en otra forma.

Dentro de todos estos protocolos existen algunos más ortodoxos que otros, y algunos más complicados que otros, y todos se han sopesado para poder utilizarlos en la aplicación a desarrollar:

- **Port forwarding** - Esta técnica consiste en mapear una dirección IP y un puerto (ruta 1) a otra dirección IP y puerto (ruta 2) de tal forma que se redirige el tráfico desde la ruta 1 a la ruta 2. Este mapeo se hace abriendo un puerto en nuestro router (gateway) para que todo el tráfico que llegue a la ruta del port forwarding se redirija. Si el mapeado no está hecho previamente, tendremos que acceder al software del router y conocer su funcionamiento. También es quizá el más limitado, ya que si estamos bajo una LAN del trabajo o pública (por pública nos referimos a aquellas zonas en las que podemos tener conexión en un sitio público, por lo que no tenemos acceso al router), no podremos

realizar el port forwarding. Por el hecho de ser tan limitado y de necesitar tener conocimientos del funcionamiento de un router, se desechó la idea de utilizarlo en el proyecto.

- **SSH** - El protocolo de Secure Shell fue concebido para reemplazar los protocolos inseguros de Unix rsh, rlogin, rexec y telnet, proveyendo una manera segura de poder loguearse a un terminal remoto en otra red distinta a la nuestra. SSH además nos asegura una encriptación como seguridad de que nuestra comunicación no podrá ser rastreada.

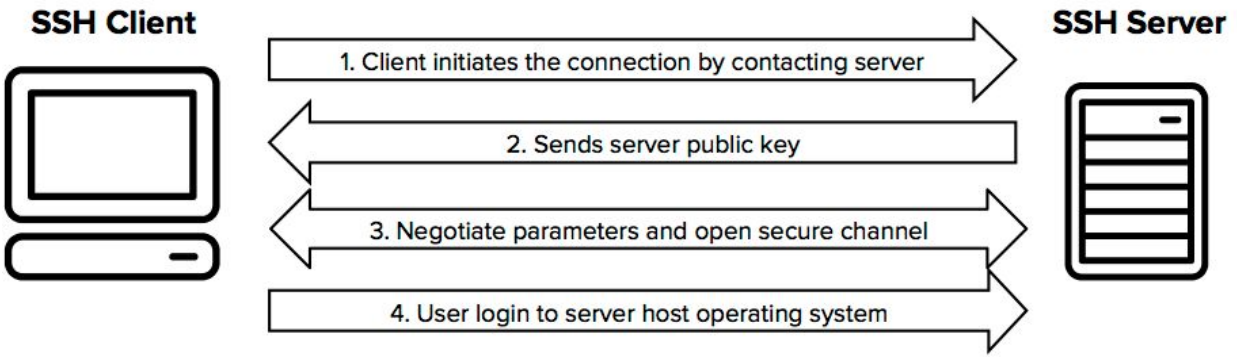

Figura 5: Cómo funciona el protocolo SSH

Se utiliza en aquellos países o ISP (Internet Service Provider) que bloquean ciertos contenidos para poder acceder de una manera libre. Aunque inicialmente se pensó que SSH podría ser la solución que necesitábamos para desarrollar la app dado el hecho de que SSH ofrece una forma de hacer bypass al firewall, mediante ssh reverse tunneling y conectar dos terminales detrás de firewalls y LAN's distintas mediante un servidor ssh intermedio que crea el túnel, no era una buena solución porque necesitamos un servidor que no esté bajo un firewall para crear la conexión entre los puntos que sí están detrás de un firewall. Esto choca frontalmente con la idea por la que usamos P2P, que es descentralizar la conexión y que solo pase por los 2 puntos, además de que tendríamos que montar un servidor permanente y siempre disponible que tendríamos que configurar, mantener siempre operativo y asegurarnos que no tiene un firewall intermedio entre el servidor e internet, añadiendo el hecho de

que aunque cree la conexión entre dos peers, el tráfico fluye por el servidor, por lo que con SSH no respetamos el P2P descentralizado que buscamos implementar.

- **Hole Punching** - El protocolo, o técnica, de Hole Punching se basa en el hecho de aprovecharse de las peticiones que salen del router para poder colar tráfico a través del puerto abierto y conocido. Esta técnica funciona mediante un servidor C, de modo que si A quiere comunicarse con B, realiza una petición al servidor C con su dirección IP y puerto, (de la misma manera hace B con el servidor C), y de esta manera el servidor responde a B con los datos de A y a A con los datos de B de manera que ambos podrán comunicarse aprovechando el agujero que la petición al servidor ha hecho en el router de cada uno ya que se conoce el puerto usado para comunicarse con el servidor.

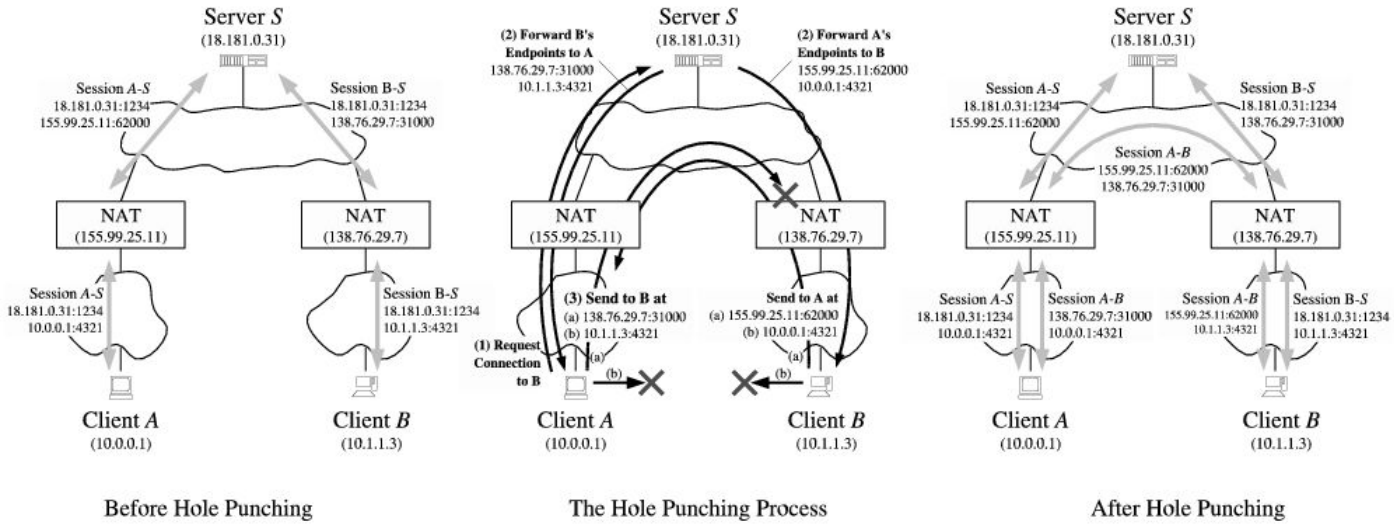

Figura 6: Esquema del protocolo Hole Punching

El problema con esta técnica es que no funciona con todos los tipos de NAT ya que no es algo que esté estandarizado y pueden surgir problemas con NAT simétricas.

- **STUN, TURN & ICE** - Todos ellos son protocolos de comunicación punto a punto utilizados por WebRTC (explicado más adelante), que realizan NAT traversal, surgidos

para la compartición de archivos de media y VoIP (Voice over IP). Todos ellos hacen uso de servidores STUN/TURN.
NAT traversal es una método que establece y mantiene una conexión punto a punto atravesando los distintos NAT de las LAN en los que están alojados los terminales.

- **STUN** - Session Traversal Utilities for NAT permite a un dispositivo conocer su IP pública. Cuando los terminales conocen su IP pública es posible realizar una conexión punto a punto entre ellos. El protocolo STUN está definido en RFC 3489.[10]

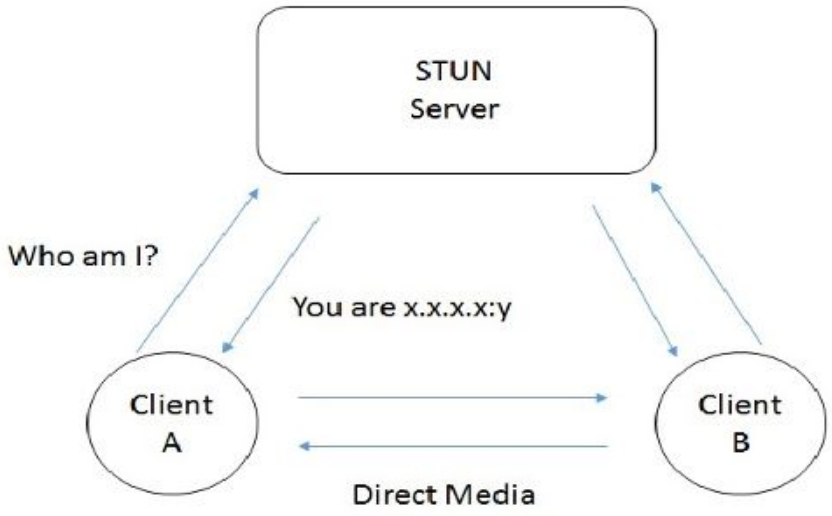

Figura 7: Funcionamiento del protocolo ICE con un servidor STUN

- **TURN** - Traversal Using Relays around NAT permite a un servidor confiar paquetes de datos entre terminales. Este protocolo se utiliza cuando la IP pública de un terminal no se puede determinar, debido a NAT simétricas, de modo que permite la conexión punto a punto mediante un servidor al cual se le confía la conexión de ambos terminales.

- **ICE** - Interactive Connectivity Establishment permite a los distintos terminales comunicar sus IP's públicas y conectarse unos a otros. ICE hace uso de ambos métodos anteriormente descritos, STUN & TURN.

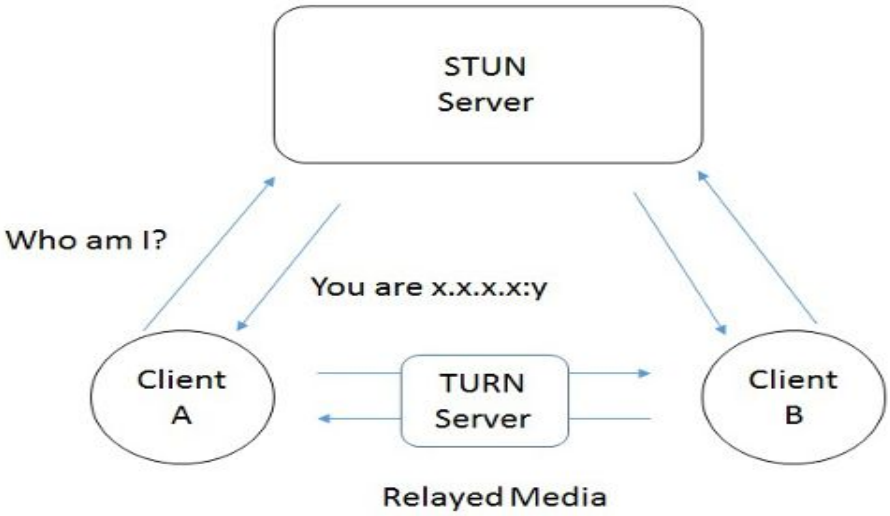

Figura 8: Funcionamiento del protocolo ICE usando un servidor STUN y otro TURN

"ICE intenta conectar los dispositivos con la menor latencia posible, mediante el protocolo de transporte UDP y los servidores STUN. Si UDP falla, ICE intenta conectar con el protocolo TCP: primero con HTTP, después con HTTPS. Si sigue sin funcionar utilizará un servidor TURN."[11]

Si no es posible la conexión punto a punto, ICE cambia la conexión al protocolo TURN. Este protocolo es usado normalmente cuando las NAT's son simétricas. TURN es un servidor intermedio que permanece después de establecerse la conexión.

- **WebRTC** (Web Real Time Communications) - Es un proyecto que provee sobre todo a los navegadores y también a aplicaciones móviles de una comunicación en tiempo real. Es utilizado sobre todo cuando se desea hacer streaming de media (video, audio) aunque también se puede hacer streaming de datos, haciéndolo mediante una conexión punto a punto. Hace uso de servidores de señal (signaling) para sincronizar a

los distintos peers que quieren conectarse a una sesión, el más conocido seguramente sea PubNub y a partir de eso conectar a ambos peers.

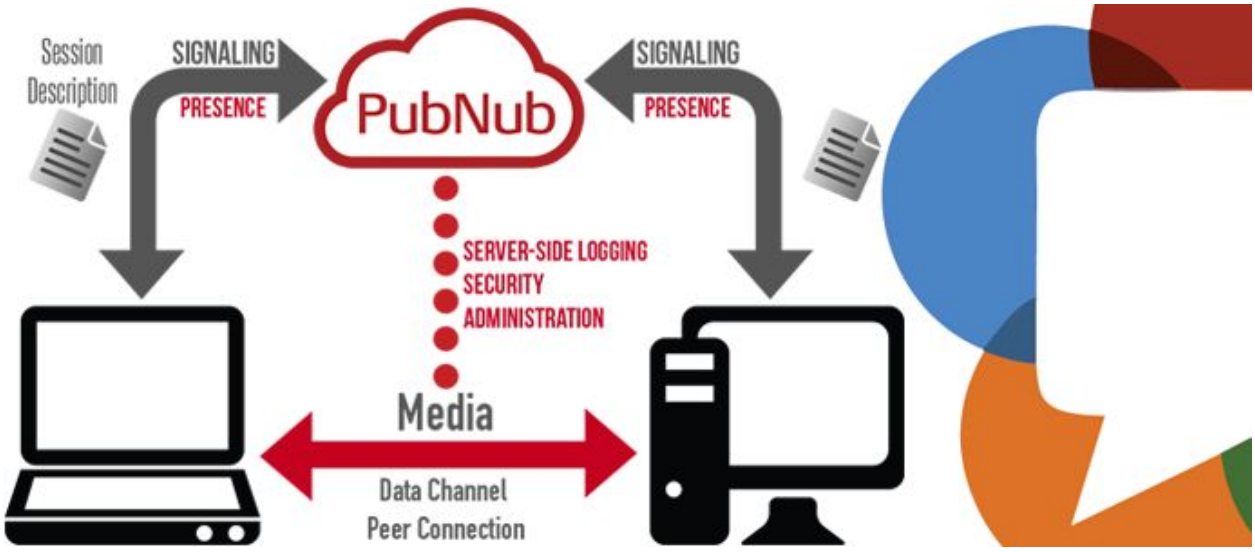

Figura 9: Esquema protocolo WebRTC

- **PubNub** - Es un servicio que nos proporciona una Red de Flujo de Datos (Data Stream Network) y una infraestructura como servicio de tiempo real (IaaS realtime) para el IoT (Internet of Things). Utilizando su servicio podemos sincronizar dos dispositivos en tiempo real para crear una conexión. Tiene una versión gratuita, en la que te registras y te dan 3 claves, una pública, una protegida y una privada y solo necesitas usar dos de ellas para establecer la conexión. Tiene una API flexible y es multiplataforma, haciendo que el uso de Android u otro SO no sea un problema. También nos proporciona una encriptación TLS y AES 256 en la conexión, siendo segura la comunicación.
  La versión gratuita permite hasta 100 conexiones simultáneas al servicio, dándonos infraestructura más que de sobra para el caso de uso y desarrollo de este TFG.

  Funciona creando una publicación, que está formada por las funciones channel, mensaje y async, que contiene un callback para esperar de forma asíncrona a que alguien se suscriba. Dentro del callback hay una función que espera la respuesta de la máquina suscrita.

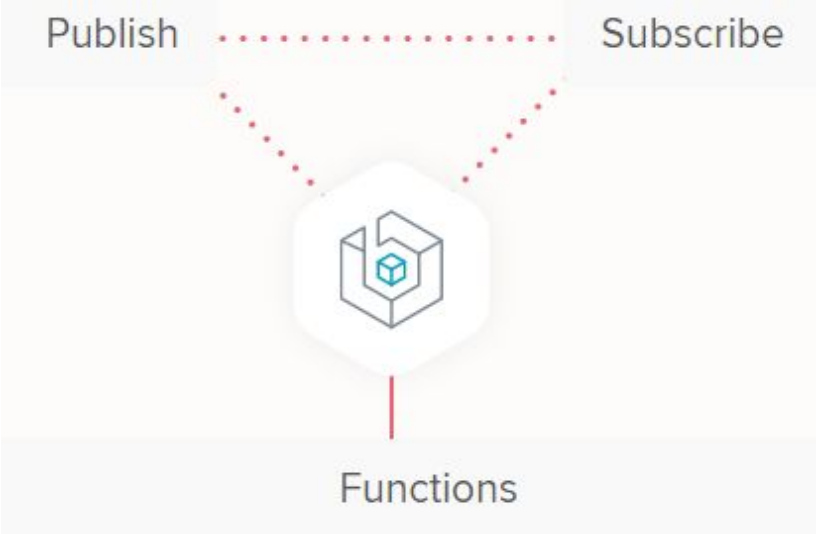

Figura 10: Esquema de la estructura de PubNub

2.3 Protocolos para conexiones

Estos protocolos a continuación referenciados son todos los que se han investigado para encontrar una solución de conexión punto a punto que evitase tener que utilizar fallos de seguridad en la conexión o un servidor de señalización para ejecutar la conexión. El problema es que ninguno de todos estos añaden una forma de crear la conexión, sólo son útiles para registrar esa conexión y mantenerla a lo largo del tiempo para no tener que crearla de nuevo.

- **UDP** "(User Datagram Protocol) es un protocolo del nivel de transporte basado en el intercambio de datagramas, el cual permite el envío de datagramas a través de la red sin que haya establecido previamente una conexión, ya que el propio datagrama incorpora suficiente información de direccionamiento."[12]

- **IPv6 -** Con este estándar en vez de IPv4 implementado en los routers, no tendríamos el recurrente problema de tener que atravesar el router a través de las NATs porque habría direcciones IP de sobra para todo el mundo y las NATs no existirían, pudiendo hacer una conexión punto a punto directamente, simplemente con sockets usando sólo la dirección IP más el puerto a conectar (si hubiese un firewall detrás cerrando los

puertos tendríamos los mismos problemas que ahora tenemos con IPv4 + NAT, y tendríamos que usar la misma solución que ahora hemos utilizado para saltarlo o alguna de las otras descritas como el Hole Punching).

Tanto IGDP como NAT-PMP como PCP sirven para permitir a las aplicaciones alojadas en nodos detrás de un gateway con NAT comunicar con el gateway para configurar el mapeo IP + puerto a otra IP + puerto distinto, haciendo la misma función que port forwarding, pero de forma dinámica.

- **IGDP (Internet Gateway Device Protocol) -** Protocolo para mapeado de puertos en sistemas NAT que no funciona en routers que no tengan NAT activado. Este protocolo es un estándar ISO y fue novedoso porque antes había que configurar el tráfico para permitir atravesar la puerta de enlace a mano, de esta forma con el protocolo se evita la pérdida de tiempo y los posibles errores que surjan en la configuración. Su sucesor indirecto es el protocolo NAT-PMP, pero al ser de Apple su verdadero sucesor directo y estandarizado es PCP.

- **NAT-PMP (Network Address Translation - Port Mapping Protocol) -** Funciona sobre UDP, usa el puerto 5351 y usa port forwarding. Sirve para determinar la dirección IPv4 de la puerta de enlace del NAT, la ventaja que tiene es que no necesita un servidor STUN y que el mapeado que hace del puerto expira con el tiempo, evita tener que estar pidiendo mensajes para mantener vivo el mapeado. Fue creado por Apple para usarlo como alternativa a IGDP. Su sucesor es el protocolo PCP.

- **PCP (Port Control Protocol) -** Se basa en el mapeado de la dirección IP junto con el puerto a conectar y el protocolo a través del que haces la conexión. Estos mapeos tienen un ciclo de vida relativamente largo (5 horas por ejemplo), durando siempre más tiempo que la conexión que se quiere lograr, ya que esta duración siempre se puede ampliar para que dure más que la conexión. Esta implementación de la duración es muy parecida a la que usa el DHCP cada vez que usa una concesión (lease). Este protocolo está estandarizado.

- **SIP (Session Initiation Protocol) -** Es un protocolo diseñado para iniciar y controlar sesiones en las que participa el usuario de una manera interactiva. Este protocolo quiere convertirse en el estándar para este tipo de interacciones. Como ejemplo de esto, se utiliza como protocolo para las comunicaciones VoIP (Voice over IP).

  A la hora de establecer una llamada VoIP, el usuario que la realiza envía un mensaje SIP que contiene su propia dirección IP. A su vez, el usuario que recibe la llamada ha de contestar con otro mensaje SIP al remitente (a la dirección IP del que realiza la llamada, obtenida en el mensaje SIP recibido). Pero esto no funciona si el usuario que realiza la llamada usa una IP privada, o una NAT. A raíz de este problema surgió el SBC, explicado a continuación.

- **SBC (Session Border Controller) -** Es un controlador que maneja las sesiones de VoIP garantizando su seguridad y fiabilidad. Actúa como un firewall para el tráfico de voz bloqueando posibles ataques e intrusiones a la sesión VoIP.

  Como se explicaba antes en SIP, al establecerse una llamada VoIP, el usuario que realiza la llamada incluye su propia dirección IP en el mensaje SIP. La función del controlador SBC radica en modificar el mensaje modificando la dirección IP, ya que al ser privada la del usuario no puede ser enrutada, y la modifica con su propia dirección IP. Las llamadas de este usuario, y las del usuario al que se llama, pasarán por el controlador SBC.

2.4 Trabajo Relacionado

Antes que esta aplicación han habido distintas aplicaciones que utilizan las redes P2P para la compartición de archivos entre distintos usuarios. Se describen algunas de las aplicaciones más famosas y utilizadas por todo el mundo para la compartición de archivos utilizando las redes P2P. Aunque a nivel de topología o estructura de la red no sean exactamente como la que se desarrolla en este TFG, es interesante explicar brevemente sus

características más importantes para comprender mejor el alcance de la aplicación P2P Sharing.

eDonkey

eDonkey es el nombre de una red P2P. El cliente que se usaba para acceder a esta red (actualmente desaparecida) era eDonkey2000. Esta red está clasificada como una red semi centralizada ya que existían servidores que manejaban y gestionaban el tráfico que había en la red, pero no eran centrales ni había alguno con una jerarquía superior a otro. Una de las características que permitía eDonkey es que estos servidores podían ser levantados y gestionados por los propios usuarios.

Los servidores de eDonkey tenían dos funciones principales:

- Establecían la conexión con el cliente que quería entrar a la red, guardando una lista de archivos que comparte.

- Si un cliente busca un archivo le envía la petición al servidor, que busca dentro de la lista de clientes que se han conectado a la red si alguno tiene el archivo solicitado y replica esta petición a otros servidores en caso negativo. Cuando el archivo es encontrado, los servidores se encargan de conectar a ambos peers para la descarga.
-

Una de las características de eDonkey, que la hacen algo lenta en cuanto a velocidad de descarga, es que la pieza del bloque a compartir era fijo, siendo este de 9800 KB.

eMule

eMule evolucionó a partir de eDonkey y se ofreció como una alternativa a este. Los problemas que presentaba la red de eDonkey fueron solucionados, en parte, por eMule, como

el problema del tamaño de los bloques. Una de las características de eMule que no tenía eDonkey y que potenció el uso de eMule fue que este cliente fue escrito en C++ y con una licencia GPL, era de código abierto, por lo que cualquier usuario podía aportar su código y modificar libremente.

EMule dispone de dos redes para la compartición de datos, y ambas son bastante distintas entre sí:

- La red clásica Ed2k basada en servidores. Esta red funciona tal como funcionaba la red de eDonkey. El usuario que se conectaba debía conocer la dirección de un servidor al cual le comunica los archivos que quiere compartir. Para buscar hace una petición al servidor que la réplica a otros o le contesta conectándolo al peer con el archivo requerido.

- Red Kad, es una red descentralizada. En esta red todos los nodos son iguales en jerarquía ya que se eliminan los servidores que si tiene la red clásica. Para conectarse a la red Kad, ya no es necesario saber la IP de un servidor, si no la de otro peer. Estos nodos guardan información de los archivos que se quieren compartir y cuando quieren buscar alguno, se busca a los nodos que tienen estos archivos.

Napster

Napster fue la red P2P de intercambio de mp3 más popular en la década de los 2000. La característica del tipo de red que utilizaba es que esta era centralizada, por lo que tenía un servidor central que servía a los distintos peers para descubrir los distintos archivos que existían en la red. Sin embargo, la transmisión de archivos se realiza con una conexión punto a punto entre los peers, sin intervenir el servidor central, por lo que la función de este servidor central es mantener una lista de usuarios activos y cuáles son los archivos que comparten.

Como se ha hablado anteriormente, la inconveniencia que tiene este tipo de red P2P es que el servidor mantiene y gestiona la comunicación entre peers, ya que permite el descubrimiento de archivos. Sin embargo, cuando este servidor cae o necesita ser mantenido, la red también cae, haciéndola vulnerable.

Gnutella

Gnutella es definida muchas veces como una red P2P pura, ya que todos los nodos tienen la misma jerarquía dentro de la red y no existen servidores o supernodos que gestionen la comunicación ni la transferencia.

Gnutella es la que hizo famoso el algoritmo de búsqueda por inundación, ya que si un nodo solicitaba un archivo, preguntaba a sus conocidos, y estos si no disponían del archivo replicaban el mensaje a sus conocidos y así sucesivamente, de manera que finalmente el peer que disponía del archivo contestaba al peer emisor original y se conectaban. La única pega que podríamos sacar de este algoritmo de búsqueda es que, si muchos peers a la vez generan mensajes de búsqueda, al crecer este de una manera exponencial cada vez que es replicado por un peer, saturaría la red y el ancho de banda con mensaje de los peers buscando ciertos archivos, por lo que normalmente se les pone un límite a estos mensajes de búsqueda.

La ventaja que tiene Gnutella frente a otros tipos de protocolos o redes P2P es que al ser una red totalmente descentralizada es más robusta frente a caídas de distintos peers, ya que es una red comunitaria, no está gestionada por ningún servidor central como si ocurre con el caso de Napster.

BitTorrent

BitTorrent es un protocolo que hace uso de la filosofía de red P2P para compartir archivos. ¿Qué es lo que hace a BitTorrent especial frente a otro tipo de protocolos de redes P2P? La manera en que los peers intercambian los archivos.

BitTorrent comparte archivos parciales, de manera que un archivo queda troceado en partes de determinados bytes que son compartidos, y estos archivos parciales son compartidos por peers distintos dentro de la red. Es decir, un archivo puede ser descargado por partes de muchos peers distintos, que contribuyen con su parte a que tu tengas finalmente el archivo que solicitas. Esto hace que cada peer dentro de la red sea servidor y pueda contribuir con un archivo parcial a otro peer que lo requiera. Al peer que está descargando un archivo se le denomina leecher, una vez que ha descargado el archivo completo pasa a denominarse “seed” dentro de la red.

Para gestionar la localización de los archivos, la lista de los peers y otras cosas de uso general en la descarga, BitTorrent hace uso de unos servidores “trackers” que como su nombre indica, son los encargados de buscar o rastrear archivos o peers para que estos puedan conectarse entre sí y compartir archivos. Los trackers guardan información de todos los peers, tanto leechers cómo seeds, y va actualizando la lista de peers del archivo que te estás descargando, por si alguno se desconecta o se conecta algún otro.

En paralelo al sistema de trackers, BitTorrent utiliza ciertos usuarios de la red que forman parte del árbol de conexiones punto a punto como supernodos (peers especiales) de ese árbol de conexiones, estos supernodos suponen que a la conexión punto a punto se le sume este tercer nodo para facilitar la transferencia. La elección de estos usuarios se hace en base al posicionamiento dentro del árbol que tienen, siendo los más centrales y con mejor conexión con el resto los elegidos. Estos supernodos contienen la información/archivos que todos los nodos puedan pedir, de tal forma que cuando un nodo hace una peticion de algun archivo, se pueden usar esos supernodos, ya sea como seguro de que alguien de la red lo tendrá y estará conectado para transferirlo o como acelerador de la transferencia al ser el supernodo un peer más que aumenta la velocidad de transferencia.

Para evitar los problemas de atravesar los intermediarios (como NAT, Firewall, Routers) para establecer la conexión en IPv4, BitTorrent usa la técnica del Hole Punching para permitir la conexión punto a punto atravesando los intermediarios que haya en la red en forma de router, firewall, etc.

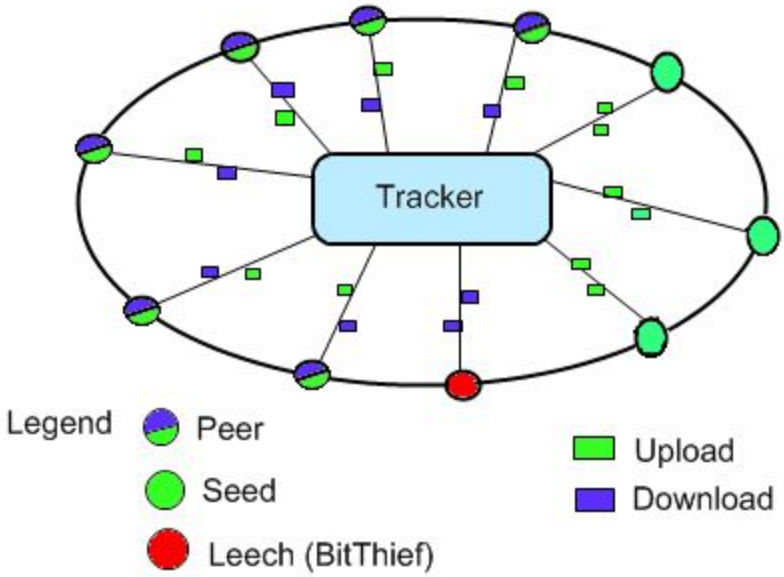

Figura 11: Ejemplo de red P2P con múltiples nodos

En Android y como referencia del trabajo que se está desarrollando, se han encontrado muchas aplicaciones de intercambio de ficheros punto a punto, pero casi todas estas aplicaciones se basan en la certificación WiFi Direct para hacer la conexión punto a punto en red local, evitando usar internet o una conexión que implique consumo de datos, con lo que esta conexión solo se puede hacer en red local, siendo este intercambio de archivos una mejora con respecto al protocolo Bluetooth pero con las velocidades de transferencia del estándar WIFI. Uno de los ejemplos más usados y descargados de la Play Store (cuenta con más de 500.000.000 descargas) es ShareIt[13], que cuenta con la múltiples funcionalidades de intercambio punto a punto usando lo comentado anteriormente.

3. Desarrollo del Proyecto

3.1 Planificación

Dentro del proceso del desarrollo se ha considerado necesaria tener una parte planificación, por dos motivos, el primero ejercer el control sobre el proyecto y el segundo para poner en práctica esta etapa de la gestión de proyectos dentro de la ingeniería de software.

La planificación tiene cuatro fases: Fase de Inicio, Fase de Elaboración, Fase de Construcción y Fase de Transición. De estas cuatro fases, solo se han puesto en práctica la fase de elaboración y la de construcción.

En la fase de elaboración se ha desarrollado el proyecto y su arquitectura, estableciendo una prioridad entre los requisitos y planificando que miembros van a realizar qué tareas y el tiempo necesario.

En la fase de construcción se ha creado el ejecutable del proyecto y probado su funcionalidad para asegurar que se han cumplido los requisitos establecidos, además de poder entregar la funcionalidad a los directores para que evalúen el desarrollo del proyecto.

Al tener tanto la idea del proyecto como los riesgos definidos (temporales en su mayoría) antes de hacer la planificación, esta etapa ya está hecha y la fase de transición no existe en este proyecto pues una vez entregado no va a haber ni iteraciones en el desarrollo ni mantenimiento del proyecto, ni feedback de cara a desarrollar nuevas funciones por parte del cliente (en este caso el tribunal evaluador).

Para realizar la fase de elaboración y construcción se ha generado una planificación en forma de diagrama de Gantt, ya que es un recurso que se ha aprendido en la carrera y facilita el seguimiento y control de los procesos involucrados en el desarrollo, como podría ser el

generar la mayoría de los requisitos que el proyecto implementará. También ayuda a reducir o eliminar los riesgos importantes que puedan surgir durante el proyecto.

3.1.1 Diagrama de Gantt

Para que se pueda ver bien, he troceado el diagrama en imagenes segun la funcionalidad. El color gris de la barra superior representa la etapa que se ha planificado. Dentro de las barras temporales el color azul claro representa trabajo de Josué, el naranja trabajo de Jesús y el verde trabajo de ambos. El bordado rojo de las barras representa el camino crítico de la planificación.

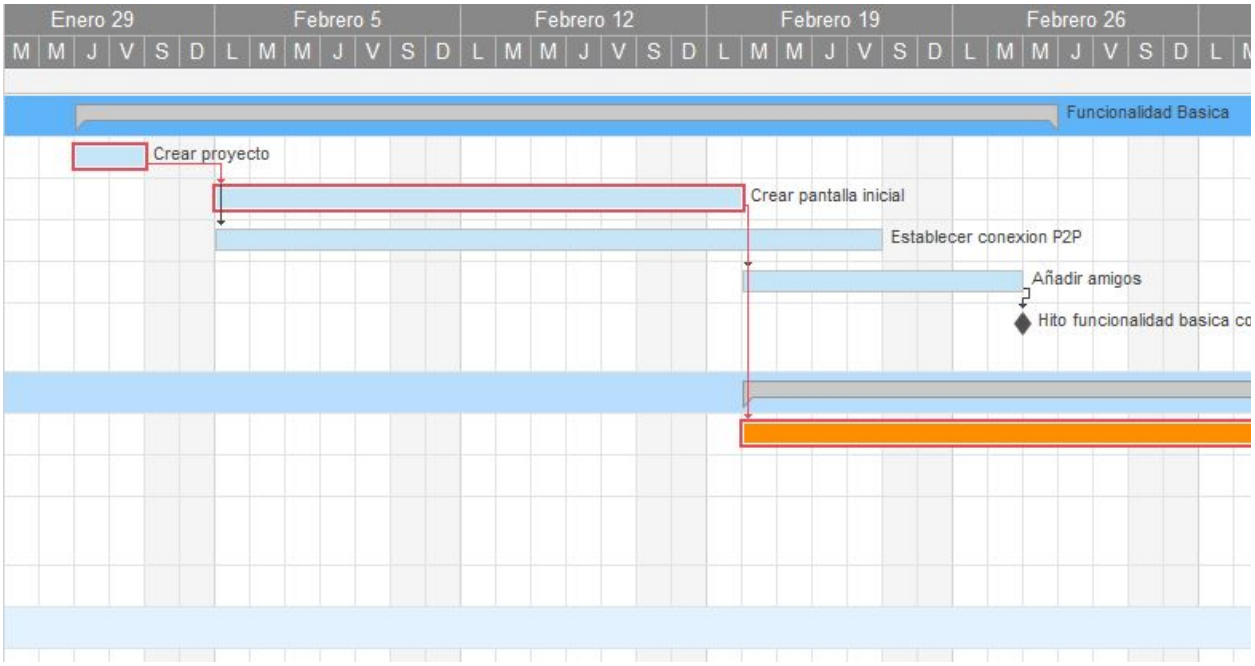

Figura 12: Planificación Funcionalidad Básica

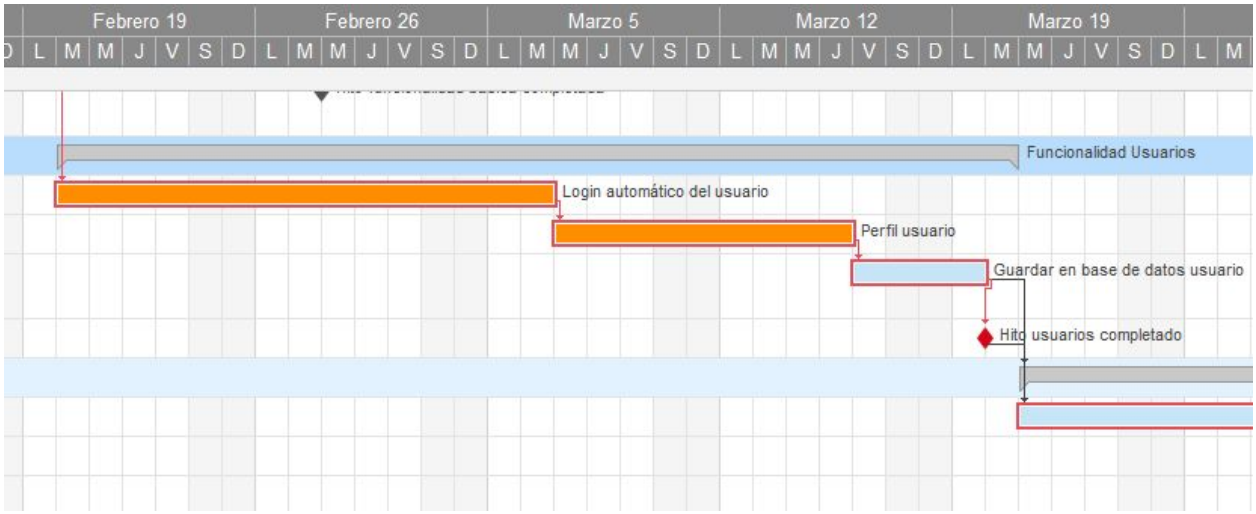

Figura 13: Planificación funcionalidad Usuarios

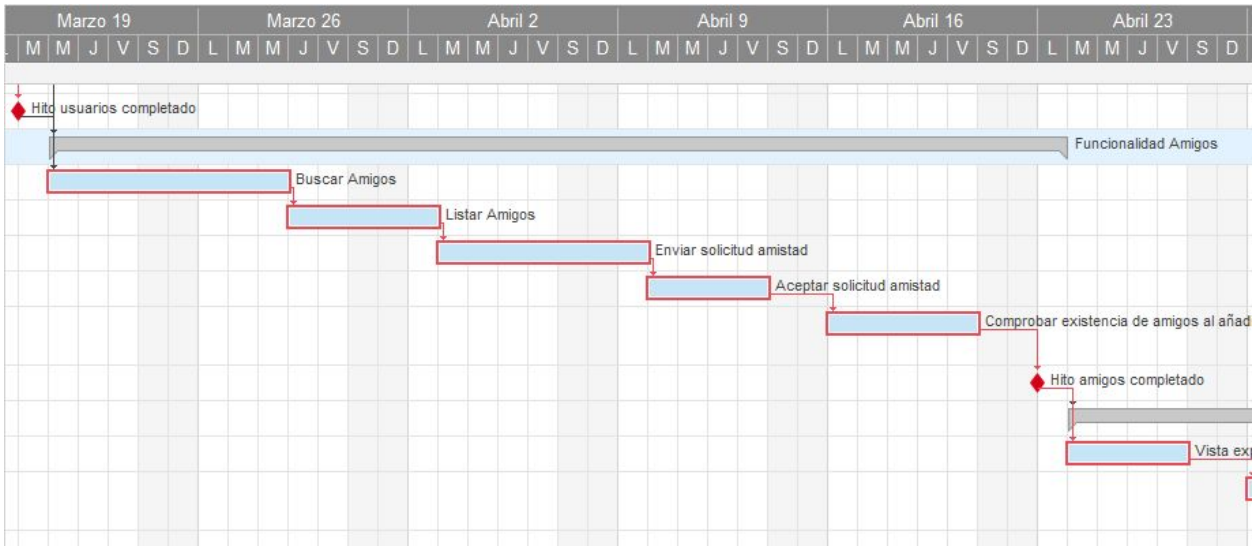

Figura 14: Planificación funcionalidad Amigos

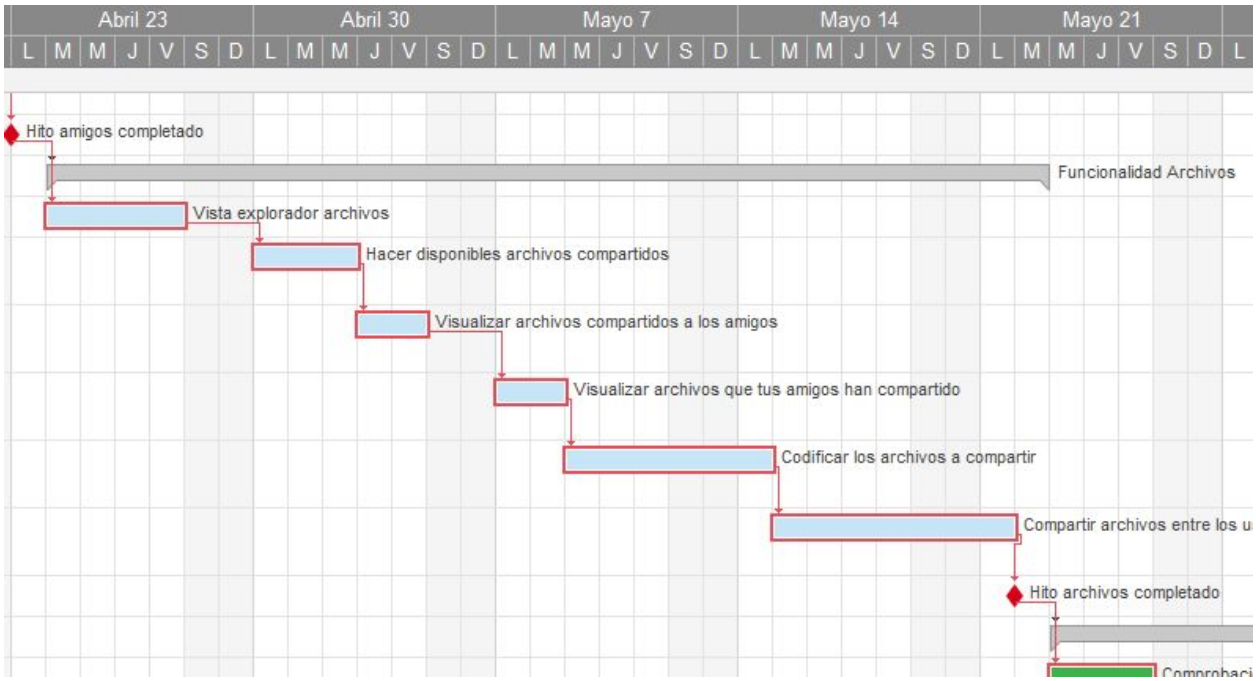

Figura 15: Planificación funcionalidad Archivos

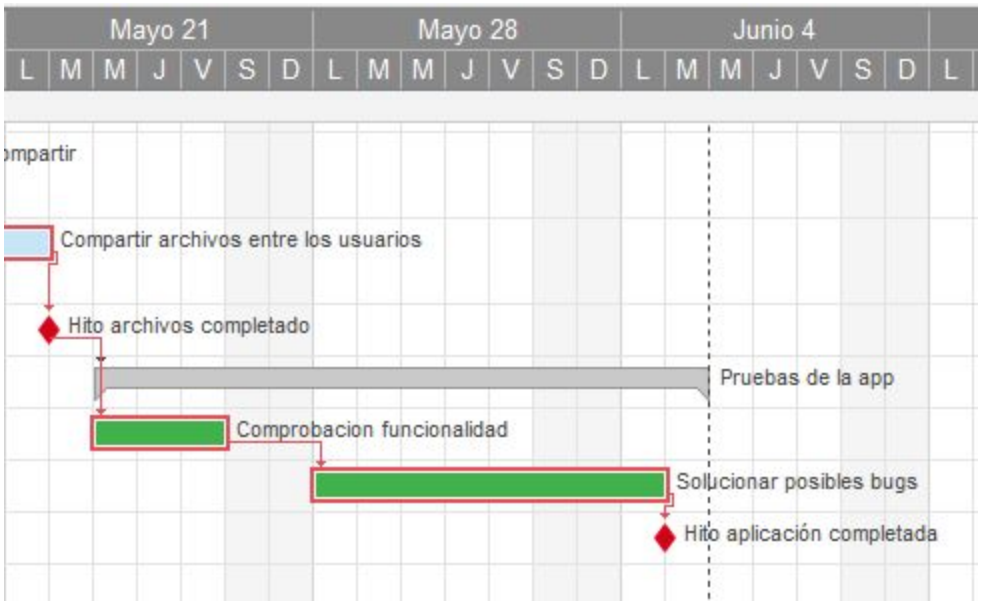

Figura 16: Planificación pruebas App

Debido a que el proyecto es puramente académico (no hay dinero involucrado) y se ha consensuado con los directores y miembros del proyecto las funcionalidades antes de la

ejecución del proyecto, todos nuestros riesgos están relacionados con el tiempo y la forma de ejecutar el proyecto (no es aceptable que el proyecto no funcione aunque haya código hecho).

Estos riesgos importantes pueden ser:

- Retraso en el inicio del proyecto.
- Falta de tiempo para completar todas las funcionalidades y requisitos pensados al hacer el proyecto
- Retraso en la realización de tareas por parte de algún miembro del equipo impidiendo al resto avanzar
- Imprevistos causados por el desconocimiento de la tecnología usada

Al tener el tiempo muy medido y la funcionalidad planificada y cerrada antes de iniciar el proyecto la planificación no se actualizará conforme avance el proyecto, una vez generada no se modificara. Esto va en contra de la mayoría de recomendaciones que se hacen al generar una planificación, ya que la planificación es un proceso continuo con capacidad de adaptabilidad según las necesidades y riesgos que se produzcan, pero el ámbito académico del proyecto nos permite hacer una planificación menos exhaustiva y más usarla como una herramienta interna de autocontrol que como un pilar sobre el que desarrollar el proyecto.

3.2 Análisis de Requisitos Básicos

La función principal de esta aplicación es la de ser un enlace entre dos usuarios para que estos puedan compartir archivos entre ellos de punto a punto. Los elementos que componen nuestra aplicación son: Usuarios y Archivos. Estos elementos los usamos como subsistemas para generar y explicar los requisitos básicos y cual es la funcionalidad que aportan a la aplicación.

Usuarios

Es un subsistema que se encarga de toda la gestión relacionada con los usuarios de la aplicación:

Crear Usuario: El usuario será dado de alta por primera vez al iniciar la aplicación. Se le pedirá que meta un nombre identificativo para mostrarlo a todo el mundo en la aplicación. En principio no existe ninguna restricción que permita tener 2 nombres iguales dentro de la red, por lo que se habla de ello en el apartado de Trabajo Futuro. Estos datos serán guardados en la base de datos con el fin de poder identificarle de una manera rápida las sucesivas veces que abra la aplicación mediante el Login.

Mostrar Perfil Usuario: Muestra la información del usuario obteniendo detalles del mismo, como el nombre y los amigos que tiene.

Login Usuario: El usuario, una vez registrado, será automáticamente reconocido por la aplicación mostrándole su nombre en la pantalla junto con un mensaje de bienvenida, después será redirigido al perfil para poder comenzar la compartición de archivos.

Salida de la aplicación: El usuario puede salir de la aplicación en cualquier momento. Si tiene descargas activas, el servicio seguirá activo lo que Android considere necesario, ya que al pasarse la actividad a un estado de “Stop”, sí Android necesita recursos en forma de memoria, destruirá la actividad aunque guardará el estado en el que se encontraba cuando se minimizó, por lo que si dejamos la aplicación en segundo plano sin haber terminado las descargas, se corre el riesgo de que Android destruya la actividad y las descargas se cancelen.

Archivos

Con los archivos tenemos distintas funciones dependiendo de lo que se quiera hacer con ellos. Existirán distintos casos de uso:

Subir archivo: Es el momento de preparación del archivo antes de ser compartido.

Compartir archivo: El momento en el que el cliente decide compartir un archivo con sus amigos y éste se hace visible para ellos.

Mostrar archivos compartidos: Muestra todos los archivos compartidos mediante la aplicación.

Amigos

En cuanto al subsistema de Amigos, se tienen los siguientes casos de uso:

Lista de amigos: Al entrar en la aplicación identificados con las credenciales de un usuario, se muestran los amigos que se han añadido anteriormente.

Enviar solicitud de amistad: El usuario, una vez registrado, puede y debe enviar una solicitud de amistad a otro usuario para poder comenzar la compartición de archivos entre ambos. Debe introducir el nick o nombre de usuario para poder enviarle la solicitud.

3.3 Tecnologías Empleadas

Una vez expuestas las tecnologías más relevantes, tuvimos que tomar la decisión de cual de todos los protocolos de conexión implementamos en nuestra App Android. Esta decisión no fue difícil de tomar debido a todos los problemas al intentar atravesar el NAT, impidiendo que se llegase a realizar la conexión, estos problemas son derivados del uso de IPv4 como antes explicamos (en adelante estos problemas serán referidos como travesía NAT).

Para empezar, debido a la facilidad de uso al ser la librería estándar de JAVA para establecer conexiones punto a punto inicialmente se probó con Sockets. El establecer la conexión en localhost con Sockets es casi trivial, haciéndola posible con este fragmento de código:

```
public class User {
        protected String ip;
        protected int puerto;
        protected ServerSocket ss;
        protected Socket cs;
        protected BufferedReader entrada;
    protected PrintWriter salida;

        public User(String op) throws IOException {
                Scanner sc = new Scanner(System.in);
                this.ip = InetAddress.getLocalHost().getHostAddress();
                this.puerto = (int) (Math.random() * 40000) + 1234;
                if(op.equalsIgnoreCase("servidor")) {
                        ss = new ServerSocket(puerto);
                        cs = new Socket();
                        System.out.println("ip a conectarse: " + ip);
                        System.out.println( "puerto a conectarse: " + puerto);
                }else {
                        System.out.println("Insertar ip que buscar: ");
                        String host = sc.nextLine();
                        System.out.println("Insertar puerto: ");
                        int port = sc.nextInt();
                        cs = new Socket(host, port);
                }
                //sc.close();
        }

}
```

Figura 17: Ejemplo de uso de sockets

El problema con sockets es que fuera de localhost la conexión nunca llega a establecerse por no haber travesía NAT, así que tuvimos que empezar una búsqueda más compleja donde encontrar una solución a ambas cosas.

La travesía NAT ha salido a la luz al hacer pruebas de implementación con las tecnologías expuestas y con otras como Hive2Hive[14] (proyecto de código JAVA junto con código Android con conexiones punto a punto implementadas), que no está detallado en esta memoria al no tener tanta relevancia tecnológica como de uso con respecto a lo ya explicado, además de no ser adecuado al usar supernodos (peers especiales) para facilitar la transferencia, haciendo que la conexión no sea estrictamente punto a punto, sino que un tercer peer se añade, terminando de hacer que desestimasemos completamente esta tecnología. Aun así, en la concepción inicial del proyecto se usó para introducirnos, explorar y probar la manera de programar código JAVA para hacer conexión punto a punto.

Después de haber continuado investigando tecnologías que sirviese de base para establecer la conexión atravesando la NAT, encontramos en WebRTC una posible solución a nuestros problemas, la configuración como abajo se explica no es compleja y nos proporciona una interfaz estandarizada y fiable, siendo la solución más sencilla que encontramos. Pero WebRTC por sí solo no conseguía lograr la conexión, con lo que continuando con la investigación descubrimos que PubNub era el servicio en el que se apoya Google (el servicio Hangouts utiliza WebRTC para transmitir audio y video a entre sus usuarios) para hacer la parte de señalización de las conexiones con WebRTC. Gracias a encontrar estas dos tecnologías se ha podido solventar todos los problemas de conexión surgidos y permitirnos comenzar el desarrollo de la aplicación Android estando seguros de que los fundamentos sobre los que creamos la aplicación se ajustan al objetivo de este TFG.

3.4 Configuración y conexión con PubNub + WebRTC

Para realizar la conexión entre dos dispositivos primero debe llevarse a cabo una señalización a cargo de PubNub. De esta manera nos aseguramos que ambos peers son avisados para prepararse de cara a una conexión con otro peer.

Mediante la señalización, PubNub se encarga de proveer de los datos necesarios para que los peers puedan establecer una conexión punto a punto mediante WebRTC. En esta señalización los peers se preparan para conectarse creando el cliente de RTC y creando un

canal por el cual escuchar a conexiones entrantes (o salientes en el caso de ser el nodo que busca conectarse).

PubNub sigue un patrón publisher/subscriber de manera que cuando un peer publica un archivo y otro peer se lo quiere descargar, se subscribe a él de manera que puede descargárselo. Cuando un peer quiere subscribirse a otro es cuando PubNub señaliza y sincroniza a ambos para una conexión.

Una vez que los peers se han conectado pueden comunicarse mediante el DataChannel que WebRTC provee. Aquí la comunicación ya es punto a punto y PubNub ya no hace nada más con los peers hasta que estos cierran la conexión. Como se comentó anteriormente, WebRTC hace uso de ICE, por lo que en primera instancia utiliza un protocolo de transporte UDP junto con los servidores STUN. Si esto esto falla se pasará a un protocolo TCP y en última instancia se utilizará un servidor TURN.

La forma que tienen de comunicarse los peers es mediante mensajes JSON, por lo que cualquier tipo de archivo binario que quieran compartirse deberá ser codificado en un string para que pueda ser insertado en un JSON, de modo que nosotros utilizamos la codificación en Base64. Actualmente esto no es necesario, ya que WebRTC implementó que mediante su DataChannel pudiera compartirse cualquier tipo de archivo sin ser necesaria una codificación en Base64 o de cualquier otro tipo para entrar dentro de un mensaje JSON. Todo esto está explicado en más detalle en la sección de Trabajo Futuro.

4. Aplicación Android

En este apartado se va a hablar de todo lo relacionado con la aplicación Android P2P Sharing que se ha desarrollado en este TFG. Primero hablamos de las decisiones tomadas a lo largo del desarrollo, tanto en materia visual de la aplicación como en la parte técnica del código, junto con los motivos que nos han llevado a tomar esas decisiones.

Aunque en la aplicación no hay una estructura muy compleja de ingeniería de software, sí que se han incorporado elementos propios de ella a nivel de código como los patrones de capas y los modelos de datos, los cuales se muestran y se explican resaltando su importancia en la aplicación. También dentro de todas las secciones se explican los problemas que han surgido y de qué manera se han solucionado.

4.1 Decisiones de Diseño

A continuación se va a pasar a describir las distintas decisiones que se tomaron en cuanto al diseño de la aplicación, tanto en el aspecto visual, como en el aspecto de ingeniería de código.

4.1.1 Diseño estructural

Las aplicaciones en Android se desarrollan en actividades, las cuales son fácilmente reconocibles ya que cada actividad cuenta con una pantalla distinta. En nuestra aplicación disponemos de 4 actividades que marcarán el diseño estructural de la aplicación y serán las que dirijan el flujo de uso de la aplicación:

- **Activity_profile**. Es la actividad más importante y sobre la que se basa toda la aplicación. Es la actividad que te permite ver a tus amigos y compartir archivos P2P ya que en ella también se encuentra toda la lógica de conexión y los distintos listeners necesarios que utiliza el protocolo WebRTC para comunicarse. En ella podemos ver a los amigos que tenemos, añadir distintos amigos, ir a mis archivos compartidos, ver los archivos compartidos por mis amigos y compartir un archivo nuevo. Esta actividad se distingue por no destruirse nunca mientras la aplicación esté activa, hecho por el cual nos decidimos a poner en ella los listeners necesarios para dirigir la conexión y comunicación de la aplicación.

- **Activity_main**. Es la actividad que arranca la aplicación. Esta actividad nos mostrará en la pantalla inicial, la cual comprobará si existe un usuario creado ya, en cuyo caso dará paso a la actividad del perfil directamente o, en caso contrario, es la primera vez que se

abre y hace falta dar de alta un usuario nuevo, en cuyo caso abre un diálogo en el que nos permitirá meter un nombre con el que podremos ser identificados dentro de la red P2P y el cual compartiremos con nuestros amigos.

- **Activity_recursos**. Actividad que tiene como tarea principal mostrar los recursos compartidos, tanto de uno mismo, como de otro amigo que seleccionemos.

- **Activity_archive_explorer**. Esta actividad se ejecutará en el momento en el que queramos buscar un archivo para compartir.

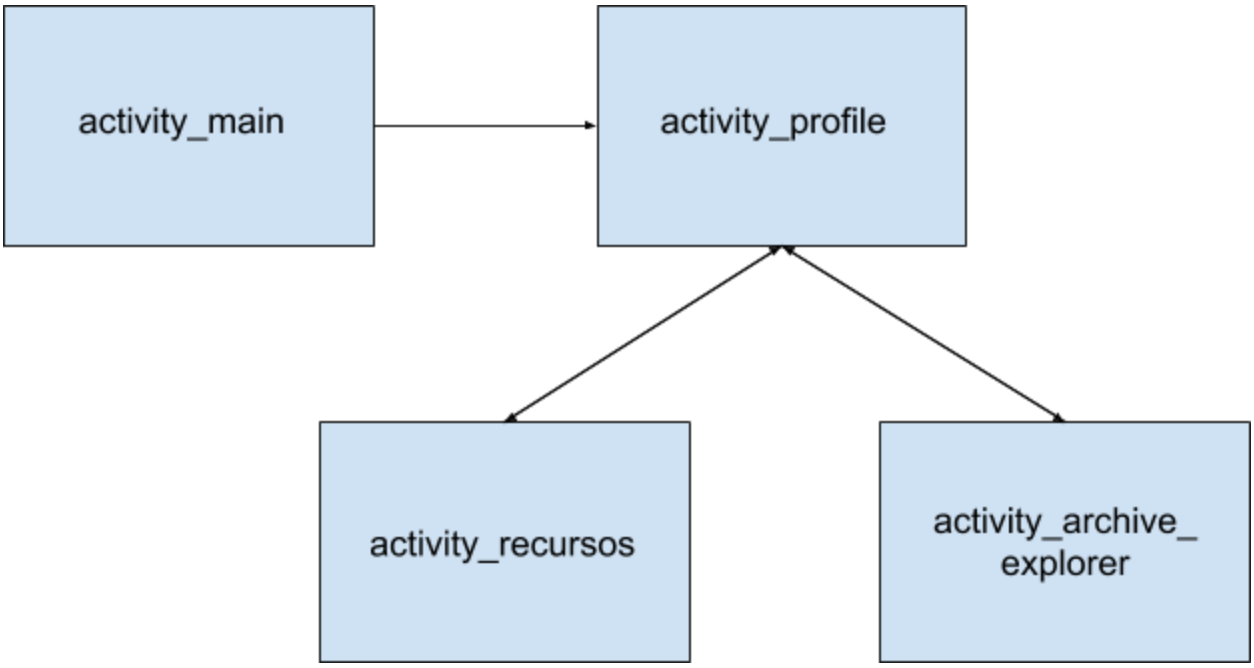

Figura 18: Dependencia de las distintas actividades

Como se ha comentado anteriormente, la actividad del perfil (activity_profile) nunca termina, es decir, aun llamando a las distintas actividades (recursos y archive_explorer), profile nunca deja de existir como actividad, corriendo en segundo plano. Esto se diseñó así para dar a los listeners la posibilidad de estar siempre disponibles para posibles peticiones. Si esto no fuera así, en el momento que se estuviera en otra actividad (recursos o archive_explorer) si un amigo lanzase una petición para poder visualizar nuestros archivos

compartidos, la petición se perdería porque al haber finalizado la actividad de perfil, los listeners serían eliminados junto con la actividad en el momento en que cambias. Por lo que surge la necesidad de tenerla siempre abierta para posibles peticiones por parte de amigos o terceros.

En otro sentido, es también necesario no terminar con la actividad sino dejarla en un segundo plano, en el aspecto de que cuando ejecuta alguna otra de las 2 actividades disponibles, son para recuperar algún tipo de dato o recurso, ya sea un archivo que se quiera compartir o un archivo que queramos descargar de algún amigo (la descarga se realiza en perfil). Por lo que podemos determinar que las 2 actividades (recursos y archive_explorer) son actividades de apoyo a la principal que es perfil.

En el aspecto de eficiencia también es más eficaz esta decisión de diseño que consiste en dejar perfil siempre existiendo en la máquina virtual de Java ya que se sabe que Java consume muchos recursos en el momento de crear/destruir objetos, por lo que una sobrecarga de creación y destrucción de la actividad de Perfil supondría una gasto desproporcionado de recursos del móvil llevando incluso a poder incurrir en un error de memoria lo que desembocaría en un aborto de la ejecución de la aplicación, y eso es inadmisible para lo que se quiere. Por lo que decidir que perfil no deje de existir en todo momento en el que la aplicación esté abierta nos lleva también a ahorrar en recursos y en agilizar la aplicación evitándose sobrecarga de ejecuciones a la hebra principal de la aplicación.

Al transmitir un archivo, se ha buscado un estándar de codificación que permita de forma fácil hacer una transmisión segura punto a punto, evitando mandar los archivos de forma directa desprotegidos y vulnerables a cualquier a cualquier intercepción que pueda ocurrir. El método de codificación que se ha usado ha sido Base64. Los motivos son que la codificación Base64 crece sólo al 133% del tamaño inicial, siendo bastante eficiente y que la forma de pasar los archivos es en binario y de ese binario con Base64 lo pasamos a string. El modo de aplicarlo en la App ha sido codificando entero el archivo a pasar (de binario a un

string/cadena) de forma que podemos trocear esa cadena codificada y enviarla por partes de tal forma que no haya problemas por la longitud de la cadena codificada que enviamos.

Al principio existía un problema de desbordamiento al codificar el archivo y es que en la lectura del archivo binario se hacía uso de un buffer de entrada, el cual intentaba alojar el archivo por completo en memoria, provocando el desbordamiento de la memoria asignada a la aplicación. Esto supuso un problema ya que la función principal de la aplicación hacía que esta misma se colgase no aportando ningún tipo de funcionalidad. El problema se solucionó eliminando el buffer de entrada intermedio y haciendo que el programa leyese directamente del almacenamiento del móvil, ya que después de discutirse se acordó que el almacenamiento de un móvil al ser tipo flash, normalmente es una tarjeta SD, no supone un cuello de botella como si podría suponerlo un disco duro mecánico en un ordenador con archivos muy grandes.

Existen otras muchas clases creadas en la aplicación que no poseen una actividad propia y que sirven como apoyo a perfil delegando ciertas tareas o abstrayendo ejecuciones necesarias para la aplicación. A continuación se procede a describir cada una de ellas:

- **DatabaseHelper.java**, clase java que extiende a la clase SQLiteOpenHelper. Esta clase controla y abstrae de una manera eficaz el manejo de amigos en una base de datos de tipo SQLite. Dispone de distintos métodos:
    - DatabaseHelper (constructora): crea la base de datos si no existe.
    - onCreate: crea las distintas tablas en la base de datos si no existieran.
    - onUpgrade: método llamado cuando se necesita actualizar la base de datos. La implementación debe llamar a este método cuando se necesite añadir o quitar tablas, cambiando así su número de versión.
    - addData: método creado para la inserción de un dato en la base de datos, en este caso de un amigo.
    - getData: método creado para devolver un cursor que apunte a los datos existentes de la base de datos.

- **ArchivesDatabase.java**, clase java que extiende a la clase SQLiteOpenHelper. Esta clase controla y abstrae el manejo de archivos compartidos en la aplicación usando para ello una base de datos de tipo SQLite. Dispone de los distintos métodos:
    - DatabaseHelper (constructora): crea la base de datos si no existe.
    - onCreate: crea las distintas tablas en la base de datos si no existieran.
    - onUpgrade: método llamado cuando se necesita actualizar la base de datos. La implementación debe llamar a este método cuando se necesite añadir o quitar tablas, cambiando así su número de versión.
    - addData: método creado para la inserción de un dato en la bbdd, en este caso de un archivo.
    - getData(String): devuelve un cursor apuntando al dato solicitado mediante el parámetro String.
    - getData(): devuelve un cursor apuntando al primer registro y recorre todos los registros existentes en la bbdd.
    - exists(String): devuelve un booleano confirmando si existe el objeto pasado por parámetro.

- **Friends.java & FriendsAdapter.java**, ambas son clases para conformar la lista de amigos en el perfil. Friends.java es la clase que crea el "objeto amigo" y FriendsAdapater.java es la clase que adapta el objeto a un ListView de Android para que pueda visualizarse correctamente.

- **Constants.java**, clase java que guarda constantes útiles para la aplicación, tales como las keys de conexión con PubNub.

4.2 Patrones de Capas implementados en la Aplicación

Como parte de la ingeniería de software que se ha utilizado al crear la aplicación Android, se han usado distintos patrones de capas vistos durante estos años en varias asignaturas como Ingeniería del Software y Modelado de Software, estos patrones aportan buenas prácticas o mejoras de funcionalidad sobre el código que se implementan, haciendo

más fácil su usabilidad, lectura y facilitando la modularidad entre componentes, o en el caso de Android entre las distintas actividades que se generan al ejecutar la aplicación. Estos patrones son los que se han implementado:

4.2.1 Patrón Observer

"Patrón de diseño de software que define una dependencia de tipo uno a muchos objetos, de manera que cuando uno de los objetos cambia su estado, notifica este cambio a todos los demás objetos dependientes. El motivo principal del uso de este patrón es su utilización como un sistema de detección de eventos en tiempo de ejecución."[15]

Este patrón tiene un uso muy concreto: varios objetos necesitan ser notificados de un evento y cada uno de ellos decide cómo reaccionar cuando el evento se produzca, evitando al cliente tener que estar pendiente de cada cambio que se produzca en el observable, ya que éste avisa a todos los clientes suscritos de sus cambios.

```
this.mPubNub.subscribe(stdbyChannel, new Callback(){ //creamos nuestro canal y nos
    @Override
    public void successCallback(String channel, Object message) { //despierta cuand
        Log.v("MA-success", "MESSAGE: " + message.toString());
        if (!(message instanceof JSONObject)) return; // Ignore if not JSONObject
        JSONObject jsonMsg = (JSONObject) message;
        try {
            if (!jsonMsg.has(Constants.JSON_CALL_USER)) return;
            connectPeer("", false);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
});
```

Figura 19: fragmento de código donde se usa el patrón Observer

4.2.2 Patrón Singleton

"Patrón de diseño que permite restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Garantiza que una clase sólo tenga una instancia y proporciona un punto de acceso global a ella."[16]

En la siguiente figura se puede ver que el patrón Singleton ha sido utilizado para instanciar el cliente RTC y evitar que cada vez que haya que hacer uso de esa instancia se cree una nueva distinta de la ya creada:

```
if(this.pnRTCClient == null) {
    PeerConnectionFactory.initializeAndroidGlobals(
            getApplicationContext(),  // Context
            true,  // Audio Enabled
            true,  // Video Enabled
            true,  // Hardware Acceleration Enabled
            VideoRendererGui.getEGLContext()); // Render EGL Context

    PeerConnectionFactory pcFactory = new PeerConnectionFactory();
    this.pnRTCClient = new PnRTCClient(Constants.PUB_KEY, Constants.SUB_KEY, this.username);

    MediaStream mediaStream = pcFactory.createLocalMediaStream(LOCAL_MEDIA_STREAM_ID);

    this.pnRTCClient.attachRTCListener(new myRTCListener());
    this.pnRTCClient.attachLocalMediaStream(mediaStream);

    this.pnRTCClient.listenOn(this.username);
}
```

Figura 20: fragmento de código donde se usa el patrón Singleton y el patrón Factory

4.2.3 Patrón Factoría Abstracta

Patrón que proporciona una interfaz (no define los métodos, sólo los declara) para crear un objeto o grupos de objetos relacionados o que dependan entre sí, definiendo cómo será la clase constructora en la que se van a basar los objetos que implementan la factoria.

En la figura 20, además de la instanciación del cliente RTC, también se puede ver el uso del patrón Factoría Abstracta en forma de una factoría para la conexión entre peers.

4.3 Código relevante

Lo más relevante que podemos destacar del código contenido dentro de toda la aplicación es la lógica de conexión, dado que la aplicación se basa en compartir archivos mediante conexiones peer-to-peer.

Como comentábamos al principio, la señalización de los distintos dispositivos la realizamos mediante PubNub que nos ofrece este servicio gratuito de hasta 50 conexiones simultáneas. Ideamos que la aplicación al entrar o iniciarse, es decir, al llamarse al onCreate de la actividad perfil dado que es la actividad principal, nos suscribieramos al servidor de PubNub de manera que pasaremos a un estado de stand-by, y esto es lo que hace el siguiente código, darse de alta con un usuario (por el cual llamarlo) y deja un callback preparado para cualquier aviso desde el server de PubNub.

```
public void initPubNub(){
    String stdbyChannel = this.username + Constants.STDBY_SUFFIX;
    this.mPubNub = new Pubnub(Constants.PUB_KEY, Constants.SUB_KEY);
    this.mPubNub.setUUID(this.username);
    try {
        this.mPubNub.subscribe(stdbyChannel, new Callback(){ //creamos nuestro canal y nos quedamos
            @Override
            public void successCallback(String channel, Object message) { //despierta cuando alguien
                Log.v("MA-success", "MESSAGE: " + message.toString());
                if (!(message instanceof JSONObject)) return; // Ignore if not JSONObject
                JSONObject jsonMsg = (JSONObject) message;
                try {
                    if (!jsonMsg.has(Constants.JSON_CALL_USER)) return;
                    connectPeer("", false);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    } catch (PubnubException e) {
        e.printStackTrace();
```

Figura 21: función initPubNub de la clase Profile

El método “subscribe” de la librería de PubNub es el que nos agrega al servidor de manera que podamos ser contactados. Igualmente existe un método en PubNub llamado “publish” que despierta al callback del usuario al que quiere conectarse e inicia la conexión al mismo.

```
private void publish(final String connectTo, final String connectionType){
    String userCall = connectTo + Constants.STDBY_SUFFIX;
    JSONObject jsonCall = new JSONObject();
    try {
        jsonCall.put(Constants.JSON_CALL_USER, username);
        mPubNub.publish(userCall, jsonCall, new Callback() {
            @Override
            public void successCallback(String channel, Object message) { //conectamos nosotros
                Log.d("MA-dCall", "SUCCESS: " + message.toString());
                connectPeer(connectTo, true); //conectamos con el peer

                if(connectionType.equals("VAR")){ //buscamos que tipo de mensaje debemos enviar
                    VAR(connectTo);
                }else if(connectionType.equals("FR")){
                    FR(connectTo);
                }
            }
        });
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
```

Figura 22: función publish de la clase Profile

Del código podemos destacar que una vez llamado al otro, este nos devuelve una señalización en forma de ACK que activa nuestro callback de manera que sabemos que el Peer existe y está disponible, por lo que una vez sabido esto comenzamos con el establecimiento de la conexión punto a punto. En todo momento quisimos dejar que el que se encargara de la

conexión fuese aquel que llama, de manera que aquel que es contactado debe permanecer pasivo hasta que se establezca con él una conexión punto a punto.

Debemos destacar que el formato de los mensajes por los cuales se comunican ambos extremos o peers es “Json” por lo que esto nos ha supuesto un desafío en el momento de codificar los distintos archivos que el usuario quisiese pasar, sobre todo los archivos binarios que tuvimos que codificarlos en Base64[17] para formar un String de manera que pudiéramos enviarlo utilizando Json como nuestro formato de mensaje.

```
private void connectPeer(String connectTo, boolean call){
    if(this.pnRTCClient == null) {
        PeerConnectionFactory.initializeAndroidGlobals(
                getApplicationContext(),  // Context
                true,  // Audio Enabled
                true,  // Video Enabled
                true,  // Hardware Acceleration Enabled
                VideoRendererGui.getEGLContext()); // Render EGL Context

        PeerConnectionFactory pcFactory = new PeerConnectionFactory();
        this.pnRTCClient = new PnRTCClient(Constants.PUB_KEY, Constants.SUB_KEY, this.username);

        MediaStream mediaStream = pcFactory.createLocalMediaStream(LOCAL_MEDIA_STREAM_ID);

        this.pnRTCClient.attachRTCListener(new myRTCListener());
        this.pnRTCClient.attachLocalMediaStream(mediaStream);

        this.pnRTCClient.listenOn(this.username);
    }

    if(call){
        this.pnRTCClient.connect(connectTo);
    }
}
```

Figura 23: función connectPeer de la clase Profile

El método connectPeer crea una instancia de cliente RTC si no existe y prepara todo lo necesario para que el peer pueda comunicarse. Esto se realiza solo cuando el cliente no existe, por lo que podríamos estar hablando de un patrón singleton reflejado en el código. Cuando creamos el cliente, instanciamos una factoría que se encarga de reunir todo lo necesario para la comunicación del cliente con otro peer. Para eso creamos unos listeners privados (myRTCListener) que nos permite gestionar de qué manera se comunican los peers.

El RTCListener privado nos permite manejar de una manera personalizada la interacción entre ambos dispositivos. A continuación se puede observar esta clase privada:

```
private class myRTCListener extends PnRTCListener{
    @Override
    public void onPeerConnectionClosed(PnPeer peer) {
        super.onPeerConnectionClosed(peer);
    }

    @Override
    public void onLocalStream(MediaStream localStream) {
        super.onLocalStream(localStream);
    }

    public void onConnected(String userId){
        Log.d("Md-a", "connectado a: " + userId);
    }

    @Override
    public void onMessage(PnPeer peer, Object message) {
        if (!(message instanceof JSONObject)) return; //Ignore if not JSONObject
        final JSONObject jsonMsg = (JSONObject) message;
        try {
           final String type = jsonMsg.getString("type"); //TODO el manejo de los mensaje
           if(type.equals("VAR")){
               VAL(jsonMsg);
           }else if(type.equals("VAL")){ //se debe manejar en la hebra principal ya que i
               Profile.this.runOnUiThread(new Runnable() {
                   @Override
                   public void run() {
                       handleVAL(jsonMsg);
                   }
```

Figura 24: clase privada myRTCListener en la clase Profile

Podemos observar en la imagen del código como nuestra clase privada extiende a la abstracta PnRTCListener y nos permite manejar la comunicación entre ambos dispositivos de una manera muy cómoda.

Ya que a nosotros nos interesa el manejo del canal de datos, manejamos sobre todo los mensajes mediante **onMessage**. Otros métodos que implementa la clase que hereda de RTCListener son **onPeerConnectionClosed**, clase que es llamada cuando se cierra una conexión y simplemente llama al método de la clase superior para que maneje el cierre de la conexión. **onLocalStream**, método llamado para adjuntar el stream de datos al listener propio, el cual también llama al método de la clase superior para que maneje lo necesario. **onConnected** es un método de debug ya que nos muestra el id del usuario al que nos hemos conectado.

Si establecemos un esquema que represente el handshake y la conexión entre ambos dispositivos y el servidor de PubNub para la señalización, este quedaría de esta manera:

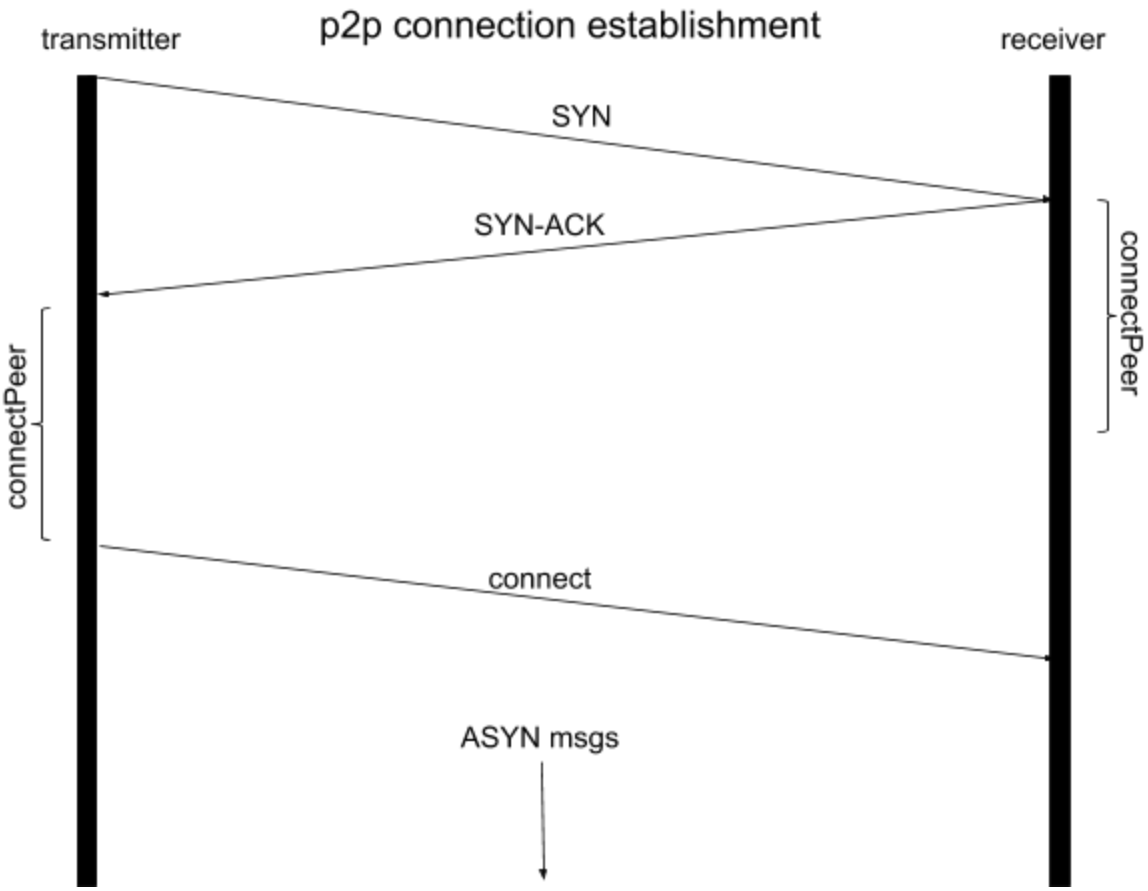

Figura 25: esquema de conexión entre dos dispositivos

Transmitter representa al dispositivo que quiere conectarse a receiver. Ambos utilizan la información del servidor de PubNub y lo utilizan para señalizarse y este les envía la información necesaria para que ambos realicen la conexión en lo que podríamos denominar como un handshake de 3 vías:

Primeramente transmitter se comunica con receiver con un SYN.

Seguidamente receiver responde con un SYN-ACK y comienza su lógica para preparar la conexión

Para terminar, cuando transmitter recibe el SYN-ACK de receiver comienza su lógica para la conexión y finalmente él es el encargado de establecer la conexión. A partir de ahí ambos dispositivos podrán comunicarse en ambas direcciones ya que se habrá creado y establecido un canal de comunicación.

Todo el código de la aplicación está disponible en:
https://github.com/jpradas/P2P-Android-App.git

4.3.1 Funcionalidad

Después de tratar de explicar el modelo de datos de la aplicación, cuales son los datos que guarda y cómo los utiliza para la conexión y compartición de archivos, se procederá a describir unos casos de uso esperados para la aplicación.

Modelo de datos

Los datos necesarios para utilizar la aplicación son simples: Un nick. Eso es lo que nos pedirá la aplicación la primera vez que la iniciemos en nuestros teléfonos. Este nick se utilizará con tus amigos como un identificador único y será la información que ellos tendrán de tí. Gracias a la gestión de PubNub de los UUID's para la conexión se gana en seguridad ya que lo único que tiene un dispositivo de información con respecto a los demás es su nick, en nuestro caso particular.

Para gestionar la lista de amigos que se van agregando en un dispositivo se ha utilizado una base de datos de tipo SQLite, del cual se guarda un id interno creado en el momento de agregar al amigo y su nick, que será el UUID que PubNub utilizará para gestionar la conexión.

Para la lista de archivos que se comparten con tus amigos se ha utilizado otra base de datos tipo SQLite, de modo que de cada archivo se guarda su nombre y su path al momento de añadirlo como archivo compartido. Se decidió esta manera de gestionar los archivos ya que así no hacía falta copiar los archivos a una carpeta concreta del dispositivo, simplemente decidimos que archivo queremos compartir, dando igual su ubicación en el dispositivo, y se hace visible para tus amigos.

En cuanto a lo que tus amigos pueden ver de archivos compartidos, únicamente podrán visualizar los nombres de los archivos, no su ruta ya que esta es gestionada por el dispositivo hospedador y en ningún momento se comparte información sensible de nuestra estructura de datos interna del dispositivo.

Contacto con peers

PubNub gestiona un servidor alcanzable mediante unas keys (pública y de suscripción), los distintos UUID's y su contacto entre ellos. Para permitir elegir a los peers a los que pasar los archivos hay que agregarlos como amigos, mandándoles una petición de amistad, que una vez aceptada permite a ambas personas descargarse los archivos compartidos por ambos. ¿Cómo funciona esto?

Como se ha comentado el trabajo lo realiza PubNub. Cuando la aplicación nos pide un nombre de usuario, ese nombre de usuario se convierte en nuestro identificador, seremos conocidos, por nuestros amigos (y no amigos) existentes en el servidor, por ese nick. De manera que si alguien quiere hacerse amigo nuestro para compartir archivos, deberá descubrirnos primeramente enviando una petición de amistad con nuestro nick. De esta manera, PubNub gestionará si existe en su servidor (aquel que se identifica por las keys pública y de suscripción) alguien con ese nombre y le sincronizará con tu dispositivo de

manera que puedan comunicarse. De esta manera, cuando el otro peer acepte nuestra petición, seremos alcanzables por él, y él por nosotros utilizando únicamente su nick almacenado en nuestra base de datos de amigos. En el apéndice A, manual de uso de la app, apartado Pantalla de envío de solicitud de amistad se expone en un caso de uso esta funcionalidad.

5. Resultados

5.1 Conclusiones

En este TFG se ha desarrollado una aplicación en Android para poder enviar archivos punto a punto sin intermediarios. La aplicación responde a la cuestión actual de ¿donde se alojan los archivos que comparto con mis amigos, tanto en WhatsApp, como con Twitter, Facebook Messenger, etc? Todas estas aplicaciones móviles siguen un patrón estándar de cliente-servidor, siendo el servidor el que maneja la información y la sirve al cliente. Al compartir archivos mediante estas plataformas con tus amigos, está quedando una copia del mismo alojada en los servidores de estas compañías, por lo que podríamos llegar a preguntarnos, ¿hasta qué punto tengo privacidad con estas aplicaciones?

En esta necesidad, o búsqueda de la privacidad, es donde aparece el sentido de nuestra aplicación. Esta hace uso de unos estándares ya existentes, no inventa nada nuevo, pues las redes P2P existen de hace mucho tiempo y la conexión punto a punto es tan antigua como internet, pero lo novedoso de esto es que está diseñado para dispositivos móviles. Ha sido interesante poder diseñar algo que no existe en el mercado, existen aplicaciones para compartir con tus amigos, pero estas no salen del ámbito de red privada, por lo que nosotros vamos más allá y buscamos compartir archivos donde quiera que te encuentres con cualquier persona que tengas como amigo.

Nuestro principal problema a la hora de realizar la aplicación fue conseguir conectar a las dos personas a través de IPv4, por culpa de las NAT y demás soluciones que las ISP han adoptado para evitar desplegar IPv6 y mantener IPv4 funcional extendiendo el número de direcciones IP de forma virtual.

Por todo esto sería necesario implementar el estándar IPv6 sustituyendo al arcaico estándar IPv4 para independizar la implementación de cualquier protocolo de conexión P2P del uso de firewalls o routers por parte de los peers que realizan la conexión, siendo trivial el método de conexión entre esos peers al no haber el problema de desbordamiento de direcciones IP que IPv4 crea, obligando el uso de NAT.

La conectividad entre dos peers no es tan sencilla como pensábamos antes de iniciar este TFG. Las NAT's, con su firewall, es un sistema muy potente de redireccionamiento y seguridad para proteger las máquinas en un ámbito privado del internet público, pero que limita en sobremanera la conexión entre dos máquinas bajo una LAN distinta.

Según avanzamos en el desarrollo de este TFG nos hemos dado cuenta de la transformación que el propio término P2P ha tenido según la evolución tecnológica ha ido dando forma y aplicando soluciones que parten de la idea original de P2P pero que acaban siendo otra forma distinta del concepto P2P, un protocolo de la capa de aplicación.

Teóricamente el P2P inicial era distribuido, cada nodo disperso en la red y con comunicación entre pares sin ningún tipo de servidor o ayuda externa a la conexión, pero esta definición no cubre el amplio espectro de soluciones tecnológicas que se necesitan en el intercambio de archivos en la vida real. Por esto no tiene sentido aplicar la idea inicial de P2P y se desarrollaron distintas topologías que cubrían parte de esas necesidades tecnológicas, aun así la idea de P2P descentralizado también chocó con los sistemas que se iban generando en la red, pues la gran mayoría usaba un servidor como intermediario de las comunicaciones y facilitador de conexiones, con lo que se mezcló el P2P con arquitecturas donde había servidores presentes para facilitar el intercambio de archivos y se generaron las soluciones híbridas.

La estructura de la red dentro del concepto P2P es clave, pues la forma de llevar a cabo la conexión y el manejo de errores son muy diferentes si cada nodo de la red es independiente (red distribuida), pide información como apoyo a un servidor (red descentralizada) o canaliza toda la acción a través de un servidor (red centralizada).

Nosotros hemos recorrido en cierto modo el camino inverso al desarrollo del término P2P, pues hemos crecido usando aplicaciones que tenían el nombre de P2P asociado como puede ser eMule o Bittorrent, pero que pertenecen a esa parte de desarrollo de soluciones con topologías centralizadas y/o uso de servidores para facilitar la conexión. Con esta información quisimos investigar la factibilidad de generar una aplicación Android que usase los conceptos iniciales de P2P, pero debido a los problemas técnicos que encontramos para aplicar una solución que cuadre con eso (mayormente debido a los problemas con la NAT transversal) y el pragmatismo que quisimos imprimir a la aplicación, nos decantamos por utilizar una opción que manteniendo una conexión punto a punto, usase un servidor de señalización (PubNub) para evitar los problemas de conexión por la NAT transversal y con eso ya poder gestionar la forma de intercambio de archivos punto a punto una vez se ha establecido la conexión. La evolución del proyecto ha partido del concepto de conexión con servidor y red P2P estructurada para buscar una solución sin servidor y con una red totalmente desestructurada para al final acabar con una especie de híbrido que usa un servidor solo para conectar y, una vez hecha la conexión, gestionar el intercambio punto a punto para mantener la red desestructurada.

Realizando el TFG nos hemos dado cuenta de cómo el término P2P se ha desvirtuado, cualquier tecnología capaz de intercambiar entre dos pares se le pone la etiqueta de P2P, sin tener en cuenta la ni la topología ni la estructura de la red de conexión. Hasta que no hemos investigado el tema, éramos absolutos desconocedores de las diferencias tanto tecnológicas como semánticas que implican.

Reflexionando sobre este asunto, el debate también ha llegado a ser casi un debate filosófico, pues todo esto es tecnología al servicio del que la usa al fin y al cabo, con lo que todos estos conceptos de topología o estructura están supeditados a la capacidad de uso que

puedan tener como solución a los problemas de intercambio de archivos que puedan haber en el futuro, como ya lo han estado en el pasado. Por ejemplo, una topología de red no estructurada se acerca más al P2P original, pues cada par es independiente de la red en la que está, pero esto para sistemas donde se quiera garantizar la disponibilidad y éxito en el intercambio de archivos no encaja, con lo que para dar mayor robustez y eficiencia a la red hay que usar una topología estructurada. En base a estos tipos de adaptaciones el P2P ha trascendido su concepto original.

Partiendo de esa idea de P2P inicial y debido a nuestros conocimientos de Java en la carrera, hemos enfocado la tarea del desarrollo Android en su concepción original, trabajando con Android nativo (programando en Java) en vez de utilizar alternativas con otros lenguajes y compiladores, ya sean en forma de frameworks, transpiradores de algún lenguaje a Java u otras herramientas alejadas en cuanto programación de la estructura nativa de Android que abstraiga esa parte.

Android es un sistema relativamente complejo de entender inicialmente partiendo desde la perspectiva de Java, pues aunque la orientación a objetos y demás conceptos del lenguaje persisten, el modelo de manejo de Android es diferente. Android se basa en actividades, que funcionan de forma secuencial y recursiva, pues partiendo de una actividad principal se crean las demás actividades, una encima de otra formando una pila de actividades que hay que manejar con cuidado para saber en cada momento qué parte de tu código se está ejecutando o no. Para cada actividad Android sigue esta pila de llamadas: onPause, onStop, onDestroy, onCreate, onStart, onResume. Primero Android se encarga de liberar las actividades que consumen recursos de forma innecesaria según sus cálculos y la capacidad del dispositivo en el que se ejecute y luego crea o reanuda las actividades necesarias. Estos principios los hemos tenido que tener en mente al programar la app de forma que durante un intercambio de archivos no se destruya la actividad que se encarga de la misma. Para evitar este problema de interrupción o desconexión durante el intercambio, la actividad inicial se encarga de hacer el intercambio, siendo esta la última que se destruiría. Aun con eso, en caso de necesidad de recursos extrema, Android podría llegar a destruir la actividad principal de la app, por lo que no podemos garantizar que siempre se lleve a buen puerto el intercambio de archivos, pero en la inmensa mayoría de veces no habrá problemas.

El desarrollo del TFG nos ha hecho tener que aprender tanto el funcionamiento de las múltiples formas de conexión de dos o más personas que hemos expuesto, como también del desarrollo de un aplicación Android.

De todo este TFG hemos conseguido crear una aplicación Android funcional, que se puede lanzar en el market de Google, con la que compartir archivos sin intermediarios que puedan guardar el registro del archivo o el propio archivo en sus servidores. La aplicación pide solo el nombre la primera vez que se inicia su uso y a partir de ahí recupera ese nombre cada vez que se inicia. La compartición de archivos se hace con una codificación en la transferencia (Base64) para evitar que una intercepción en la conexión P2P consiga obtener directamente los datos transferidos entre los peers. Además, la aplicación permite intercambiar desde archivos TXT, PDF o cualquier otro tipo de texto, hasta imágenes o vídeos sin perder calidad en la transferencia al no hacer ninguna compresión de los archivos.

5.2 Trabajo Futuro

Como trabajo futuro que se pueda realizar quedan algunos puntos que desde una perspectiva de aplicación sería interesante desarrollar e implementar dentro de la aplicación, dejándose a hacer para futuros TFGs:

- Añadir la compartición de archivos de PubNub sin límite. Actualmente, por el tipo de versión utilizado de WebRTC (libjingle 9694) existen varias restricciones en cuanto a la compartición de archivos. La primera de ellas es que existe un límite máximo del tamaño de los mensajes, por lo que archivos muy grandes deben ser troceados y enviados parte por parte. La segunda restricción que tiene la versión de libjingle es que el único formato de mensaje soportado es JSON en nuestro caso, por lo que todos los ficheros binarios (imágenes, pdfs, vídeos, etc) deben ser codificados en Base64 para que tengan un formato String que pueda ser enviado en JSON y posteriormente decodificado en el dispositivo receptor. Todos estos impedimentos podrían evitarse si se utiliza una función que implementó PubNub que permite la compartición de

archivos sin límite, y no solo hablamos del límite, sino que también permite enviar ficheros binarios sin tener que codificarlos, lo que resulta más cómodo. Esto además mejoraría el rendimiento de la aplicación ya que no se gastaría tiempo de ejecución en tener que codificar y decodificar [18][19].

- Previsualización de ficheros pre-compartir. No se logró diseñar un método que sirviera para poder previsualizar el tipo de documento antes de compartirlo, por lo que quedó en un método de exploración de archivos muy básico que muchas veces puede inducir a confusión del archivo compartido. De esta manera, se habló de diseñar una previsualización de archivos, de modo que se pueda saber de manera certera qué es lo que se está compartiendo sin tener que ir comparando el nombre de archivo que tiene. De esta manera se podría llamar al método que tiene nativo el propio sistema Android que permite abrir distintos tipos de archivos con una aplicación instalada que sepa leer el tipo de archivo.

- Compartición de carpetas completas. Una de las ideas iniciales que no se terminó de implementar era tener la posibilidad de, no solo compartir archivos individuales, sino poder compartir carpetas completas de archivos. De esta manera estaríamos haciendo que lo que se comparte no es solo un directorio con archivos, sino que le añade complejidad en el aspecto de poder compartir un directorio de directorios, junto con todo lo que este contenga.

- Grupos de amigos. Se consideró también la posibilidad de tener grupos de amigos a los que compartir cosas. De esta manera, no solo tendríamos una carpeta genérica que cualquier amigo que tengamos pudiese ver, sino que podríamos de alguna manera dirigir lo que cada amigo puede ver, ya que si formamos un grupo de amigos podríamos asociar ciertos archivos que compartir con ese grupo específico de amigos.

- Publicación en Play Store. Esto puede hacerse según el criterio de cada grupo de personas que compongan un TFG futuro a este. En nuestro código inicialmente pusimos una licencia MIT la cual permite utilizar el código para construir una aplicación

propia, de esta manera nuestro software creado es de código abierto y libre. Sin embargo, a modo de trabajo futuro, cualquier otro que diseñe una aplicación utilizando nuestro trabajo, podría mirar la opción de lucrarse con la aplicación que cree subiéndola al Play Store (y al Apple Store si se escribe la aplicación sobre un Apache Cordova que permite pasar una aplicación de un SO a otro).

- Encriptación de seguridad de los mensajes. Podría implementarse este punto como un añadido de seguridad a la misma. Una encriptación de extremo-a-extremo añadiría más valor a la aplicación.

- Link de amistad. En un punto del desarrollo se pensó en crear un sistema que generase un link para que pudieras añadir a un amigo pinchando en el mismo. Finalmente no se añadió por otras causas, pero sería un punto interesante a desarrollar ya que añadiría versatilidad a la aplicación y usabilidad, a parte de añadir intuitividad ya que aplicaciones de mensajería instantánea como Whatsapp ya desarrollan este método de añadir amigos.

- Restringir 2 nombres iguales en la red. Hasta ahora no existe restricción para que en la lista de amigos tengas a alguien que se llame igual que tu, lo que finalmente puede resultar en errores. No se ha encontrado una solución a esto, por lo que para un trabajo futuro podría estudiar si PubNub da herramientas para gestionar este hecho que actualmente se podría producir.

- Crear una red distribuida P2P. Como último punto dentro de trabajo futuro, y el más interesante que se podría desarrollar, podría intentarse crear una aplicación que sirva como cliente para crear una red P2P distribuida (es decir, totalmente descentralizada). La aplicación actualmente se centra en un modo de compartición punto a punto, pero no crea una red P2P ya que está limitada al ámbito de tus amigos. Si se extendiese esta opción, podría tenerse una red en la que todos estuviesen conectados, aunque no fuesen exactamente amigos, y crear así una red no estructurada y aplicar algoritmos usados en estas redes para buscar archivos como la “búsqueda por inundación”. Este

punto le daría mucho más juego a la aplicación y sin duda podría suponer algún tipo de revolución dentro del mercado de apps móviles actual.

Bibliografia

[1] "What Is a Socket? (The Java™ Tutorials > Custom Networking > All ...." - https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html.

[2] "WebSockets | MDN." - https://developer.mozilla.org/es/docs/WebSockets-840092-dup.

[3] "eMule-Project.net" - http://www.emule-project.net/.

[4] "BitTorrent." - http://www.bittorrent.com/.

[5] "Napster: Home." - https://us.napster.com/.

[6] “What is Gnutella?” - https://whatis.techtarget.com/definition/Gnutella.

[7] “Topologías de redes P2P” - https://www.redeszone.net/redes/topologias-de-redes-p2p-peer-to-peer/

[8] “Tabla de hash distribuida” - https://es.wikipedia.org/wiki/Tabla_de_hash_distribuida

[9] “Secure Hash Algorithm” - https://es.wikipedia.org/wiki/Secure_Hash_Algorithm

[10] "RFC 3489 - STUN" - http://www.rfc-base.org/rfc-3489.html.

[11] “Evaluación de WebRTC en redes moviles” -

https://riunet.upv.es/bitstream/handle/10251/89020/TOMAS%20-%20EVALUACI%C3%93N%20DE%20WEBRTC%20EN%20REDES%20MOVILES.pdf?sequence=1

[12] “UDP” - https://es.wikipedia.org/wiki/Protocolo_de_datagramas_de_usuario

[13] "SHAREit-The world's fashionable cross-platform sharing app." - http://www.ushareit.com/.

[14] "Hive2Hive · GitHub." https://github.com/Hive2Hive.

[15] “Patrón Observer” - https://es.wikipedia.org/wiki/Observer_(patr%C3%B3n_de_dise%C3%B1o)

[16] “Patrón Singleton” - https://es.wikipedia.org/wiki/Singleton

[17] “Codificación en Base64” - https://es.wikipedia.org/wiki/Base64

[18] “RTC PubNub Guide” - https://pubnub.github.io/rtc-pubnub-fileshare/guide/

[19] “WebRTC File Transfer in the Browser: PubShare | PubNub” - https://www.pubnub.com/blog/2013-06-26-transfer-files-in-the-browser-pubnub-rtc-fileshare

Figura 1 - https://profile.es/blog/que-es-blockchain-fundamentos-basicos-de-la-cadena-de-bloques/

Figura 2 - https://www.redeszone.net/redes/topologias-de-redes-p2p-peer-to-peer/

Figura 3 - https://www.redeszone.net/redes/topologias-de-redes-p2p-peer-to-peer/

Figura 4 - https://es.wikipedia.org/wiki/Inundaci%C3%B3n_de_red

Figura 5 - https://www.ssh.com/ssh/

Figura 6 - http://www.brynosaurus.com/pub/net/p2pnat/

Figura 7 - https://www.avaya.com/blogs/archives/2014/08/understanding-webrtc-media-connections-ice-stun-and-turn.html

Figura 8 - https://www.avaya.com/blogs/archives/2014/08/understanding-webrtc-media-connections-ice-stun-and-turn.html

Figura 9 - https://www.pubnub.com/blog/2013-08-01-making-peer-data-connections-in-the-browser-with-webrtc/

Figura 10 - https://www.pubnub.com/

Figura 11 - https://www.slideshare.net/ShivaIIIT/torrent-file-sharingverified

Apéndices

Apéndice A. Manual de uso de la App

Este manual de uso sirve para que los usuarios que vayan a utilizar esta aplicación sepan qué se van a encontrar en cada pantalla de la aplicación y poder resolver las dudas que surjan con el uso de la misma.

A continuación se procede a explicar el uso de la aplicación desde el momento en que esta se instala en el dispositivo móvil del usuario.

Instalación de la aplicación

Como punto de partida, el usuario ha de realizar la instalación mediante un archivo .apk generado a través del entorno Android Studio de la propia aplicación, ya que aún no se puede conseguir a través del Play Store. Una vez conseguida, se lanza el archivo .apk para proceder con la instalación de la aplicación, y una vez lanzado se podrá ver la pantalla que se muestra en la figura 26.

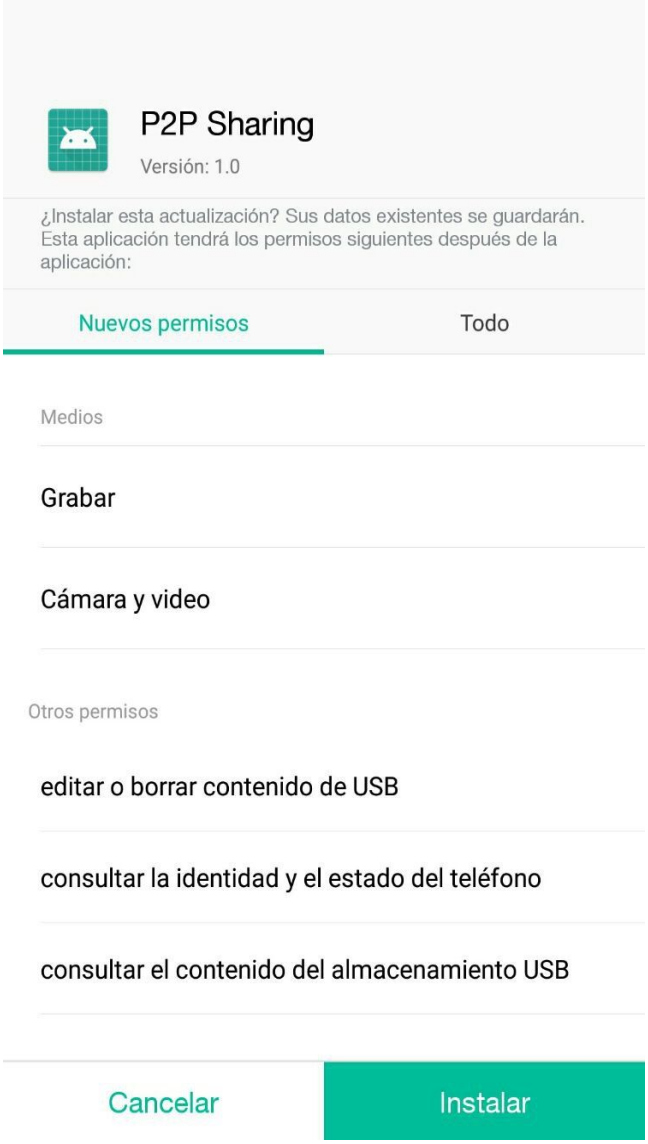

Figura 26: pantalla de instalación de la aplicación mediante la apk .

En esta pantalla el usuario tiene dos opciones bien claras y diferenciadas: cancelar o instalar la aplicación. Si se opta por cancelar se sale del instalador. Si el usuario opta por instalarla se pasa a la pantalla. Mostrado en la figura 27.

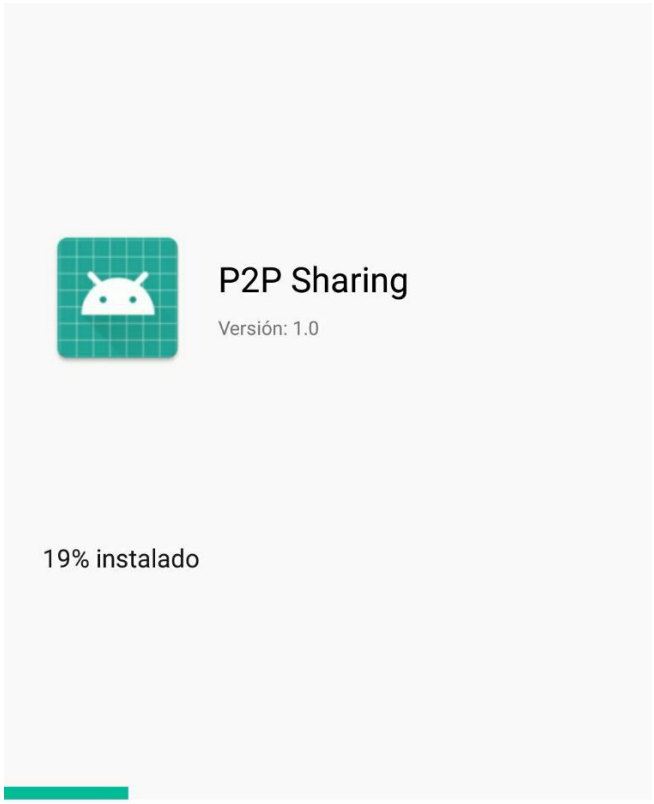

Figura 27: pantalla de progreso de la instalación de la aplicación.

En esta pantalla se puede seguir la instalación mediante una barra de progreso. Teniendo en cuenta la velocidad del/los procesador/es actuales en todos los dispositivos móviles, la instalación no debe llevar más que unos pocos segundos. Una vez finalizada la instalación, informa de qué es lo que desea hacer el usuario como se puede observar en la última pantalla del proceso de instalación. (Ver figura 28)

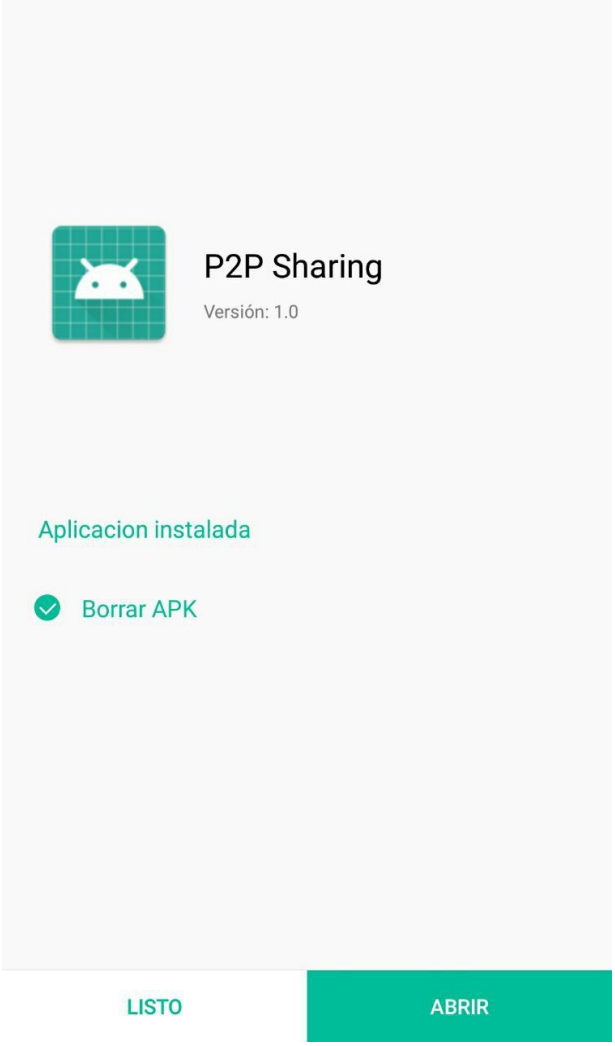

Figura 28: pantalla de información de fin de la instalación.

Tras la instalación, se muestran las dos siguientes opciones: si el usuario quiere continuar más tarde con su viaje por la aplicación puede pulsar *Listo* para salir del instalador, y si desea lanzar ya directamente la aplicación ha de pulsar la opción *Abrir*.

Pantalla de inicio

Una vez instalada la app en el dispositivo, al lanzarla el usuario se encuentra con la pantalla de bienvenida donde ha de introducir un nombre de usuario para poder identificarse. (Ver figura 29)

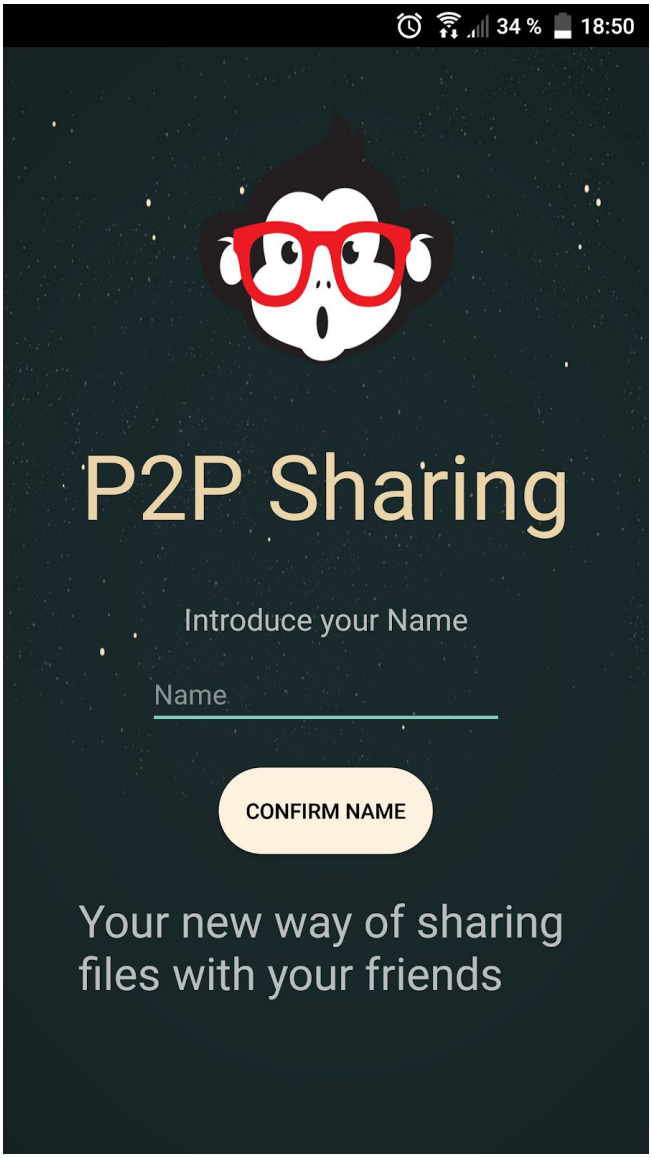

Figura 29: Pantalla de Inicio de la aplicación

Pantalla de amigos

Antes de acceder a la pantalla que actúa como pantalla principal de la aplicación, se pide al usuario su consentimiento para poder acceder al contenido de los distintos directorios

para poder elegir el contenido que se desea compartir con la siguiente ventana emergente. (Ver Figura 30)

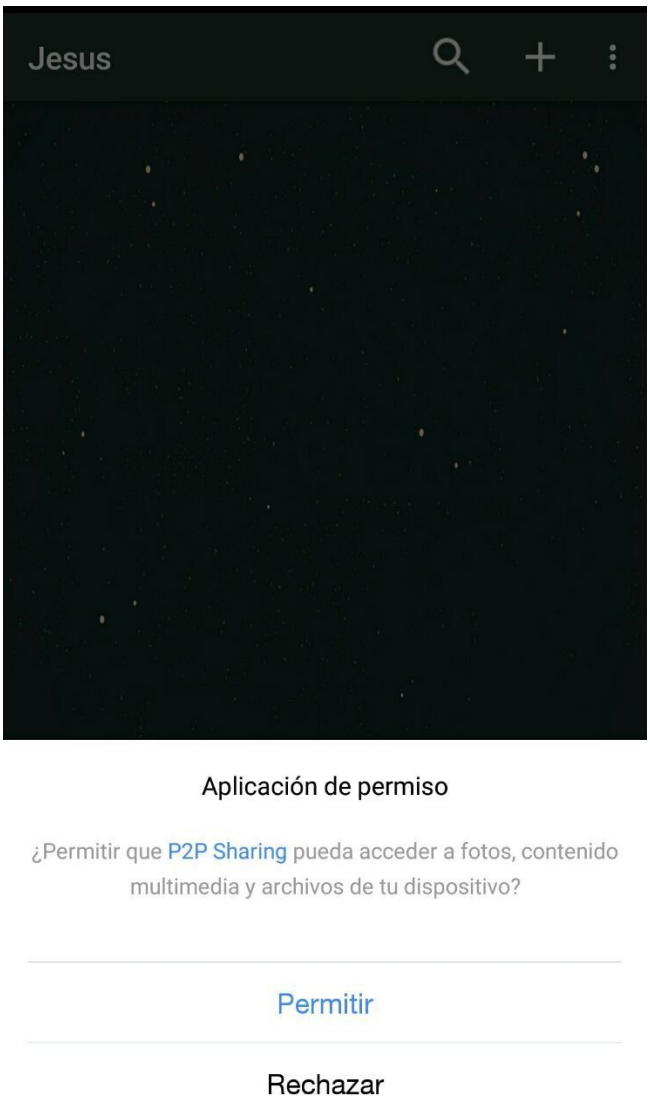

Figura 30: pantalla previa a la pantalla principal donde pide acceso al contenido multimedia.

Una vez dado el permiso anterior, el usuario es redirigido a la siguiente pantalla, la pantalla de Amigos (Figura 31). En esta pantalla, el usuario puede ver los Amigos con los que puede compartir sus archivos al haberlos buscado previamente introduciendo su nick y enviandoles una petición de amistad. Una vez aceptada por parte del usuario que recibe la petición, se mostrará en su pantalla de Amigos.

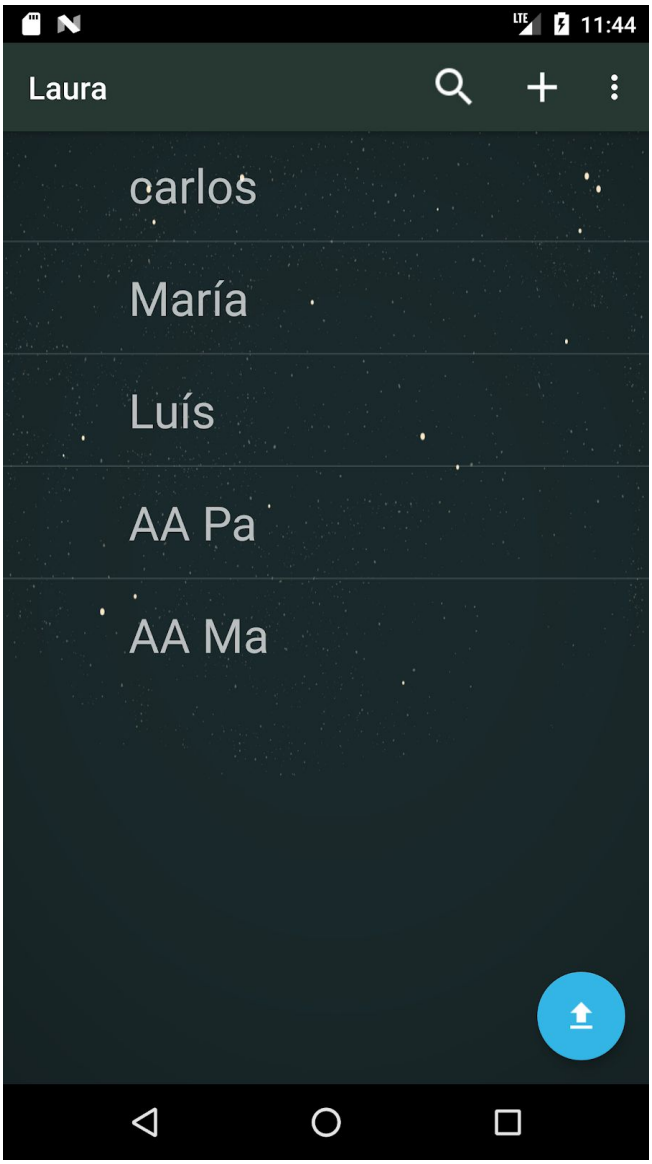

Figura 31: pantalla de Amigos.

Se puede observar que en esta pantalla aparece el nombre con el cual se ha logueado el usuario en la parte superior izquierda. En ese mismo menú superior el usuario también se encuentra tres botones y un cuarto en la parte inferior derecha de la pantalla:

- Una lupa para poder buscar a los usuarios, que previamente ya se hubieran añadido, con los cuales se desea compartir algún archivo. Al pinchar en ella, se habilita un

pequeño buscador para introducir el nombre del usuario y buscarlo entre la lista de Amigos. Útil cuando se tiene una gran cantidad de Amigos agregados en esta pantalla. Sin embargo, esta funcionalidad no terminó de implementarse por lo que no tiene uso en la aplicación actual.

- Signo + para añadir o agregar a la lista de Amigos al usuario deseado con el cual se desea compartir. Al pulsar en él aparece una pequeña ventanita emergente donde se ha de introducir el nombre de usuario al cual se desea agregar a la lista de Amigos. Una vez introducido el nombre se pulsa el botón “Add friend” para enviar una request o petición de amistad al usuario para poder convertirse en Amigos. El usuario al cual se ha enviado la petición de amistad ha de aceptarla para que se muestre en la lista de Amigos.

- Botón de opciones (3 puntitos en vertical), donde se puede acceder a la lista de archivos ya compartidos con anterioridad con los Amigos. Al pulsar en él se puede visualizar una lista de estos archivos.

- Botón de subida de archivos (círculo azul con una flecha blanca) para poder elegir qué archivo de la galería del usuario compartir con los Amigos. Al pulsar en él, conduce al usuario a una pantalla donde se pueden ver todos los directorios del dispositivo móvil para poder elegir cualquier fichero deseado. En la siguiente figura se muestra el listado de los directorios del dispositivo, para buscar la foto que se desee compartir.

Una vez se tienen amigos añadidos, al pulsar en su nombre se muestra una ventana emergente indicando dos posibles acciones a realizar: se puede eliminar al usuario de la lista de amigos, y se pueden ver los archivos que ese usuario ha subido para la compartición con sus amigos. Pulsando en cualquiera de ellos se inicia su descarga. (Ver figura 32)

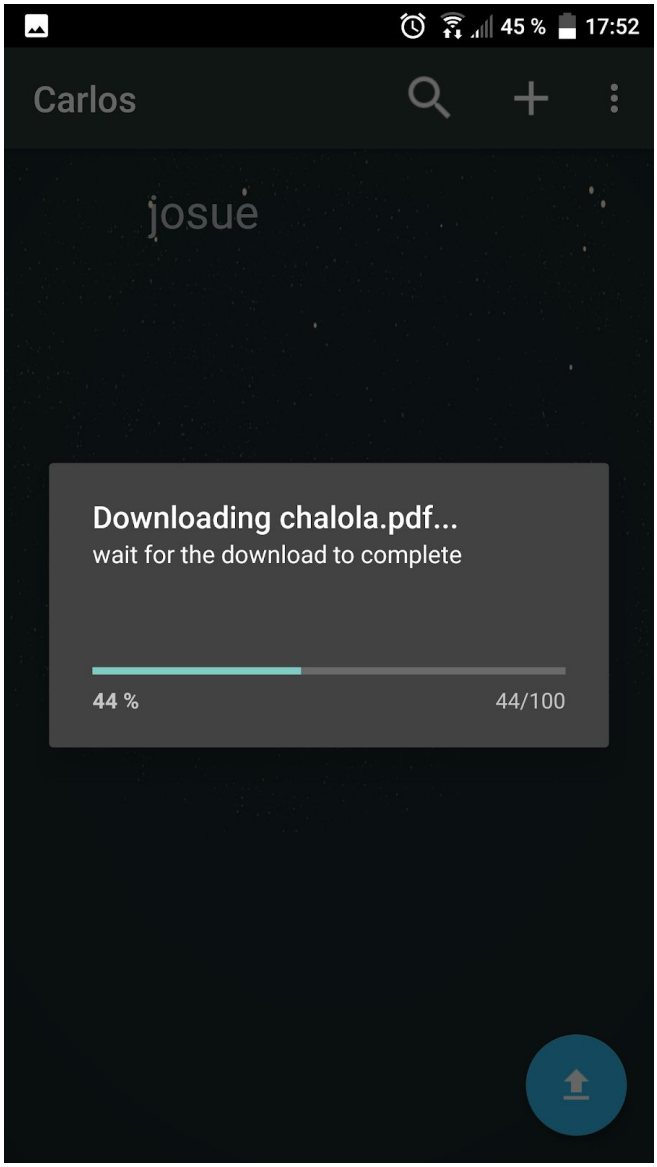

Figura 32: pantalla de ventana que indica el progreso de la descarga de archivo.

Mediante una barra de progreso como la que se muestra en la anterior figura el usuario puede ver como avanza el progreso de su descarga. Una vez haya finalizado la descarga, en la carpeta Downloads del teléfono se encontrará el archivo descargado

Pantalla de envío de solicitud de amistad

Al pulsar en el botón (+) para enviar solicitud de amistad (ver figura 33), aparece una pequeña ventanita emergente para poder buscar por el nombre al usuario con el cual se

quiere compartir el o los archivos deseados para agregarlo a la lista de Amigos. Una vez hayan aceptado la solicitud que se ha enviado, se añade dicha lista para poder ver sus archivos subidos para compartir.

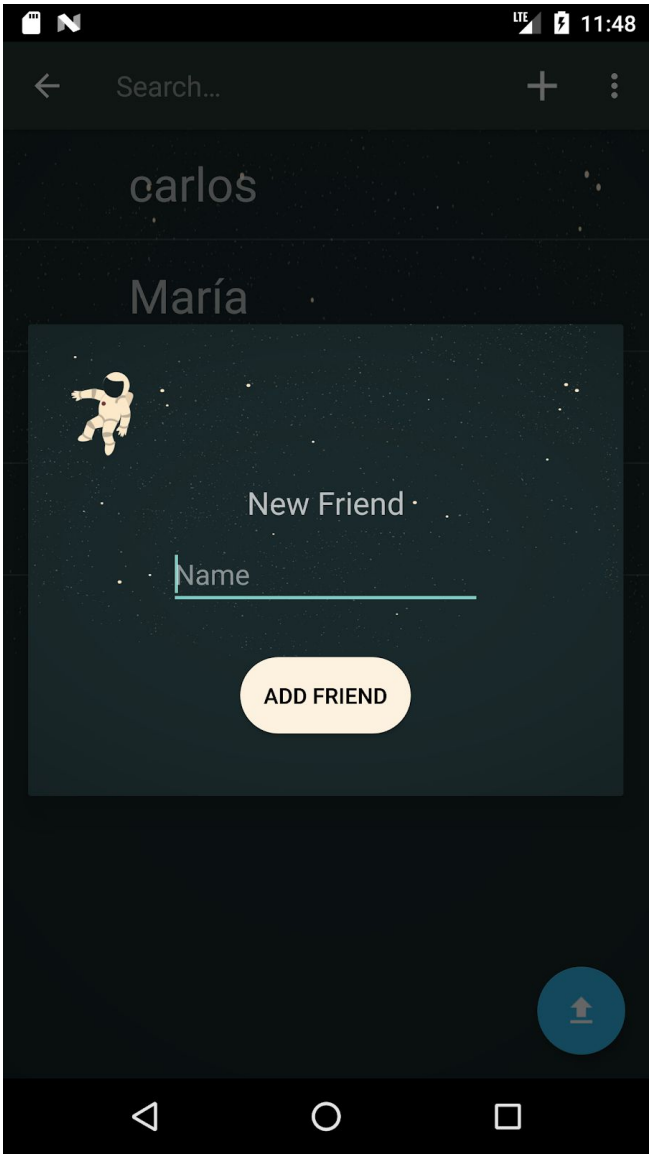

Figura 33: pantalla de Envío de solicitud de amistad.

Una vez enviada la solicitud, o si se es el usuario que recibe la solicitud, aparecerá una ventana emergente alertando al usuario que otro usuario le ha enviado una solicitud de amistad.

Pantalla de recepción de solicitud de amistad

Si se recibe una solicitud de amistad, se muestra en una pequeña ventana emergente dicha petición indicando el nombre del usuario que la ha realizado. Hay dos opciones, aceptar como amigo al usuario que ha realizado la petición, o denegar la solicitud. (Ver figura 34)

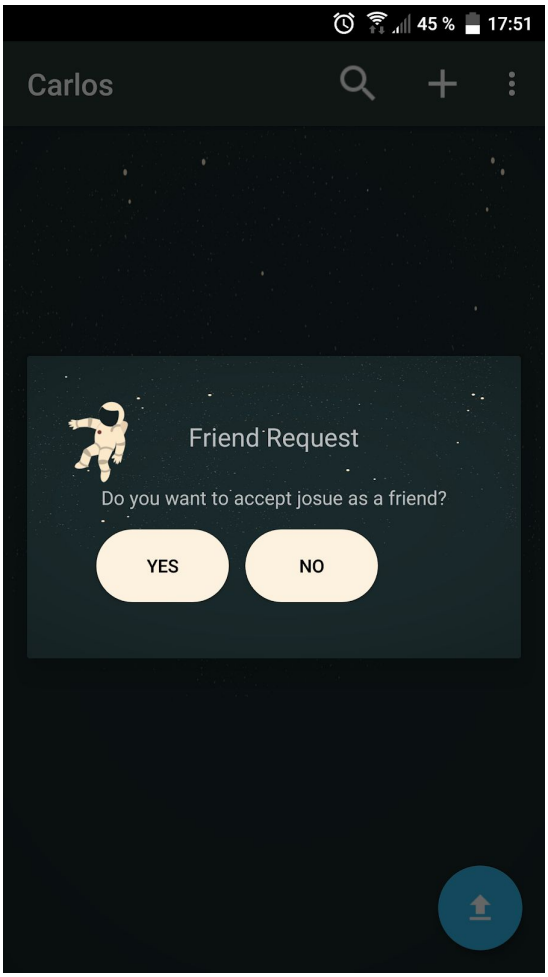

Figura 34: pantalla con la ventana emergente de una solicitud de amistad entrante

Al aceptar la solicitud recibida, el usuario que la ha enviado se añade a la lista de amigos para poder ver sus archivos elegidos para la compartición.

Pantalla de subida de archivos para compartición

Al pulsar en el botón para subir archivos a compartir, (ver figura 35), al usuario se le muestra el árbol de directorios del dispositivo. Desde este se puede navegar a través del móvil para encontrar el archivo que se desea subir para la compartición. Todos los Amigos que se tengan podrán ver los archivos subidos para la compartición, y podrán descargarlos.

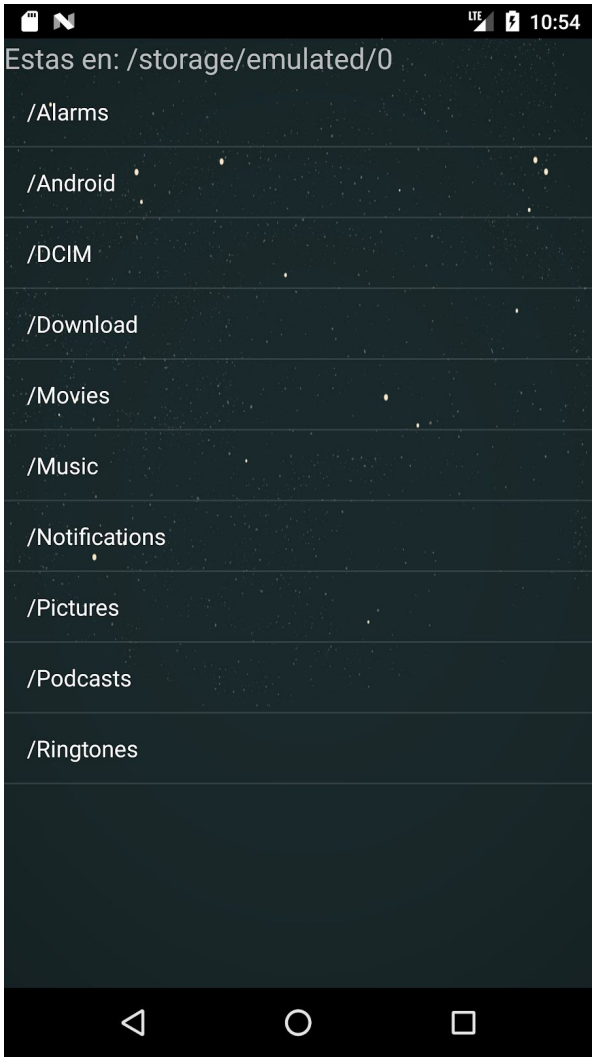

Figura 35: árbol de directorios del dispositivo para elegir el archivo a compartir.

B. Contribuciones de cada miembro

A continuación se listará cada una de las aportaciones individuales que ha realizado cada uno de los autores del proyecto.

Jesús Perucha Pérez

Al empezar todo este proyecto, lo primero que hice fue buscar en el repositorio de la UCM todas las memorias que obtuvieron matrícula de honor, para ello tuve que buscar la lista que saca la UCM con las notas de la gente que va al tribunal de matrícula de honor e ir comparando esos nombres con los de las memorias que hay disponibles. Además de esas memorias también recopile las memorias de nuestro departamento.

Todas estas memorias las puse a disposición de mis compañeros en una carpeta compartida para poder comparar de forma fácil en cualquier momento la forma de escritura, las secciones que tenían cada memoria o cualquier otra cosa que pudiera sernos útil para conformar la memoria. Gracias a este trabajo y comparando muchas de ellas junto con la lista de requisitos para los TFG que hay publicados en la web de la UCM forme el esqueleto de la memoria, creando toda la estructura de secciones que hemos completado y que forman la memoria.

También en todo el tiempo que ha durado el proyecto me he encargado de todas las decisiones de estructuración, inclusion o exclusion de secciones, decidiendo el contenido de ellas, orientando a mis compañeros sobre mi idea sobre lo que tenía que incluir cada sección y los subapartados que la conforman, de tal forma que toda la toma de decisiones sobre esta memoria ha pasado por mi.

Antes de empezar esta memoria mis compañeros habían generado escritos sobre la documentación que habían leído, los cuales me encargué de limpiarlos, adaptarlos e insertarlos en los puntos correspondientes de la memoria para poder usar ese trabajo inicial.

Con relación a la tarea estructurar de la memoria, me he encargado de revisar la memoria cada poco tiempo y ver todos los cambios que mis compañeros han hecho. He realizado correcciones sobre el conjunto global de la memoria, releyéndola entera para encontrar puntos redundantes y crear nuevas secciones de la memoria según avanzaba el desarrollo, también reestructurando los puntos nuevos que se iban escribiendo en la memoria con los puntos antiguos para evitar mezclar contenido de secciones y tenerlo todo actualizado y lo más paralelo posible al desarrollo, de esta forma también evitar colar erratas o conclusiones desfasadas por nuevas vías de desarrollo del proyecto.
También por ese motivo de encargado de la memoria, me encargaba de solicitar las reuniones con los tutores cuando nosotros podíamos, escribiendoles a ellos la situación en la que estaba el proyecto y enviandoles los avances que hacíamos o también escribiendo las dudas que tuviésemos o cualquier otra cosa que necesitáramos.

En la parte de la documentación, me centre en leerme algunos enlaces y fragmentos de libros con temática P2P que nos paso Simon además de paginas web con información sobre los puntos tratados en esta memoria, para tener unas referencias diversas sobre las que aprender y escribir la memoria. Gracias a los conceptos investigados he podido escribir sobre las Redes P2P, su topología y estructura, además también investigue aplicaciones que usasen el P2P como forma de intercambio de ficheros, como eMule o BitTorrent para contrastar sus particularidades en cuanto a la red P2P que forman, ayudando al enfoque de nuestro proyecto. También me encargue de investigar y escribir la parte de protocolos, en la que inicialmente buscamos una solución al problema de la NAT transversal, pero como no lo encontramos, Josué siguió mirando por otro camino y encontró PubNub + WebRTC, asi que tuve que mirar cómo funcionaba y hacer la sección donde explico la elección de esta tecnología sobre otras para poder realizar la conexión en la app.

En la parte del desarrollo del proyecto me encargué de hacer y escribir toda la parte relacionada con la ingeniería de software, creando la planificación y la especificación de requisitos de la App. Estas partes están hechas con la colaboración de mis compañeros pues nos reunimos para decidir cómo hacer ambas partes. Dentro de la parte de la planificación y

con la herramienta SmartSheet realice el diagrama de Gantt para reflejar la planificación de la aplicación de cara al desarrollo de la misma.

En la parte de la aplicación android tuve que aprender primero, pues nunca había trabajado con Android, para formarme recurrir a un curso gratuito de Udacity sobre android (este), impartido por profesionales de Google que ayudaron en la creación de Android y que usa el IDE de desarrollo Android Studio que es el que decidimos usar para crear la aplicación. Todo el aprendizaje de Android ha sido muy rápido y condensado, quedandome un mal sabor de boca pues fuera de los conceptos manejados en la App y explicados en la memoria, no he aprendido todo lo que quería aprender sobre Android al elegir este TFG ya que quería salir con un conocimiento adecuado para trabajar en más aplicaciones.

Ayude a Josué con la maquetación y presentación visual de la aplicación. Ambos decidimos darle a la aplicación una temática espacial utilizando fondos oscuros y resaltando las letras con colores claros para que sean legibles y fácilmente reconocibles. Para el resto de pantallas de actividades decidimos quitarle tanto contenido gráfico y seguir con la línea espacial, utilizando tipografía clara sobre fondo oscuro. Sobre la parte puramente funcional de la aplicación me encargue de crear el sistema de entrada a la aplicación, evitando el login que Josué había creado previamente donde había que registrarse dentro de la aplicación con un usuario y contraseña, y simplificandolo para solo pedirte un nombre al inicio de la aplicación y recuperar ese nombre con cada ejecución posterior que se haga. Josué se centró en el grueso de la aplicación creando todas las funciones y le ayude con lo que él me pedía, como por ejemplo revisando y ayudando a definir el código.

Para mi el problema de aprender Android nativo es que los conceptos son distintos a lo que se usan en Java, los layouts para generar la representación gráfica Android los compila como objetos que hay que manejar desde el código Java y es un concepto nuevo que he aprendido. La representación gráfica de la aplicación se hace con XML, lenguaje que no había usado nunca antes con lo que también he tenido que aprender lo básico para generar la interfaz gráfica de la aplicación.

Josué Pradas Sacristán

A lo largo del Trabajo de Fin de Grado ha habido varios trabajos a desarrollar en los cuales he estado presente, aportando ideas y solucionando problemas.

Primeramente, hablemos sobre el código. He aprendido a utilizar el IDE Android Studio 3.0 desarrollado por Google y JetBrains. En él he desarrollado la app, junto con mi compañero Jesús, que finalmente se hará entrega para ser revisada. Al principio fue tortuoso tener que aprender el modo de programación que Android tiene, ya que aunque sea en Java (en poco va a cambiarse el lenguaje de programación por Kotlin), se tiene que aprender a programar una app para un sistema distinto del convencional: un dispositivo móvil. En un móvil los recursos disponibles son muy inferiores comparado con los recursos de los que dispone un ordenador, en memoria, hilos de ejecución, carga sobre los mismos, renderizado gráfico. Por lo que aprender a programar para móvil es casi aprender a programar de nuevo, ya que debes ser muy eficiente y tener en cuenta muchas variables que normalmente con un programa de ordenador convencional no tienes en cuenta.

En cuanto al desarrollo de la app he participado en todo lo que finalmente se ha desplegado de código, documentandome, resolviendo los problemas que surgieron y escribiendo código para avanzar en el proyecto. En este apartado de código he participado en el desarrollo de Profile, ArchivesDatabase, ArchiveExplorer, DatabaseHelper, Friends, FriendsAdapter, MainActivity y Recursos. Todas estas clases desarrollan el núcleo de la aplicación, sobre todo Profile que es la actividad sobre la cual gira toda la lógica de conexión y compartición de archivos.

Hablando de Kotlin, este lenguaje desarrollado por Google y JetBrains, busca ser el nuevo estándar en desarrollo de aplicaciones android. Aunque es totalmente interoperable con Java, de manera que cualquier función o método escrito en un lenguaje puede ser convertido al otro, y viceversa, Google busca establecer Kotlin como el predeterminado y dejar de usar Java. Esta es otra de las cosas que tuvimos que tener en cuenta a la hora de programar la aplicación: ¿Nos interesa hacerla para que dure a lo sumo, 2-3 años hasta que

Google establezca Kotlin, o miramos al futuro y aprendemos a usar Kotlin para que la aplicación pueda ser desarrollada por futuros TFG's sin ningún problema? Finalmente nos decidimos por Java, ya que es un lenguaje que conocemos, aunque tuviéramos que adaptarnos al nuevo medio al que iba dirigido y sabiendo que es totalmente interoperable no vimos ningún problema por decidirnos por Java antes que aprender a usar Kotlin y cargarnos con más trabajo, a priori, innecesario.

En cuanto a la memoria, he participado en su relleno y en su corrección posterior. Apartados como Redes P2P, Código relevante, Tecnologías investigadas y más. Alguna de ellas las he escrito yo solo y otras se han realizado colaborativamente con el otro compañero de TFG. En cuanto a la corrección de la misma he participado en la limpieza de formato de la memoria, detección de casos de copia de internet, su identificación y eliminación o cambio.

Se ha seguido un formato genérico en cuanto al usado en la memoria, de manera que es bastante estándar en cuanto a formato, situacion de texto en el documento y párrafos. Cuando buscamos memorias modelo para inspirarnos y utilizar como referencia para la nuestra acudimos al banco de TFG's que cuelga la universidad en su web y de esta manera, añadiendo el toque personal de cada uno, pues de gustos no hay nada escrito, y tomando referencias en cuanto a TFG's pasados, los párrafos escritos, el interlineado, tipografía, etc, hemos desarrollado la memoria actual.

Para concluir, decir que he participado de manera activa en cada una de las etapas y necesidades que este TFG ha requerido o pensamos que serían necesarias desplegar. Siempre me he mostrado activo y he ayudado a resolver los problemas que iban surgiendo mirando a la meta de tener todo preparado y listo para terminar en la convocatoria de junio y así poder terminar la carrera.

Gracias a este TFG también he podido aprender de las redes, cómo se comunican los ordenadores, como se despliegan estas redes y sobre todo, especialmente, como funcionan las redes P2P, como fueron concebidas y por qué. He sacado en valor el desarrollo de una aplicación en Android que al principio de curso no veía capaz de hacer y entender cómo funciona el sistema operativo de Android.