UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Red P2P centralizada para el streaming de vídeo almacenado

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Cantos Agudo, Juan Carlos

Tutor: Oliver Gil, José Salvador

Curso 2019-2020

Resum

En l'actualitat, gran part de l'entreteniment a internet es basa en la visualització de vídeos ja siga en forma de pel·lícules, series, etc...

La majoria d'aquestes plataformes utilitzen una arquitectura Client/Servidor on el servidor s'encarrega d'emmagatzemar tots els arxius i repartir-los segons les peticions que envien els clients, carregant al servidor amb grans quantitats de treball.

Aquesta arquitectura és la més popular per que és fàcil d'implementar, però te diversos problemes, i el major d'ells és l'escalabilitat, necessitem canviar el servidor o destruir diferents servidors per a poder augmentar, però sería una infraestructura molt costosa, a més del punt d'error únic, si el servidor es desconnecta cap dels clients podria obtindre recursos d'ell i el sistema es quedaria inútil fins que el servidor tornara a connectar-se.

Una solució per aquest problema son les arquitectures P2P, amb capacitat per descentralitzar totalment la xarxa utilitzant el menys possible la figura del servidor, o en algún cas, no utilitzar-la.

En aquest projecte, s'estudiarà les diferents arquitectures P2P, es parlarà tant de les seues característiques com dels seus avantatges i desavantatges. També s'estudiarà el streaming de vídeo emmagatzemat, diferents tipus de streaming de vídeo.

Al llarg d'aquesta memòria s'exposaran tant les ferramentes utilitzades com les provades que poden ser utilitzades per a futurs projectes similars.

Tot açò per a completar l'objectiu principal d'aquest projecte, crear una xarxa P2P centralitzada per al streaming de vídeo emmagatzemat.

Per finalitzar, aquesta xarxa serà provada de diferents maneres per a validar el seu funcionament.

Paraules clau: Xarxa, P2P, Streaming, Par, Vídeo, VoD

Resumen

En la actualidad, gran parte del entretenimiento en internet se basa en la visualización de vídeos: ya sea en forma de películas, series, etc...

La mayoria de estas plataformas utilizan una arquitectura Cliente/Servidor donde el servidor se encarga de almacenar todos los archivos y repartirlos según las peticiones que envíen los clientes, cargando al servidor con grandes cantidades de trabajo.

Esta arquitectura es la más popular porque es fácil de implementar, pero tiene varios problemas y el mayor de ellos es la escalabilidad. Necesitariamos cambiar el servidor o distribuir diferentes servidores para poder aumentarla, pero sería una infraestructura muy costosa. Además del punto de fallo único, si el servidor se desconecta, ningún cliente podría obtener recursos del servidor y el sistema se quedaria inutilizado hasta que el servidor volviese a conectarse.

Una solución a este problema son las arquitecturas P2P, capaces de descentralizar totalmente la red utilizando lo menos posible la figura del servidor o, en algúnos casos, no utilizarla.

En este proyecto, se estudiarán las diferentes arquitecturas P2P, se hablará tanto de sus caracteristicas como de sus ventajas y desventajas. También se estudiarán el streaming de vídeo almacenado, diferentes tipos de streaming de vídeo. A lo largo de esta memoria, se expondrán tanto las herramientas utilizadas como las probadas que puedan ser útiles para proyectos futuros similares.

Todo esto para completar el objetivo principal de este proyecto: crear una red P2P centralizada para el streaming de vídeo almacenado. Para acabar, esta red será probada de diferentes maneras para validar su funcionamiento.

Palabras clave: Red, P2P, Streaming, Par, Vídeo, VoD

Abstract

Nowadays internet's most consumed type of entertainment is video, whether in form of movies, series, etc...

Most of these plataforms us a Client/Server architecture where the server is responsible for storing all the files and distributing them according to the requests sent by the clients, loading the server up with work.

This architecture is commonly used because it's easíly implemented, but it has several problems, and the biggest one of them is scalability; to increase it we would need to get a more powerful new server or to distribute various of them throught the network. Either way, it's a very expensive infrastructure. In addition, Server/Client has the single point of failure: if the server gets desconnected the clients wouldn't be able to download the resoruces in the server, so the network would be useless until the server is reconnected.

A solution to this problem is P2P architectures, capable of fully decentralizing the network by using the server figure as little as possible or, in some cases, not even using it.

In this project we will study the different P2P architectures, talking about its characteristicas as well as their advantages and disadvantages. Also we will be studying different types of video streaming and how it works. Throughout this report, both the used tools and the ones that can be useful or future similar projects will be exposed.

All this to complete the main objective of this project: to create a centralized P2P network dedicated to streaming stored video. To finish, the network we created will be tested in different ways to validate the implementation.

Key words: Network, P2P, Streaming, Peer, Video, VoD

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

El entretenimiento es una parte fundamental de nuestra vida y, con Internet, se ha hecho cada vez más facil acceder a entretenimiento y ocio, sobre todo en forma de películas, documentales o vídeo en general. Pongamos un ejemplo actual: Netflix es una gran plataforma donde hay una cantidad enorme de contenido que puede consumerse en cualquier momento. Esto se llama Vídeo bajo demanda o Video on Demand (VoD; pero no es algo nuevo: hace poco más de 10 años, YouTube ya existía y, hace 20, la antigua ONO también ofrecia VoD en su red.

Figura 1.1: YoutTube

Figura 1.2: Netflix

En aquel momento, ver un vídeo por internet suponía dos problemas: la calidad era muy mala y la velocidad de descarga, lenta. Con la tecnología de ahora, esos problemas han desaparecido y podemos ver vídeos a una calidad que, hace 10 años, no podríamos imaginar. Y es por eso que ahora tanta gente lo utiliza: aproximadamente 6 de cada 10 españoles ven sus programas, series y películas mediante streaming[9]. No obstante, sigue teniendo problemas.

Muchas de estas plataformas utilizan un sistema Cliente/Servidor1.3 que dispone de uno o varios servidores distribuidos, que cargan con el peso de una gran cantidad de vídeos y, además, tienen que aguantar la conexion de los clientes que quieran ver el vídeo. Esto es cada vez más problemático, pues los servidores se saturan -sobre todo con la distribucion de vídeo en alta definición o HD y con grandes cantidades de clientes-. Esto puede llegar a ocasíonar cuellos de botella en el servidor.

Para solucionar esto, se necesita una infraestructura robusta y cada vez más costosa. Sin embargo, hay varias soluciones para este problema, como la arquitectura P2P, que ayuda a solucionar tanto el problema tanto de escalabilidad como de eficiencia de costes pues, como su nombre (Peer-to-Peer o Cliente a Cliente) indica, esta arquitectura de red busca aprovechar diferentes atributos de los clientes como el ancho de banda o la capacidad de procesamiento sobrante para minimizar el trabajo del servidor o, en algúnos casos, eliminarlo de la red.

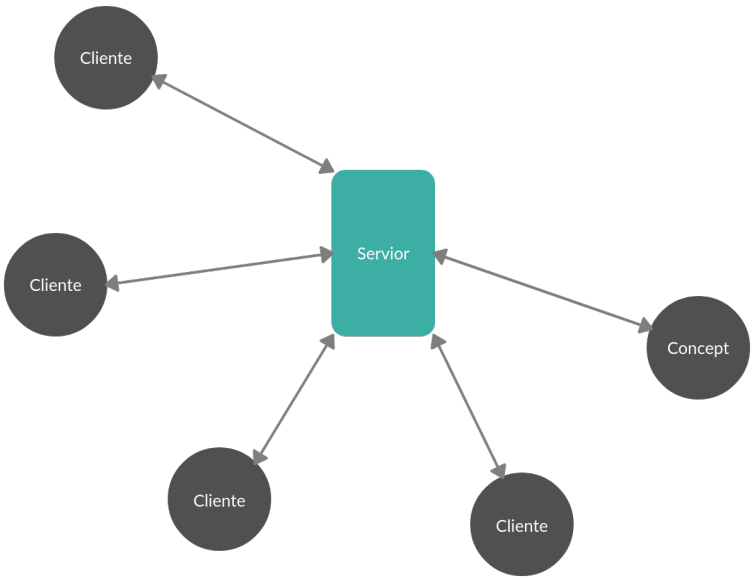

Figura 1.3: Arquitectura Cliente/Servidor

Hemos tenido tiempo para ver lo eficaces que son las redes P2P y que cada vez son más utilizadas[10]. Un ejemplo de esto es el protocolo BitTorrent1.4, como se explica en *The Big Book of BitTorrent*[17], BitTorrent es una serie de normas que crean las bases para un intercambio de archivos entre pares. Este protocolo consiste en dividir el archivo en pequeñas piezas de entre 64KB y 4MB junto con un identificador de cada pieza; todo esto se guardará en un archivo .torrent, que se subirá y consultará para saber de dónde descargar las piezas que conformarán el archivo completo. Un cliente de este protocolo es BitTorrent, pero este cliente se encarga de la descarga de archivos, que es diferente a streaming de vídeo pues en la descarga de archivos se utiliza 'el más raro antes'. Este modelo busca la parte más difícil de encontrar la primera y, a medida que descarga el archivo, la búsqueda es cada vez más facil. Sin embargo, en el streaming de vídeo tenemos que empezar por el principio. Un cliente interesante sería Pop Corn Time[1], que reproduce el vídeo a medida que lo descarga de diferentes clientes.

En este proyecto se hará una investigación de las diferentes redes P2P, sus principales diferencias y situaciones en las que serían viables, tambíen se listará una serie de herremianetas que puedan ser de utilidad para la creación de una red P2P y streaming de vídeo. Todo esto para finalizar con el diseño e implementación de una red P2P para streaming de vídeo almacenado.

[1] https://popcorntime.app/es/

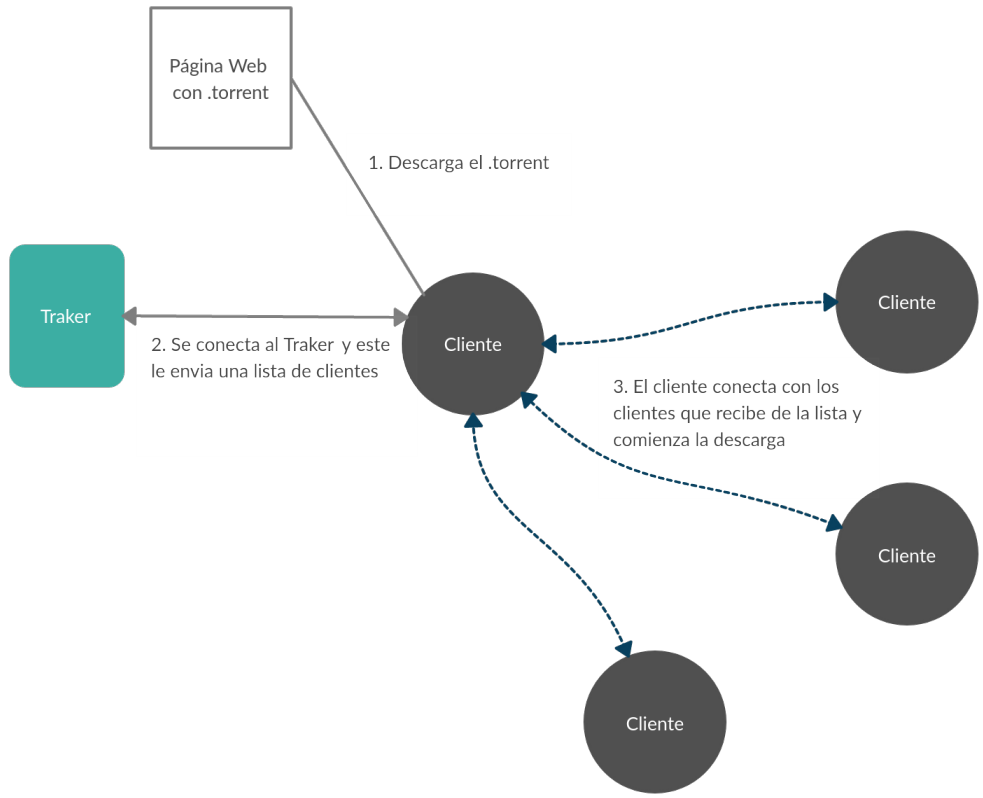

Figura 1.4: Protocolo BitTorrent

1.1 Motivación

Después de ser un usuario asiduo de redes P2P como BitTorrent[2] o Pop-Corn Time[3] y de estudiar las redes P2P durante la carrera, decidí que mi TFG sería una buena oportunidad para profundizar en este tema, embarcándome en la creación de una red P2P centralizada pero además añadiendo el elemento de streaming de vídeo almacenado, pues el poder ver el vídeo a medida que se descarga y poder avanzarlo, pausarlo o retrocederlo se me antoja un gran avance.

Además, dada la tecnología y avances hasta la época, me gustaría plasmar toda la información que pueda sobre diferentes herramientas que considere útiles, ya sean para una red P2P o para reproducción de vídeo en streaming.

1.2 Objetivos

Este proyecto tiene como objetivo principal crear una red P2P centralizada que sea capaz de emitir vídeo en streaming, y para lograr nuestro objetivo principal necesitaremos cumplir una serie de objetivos secundarios que serán:

- Diseño de una red P2P centralizada.

[2]Más información en:https://www.bittorrent.com/es/
[3]Más información en:https://popcorntime.app/es/

- Creación de un servidor y un cliente capaz de conectarse.
- Diseño y creación de un sistema de *signaling*.
- Encontrar un método para emitir vídeo almacenado.
- La emisión podra pausarse, reanudarse, adelantarse, retrasarse o descargarse.

Para la validación de estos objetivos se crearán varias pruebas que el sistema tendrá que pasar.

Además, como objetivo colateral, esta memoria recogerá información de diferentes herramientas útiles tanto para la creación de una red P2P como para streaming de vídeo.

1.3 Metodología

Para el previo estudio del proyecto y con motivos de reunir información antes de comenzar el diseño y desarrollo, se buscará en diferentes portales, como RiuNet, e-Archivo Trabajos Académicos, Recolecta o el motor de búsqueda de Google, trabajos de fin de grado relacionados con el tema que trata este, utilizando ciertos criterios, entre ellos que la fecha de emisión sea posterioral 2010, ya que las estructuras P2P son una tecnología que avanza rápidamente y la información o trabajos previos añ 2010 tienen altas probabilidades de estar obsoletos gracias a dichos avances tecnologicos. También usaremos las siguientes palabras clave: P2P, Peer-to-Peer, Vídeo bajo demanda, VoD, Streaming. Adicionalmente se buscará información de implementaciones similares a nuestra solución.

Una vez tengamos una base, comenzaremos a investigar herramientas y tecnologias que puedan ser útiles para lograr el objetivo final de este proyecto, crear una red P2P centralizada y streaming de vídeo; nos centraremos en herramientas que sean capaces de crear un servidor, un cliente y una conexión entre ellos, facilidad de uso y potencia.

Con la información útil de los documentos encontrados y las herramientas, empezaremos a crear pequeños diseños para probar las herramientas listadas y, con las herramientas del diseño más eficaz, se continuará hasta lograr los objetivos de este proyecto.

1.4 Estructura de la memoria

Esta memoria se estructurará de manera que comenzaremos hablando de lo que es una red P2P: sus diferentes tipos, sus características, ventajas y deventajas de cada tipo de red -esto último se plasmará en una tabla-. También se hablará sobre el streaming de vídeo almacenado, sobre el VoD y varios ejemplos de servicios de este tipo.

A continuación investigaremos sobre anteriores proyectos relacionados con este, ya sea por que trata sobre una red P2P centralizada o redes P2P en general.

También buscaremos trabajos relacionados con el streaming de vídeo almacenado.

La siguiente parte de esta memoria irá dirigida al desarrollo del proyecto, explicando el diseño, la estructura y soluciones a errores que fuimos encontrando en el progreso de este.

Para finalizar, se expondrán las pruebas que realizamos para la verificación del correcto funcionamiento y comprobaremos que se hayan logrado los objetivos que se marcaron al principio; se explicarán mejoras que se podrían implementar y conclusiones e ideas que se obtuvieron durante el desarrollo del proyecto.

CAPÍTULO 2
Estado del Arte

En este capítulo se realizará un estudio sobre las redes P2P y otros trabajos relacionados con las redes P2P o streaming de vídeo almacenado. Además, se hablará sobre proyectos importantes que implementen una red P2P centralizada y/o reproducción de vídeo en streaming.

2.1 Redes P2P

Para la investigación de las redes P2P se recurrió a diferentes libros, como *The World of Peer-to-Peer (P2P)*[16] y, sobre todo, a dos capítulos del libro *Peer-to-PeerComputing: Principles and Applications*[18], en concreto los capítulos *Architecture of Peer-to-Peer Systems* y *P2P Programming Tools*.

Como se explica en la introducción, las redes P2P se basan en la conexión directa entre Clientes o nodos, aprovechando sus capacidades, quitando trabajo al servidor y evitando problemas de escalabilidad.

Basandonos en que la descarga del fichero, o reproducción del vídeo en nuestro caso, sea entre clientes, existen diferentes tipos de organizar la red y por tanto, diferentes versiones de una red P2P, y cada una prioriza ciertas características.

En el texto que redactó Paul Baran[1], *On Distributed Comunications. Introduction to Distributed Comunications Network*[2] que iba destinado a recomendar a la fuerza aérea de Estados Unidos el uso de un sistema distribuido para poder responder más fácilmente a un ataque enemigos, también definió lo que es la tipología de una red y como están conectados los nodos.

A partir de un análisis topologico de las redes P2P podemos diferenciarlas en 3 tipos, que serán:

1. **Red P2P descentralizada**.
2. **Red P2P híbrida o semicentralizada**.
3. **Red P2P centralizada**.

[1]Paul Baran fué uno pionero en redes de computadoras, más informacion en: https://www.rand.org/about/history/baran.html

2.1.1. Redes P2P descentralizadas

En una red P2P descentralizada, como su nombre indica. no existe un nodo central, sino que un usuario ayuda a otro para enlazar las comunicaciones: los nodos hacen de cliente y servidor.

Son las más comunes gracias a su versatilidad, porque todas sus comunicaciones son usuario a usuario y no depende de un nodo central. Por los mismos motivos, este tipo de red goza de una alta disponibilidad pues, si un nodo se desconecta, otro nodo será capaz de ayudar a alguien que busca un archivo.

Pero no tener un gestionamiento central también trae problemas, como la dificultad de saber quién controla los recursos de la red o el inconveniente de administrar la red a medida que crece y se hace más compleja. Otro fallo es la seguridad pues, al no haber una seguridad central, cada nodo es responsable de la suya propia; esto nos puede llevar a descargar un archivo malicioso disfrazado con el nombre del archivo que buscabamos.

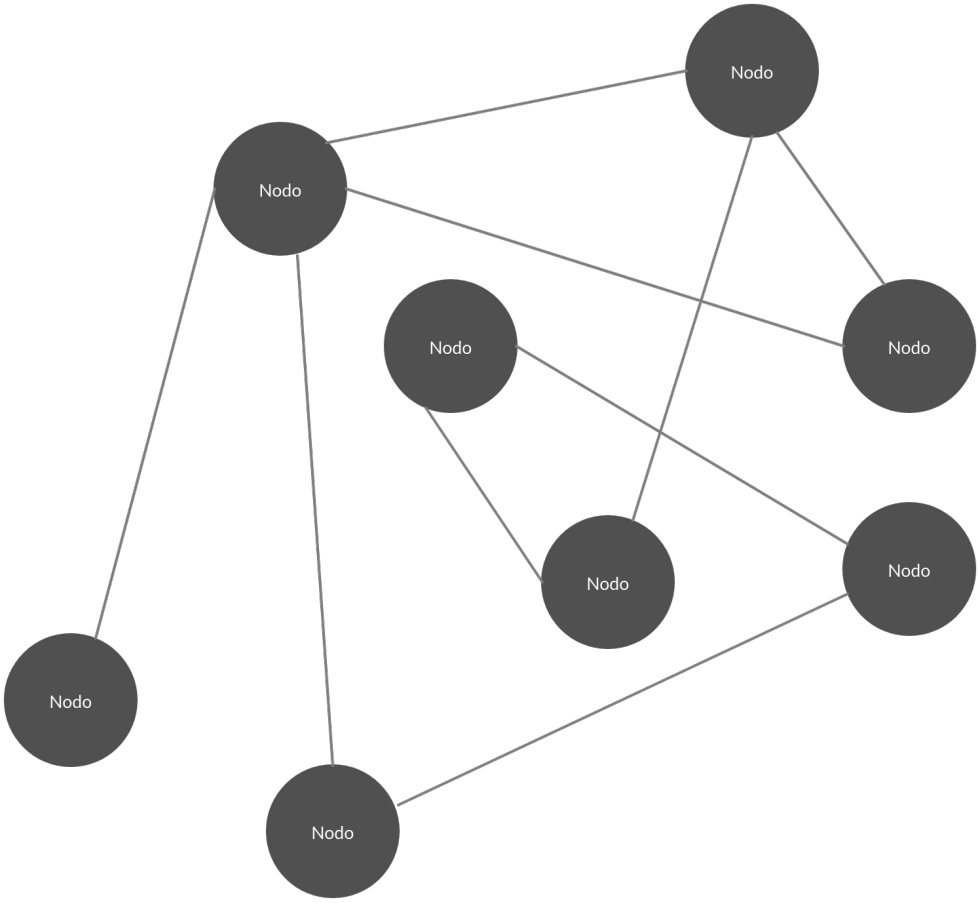

Figura 2.1: Mapa Topológico - Red P2P Descentralizada

Ejemplos de estas redes no son difíciles de encontrar: Gnutella y Ares son programas bastante conocidos que utilizan un sitema de red descentralizado.

Al no tener ningún nodo que se encargue expresamente de enlazar los nodos o de saber donde están los ficheros que se buscan, este sistema empezó utilizando el *flooding*. Este mecanismo trata de enviar la búsqueda de un nodo a sus vecinos si este no tiene el archivo que se esta buscando, de esta manera se la red se 'inunda' con la búsqueda hasta que encuentra un nodo que contiene el archivo o sobrepasa el número de saltos especificado en la búsqueda.

Este método es sencillo de implementar y es viable en redes pequeñas pero, cuanto más grande sea la red, más trafico genera y más grave es la inundación. Por suerte, actualmente contamos con mecanismos estructurales que facilitan la búsqueda de un archivo en una red P2P completamente descentralizada. Además del metodo de búsqueda, la estructura que siguen también es un punto determinante en la eficiencia de estas redes y, dependiendo de esto, podemos distinguir dos tipos de redes P2P descentralizadas:

- Estructuradas.
- No estructuradas.

Aqui se muestra una tabla resumiendo sus ventajas y desventajas:

Ventajas	**Desventajas**
Alta disponibilidad y versatilidad, un nodo puede tomar el puesto de otro cuando uno se desconecta	Dificultad a la hora de administrar la red
Eficaz en redes pequeñas	Se reduce su eficiencia a medida que la red crece
Fácil de implementar	Baja calidad de seguridad
	Altos tiempos de búsqueda

Tabla 2.1: Ventajas y desventajas de una red P2P descentralizada

Estructuradas

En este tipo de redes la sobrecapa u 'overlay' se organiza de forma específica con la finalidad de que el protocolo que utiliza sea capaz de encontrar el archivo buscado de la forma más eficiente posible.

El protocolo más utilizado en las redes P2P estructuradas son las tablas hash distribuidas o DHT. Como su nombre indica, se trata de una tabla hash que parte las 'claves' y las distribuye entre un grupo de nodos.

Para cada fichero que se suba a la red, se calcula una hash para su clave y esta es enviada a cada nodo que participe en ella. Este mensaje es tomado y enviado por los nodos hasta que llega al nodo que creó la clave. Es aquí donde se guarda el par('clave','valor').

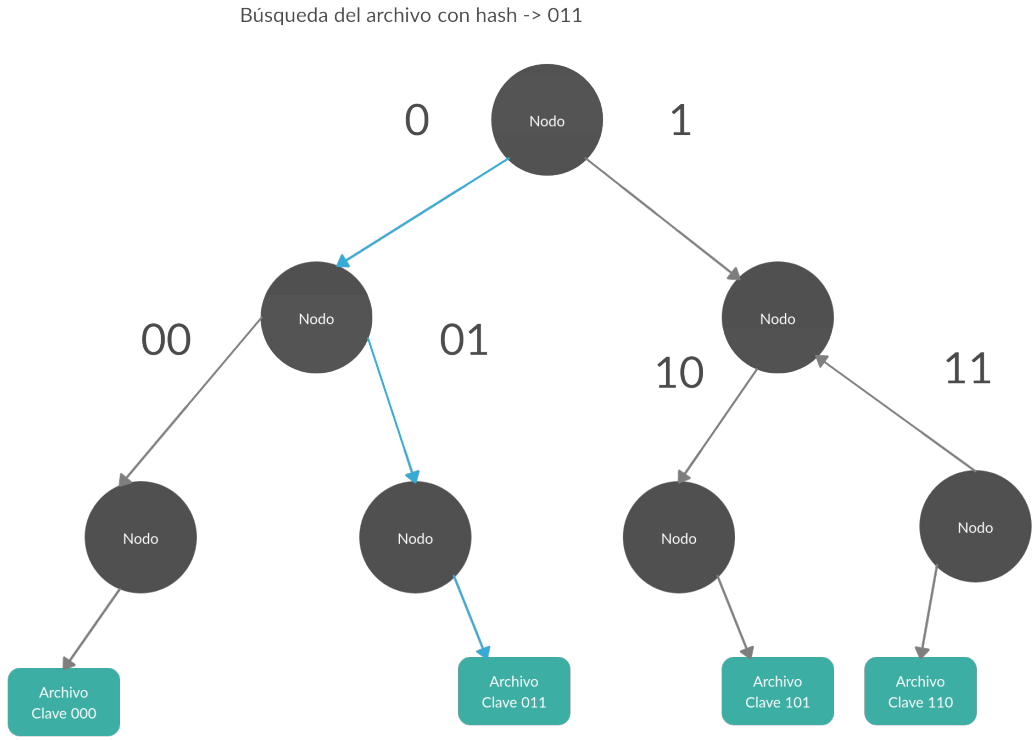

Figura 2.2: Protocolo DHT - Ejemplo

Ejemplos de implementaciones de DHT como estructura para redes P2P descentralizadas son Chord, Pastry y Tapestry.

Ventajas	**Desventajas**
búsqueda rápida en una arquitectura descentralizada	Pierde versatilidad, si un nodo se desconecta se pierden sus partes de hash y se rompe el camino
Garantia de que el archivo se encontrará	Altos costes a la hora de dar un archivo a conocer
	Complejo de implementar

Tabla 2.2: Ventajas y desventajas de una red P2P descentralizada esctructurada

No estructuradas

Este tipo de redes son fáciles de construir, pues no se organizan de ninguna manera en específico y los enlaces entre nodos se crean aleatoriamente. También, como todos los nodos cumplen las mismas funciones, es una red muy robusta, más especificamente con problemas de desconexión de uno o varios nodos.

Pero esta falta de estructuración y organización también trae problemas, sobre todo cuando un nodo busca un archivo: la petición inunda la red, pues necesita encontrar todos los nodos posibles que tengan ese fichero. Este método de inundación de la red cada vez que queremos buscar algo crea demasíado tráfico de datos y gasta más recursos de procesamiento y memoria(pidiendo a todos los nodos que procesen la petición de búsqueda). Además no garantiza que encuetres el fichero que buscas después de haber inundado la red.

Gnutella, Gossip y Kazaa son ejemplos de redes P2P descentralizadas no estructuradas.

Ventajas	**Desventajas**
Fácil implementación	Recurrente inundación de trafico o flooding
Eficiente en redes pequeñas	Pierde eficiencia cuando más grande es la red
	Altos tiempos de búsqueda

Tabla 2.3: Ventajas y desventajas de una red P2P descentralizada no estructurada

2.1.2. Redes P2P híbridads o semicentralizadas

La principal característica de las redes P2P híbridas es la presencia de uno o varios servidores que se encargan principalmente de la administración de las solicitudes entre los nodos que se han conectado a él. El set de funciones de un servidor en esta red es simple y permite que un nodo se convierta en servidor si la red lo requiere, transformándolo así en un súper nodo.

Estos súper nodos se encargan del enrutamiento y conexión de un grupo de nodos: es decir, es el centro de un grupo de nodos y normalmente está conectado a otro super nodo de la misma red.

En la siguiente figura se ve claramente la topología de esta red:

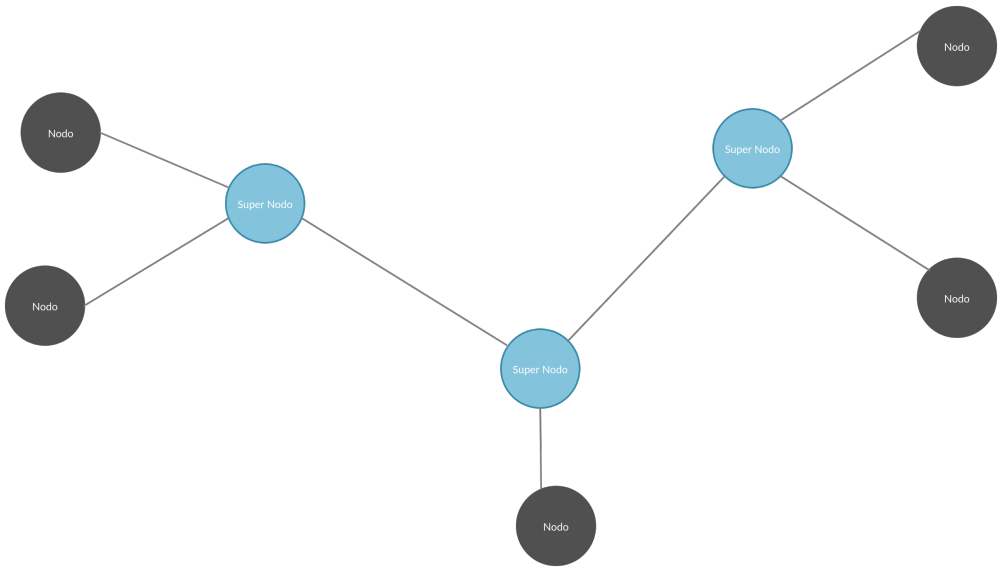

Figura 2.3: Mapa Topológico - Red P2P Híbrida

De todas formas, un nodo se convertirá en súper nodo si la red lo requiere y siempre cumpliendo los criterios de distribución, que son:

1. **Dispersión**: Los supernodos deben estar distribuidos homogeneamente, no se deben formar nucleos de supernodos.

2. **Acceso**: Los nodos deben tener poca latencia a uno o dos supernodos, esto se puede medir mediante saltos.

3. **Proporción**: Debe haber siempre un número de supernodos para el correcto funcionamiento y requerimientos que el sistema necesite para funcionar.

4. **Carga**: Cada supernodo debe servir a un número concreto de nodos dependiendo de sus capacidades, si no esto afectaria a la eficiencia del sistema.

5. **Heterogeneidad**: Cuanto más crezca la red, más supernodos habrán y más difícil será administrar la red, por eso se pide unos minimos requisitos, estos requisitos pueden ser el tiempo de vida del nodo, su capacidad de procesamiento, almacenamiento...

6. **Adaptabilidad**: En la búsqueda de un supernodo es crucial que el sistema pueda aguantar las peticiones fallidas o declinadas y responda rápidamente, sobretodo por la naturaleza dinámica de la red P2P.

7. **Torerante a fallos**: Cuando un supernodo sale de la red o por algún error se cae, otro nodo tiene que tomar su posicion rápidamente.

8. **Securidad**: Es muy importante que el nodo que se vaya a convertir en supernodo sea fiable, puede darse el caso de que se promocione un nodo a supernodo que de información erronea y perturbe la red.

Este tipo de red P2P tienen actualmente un rendimiento más alto que las redes P2P descentralizadas; cosas como buscar un archivo son mucho más eficaces en una red mínimamente centralizada, y tiene la misma robustez pues durante la descarga del archivo, si el súper nodo se desconecta, esto no afecta a la comunicación entre dos nodos ya conectados.

Ejemplos de este tipo de red P2P puede ser BitTorrent, o eDonkey.

Ventajas	**Desventajas**
Eficacia en la conexión de nodos	difícil de administrar
Robusta, la caida del servidor no afecta a la conexión entre clientes	Baja seguridad
Puede llegar a tener una alta escalabilidad, depende de los super nodos	

Tabla 2.4: Ventajas y desventajas de una red P2P descentralizada no estructurada

2.1.3. Redes P2P centralizadas

En esta arquitectura de red P2P todas las transacciones se hacen a través de un único servidor al cual todos los nodos se conectan. Este servidor será el punto de encuentro entre dos nodos y su funcionamiento es similar a la arquitectura Cliente/Servidor, pero, en lugar de descargar el fichero diractamente del servidor, este buscará en su base de datos si algún nodo que esté conectado tiene ese archivo. En el caso de que algún nodo de la lista lo tenga, el servidor pasará la dirección/identificación de este al cliente que mandó la petición y se conectará con el nodo que tiene el fichero para descargarlo directamente. Si, por el contrario, ningún nodo tiene el fichero que busca el cliente, el servidor buscará en su almacenamiento y, si lo tiene, se lo pasará al cliente y además guardará en su base de datos que ahora ese nodo tiene el archivo.

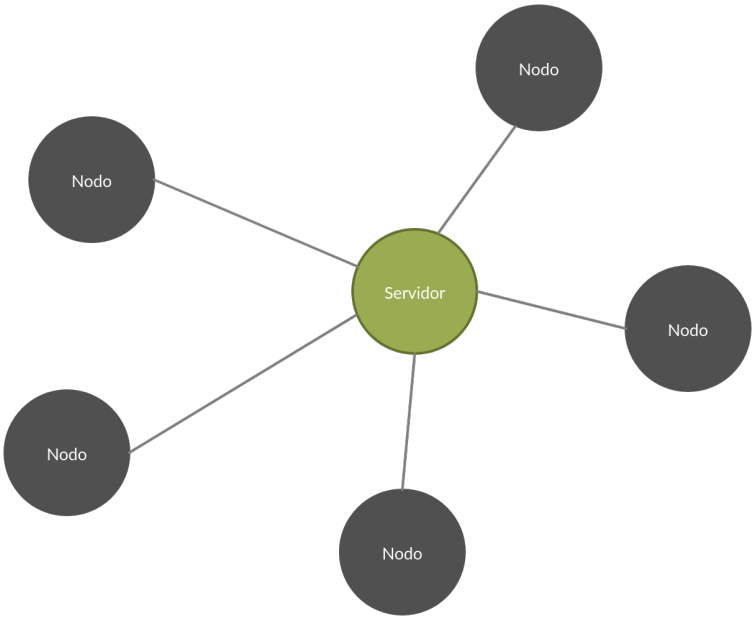

Figura 2.4: Mapa Topológico - Red P2P Centralizada

Este tipo de redes se caracterizan porque tienen un único servidor que se encarga de conectar nodos y además almacena contenido. Esto, en términos de enrutamiento, es muy eficiente y es capaz de crear conexiones rápidamente sin malgastar tiempo buscando el archivo deseado y aumenta la disponibilidad de los archivos, pues, si alguien pide un archivo que ningún nodo conectado tiene, el servidor lo puede pasar fácilmente. Pero un servidor único del cual dependen todas las comunicaciones entre nodos trae varios problemas, uno de ellos siendo la existencia de un punto de fallo único: si el servidor se desconecta toda la red desaparece, además del mantenimiento es problematico, alto consumo de ancho de banda y la escalabilidad más reducida comparada a los otros tipos de redes P2P.

Ventajas	Desventajas
Fácil de administrar	Menos escalable que las demas arquitecturas P2P
Alta seguridad	Solo soporta lo que el servidor pueda soportar
Fácil de implementar	Punto de fallo único

Tabla 2.5: Ventajas y desventajas de una red P2P descentralizada no estructurada

2.2 Streaming de Vídeo

Una vez finalizada nuestra investigación sobre las redes P2P y sus diferentes características, es hora de profundizar en la otra parte importante de este proyecto: el streaming de vídeo.

El streaming de vídeo es un método de transmisión de un archivo de medios en un flujo continuo de datos que son procesados por el cliente antes de que se haya enviado el archivo completo. Esto implica varias cosas: un streaming de un vídeo puede ser pausado, reanudado, retrasado y adelantado, y el vídeo puede visualizarse a medida que es descargado. En ocasiones estos streamings de vídeo ofrecen diferentes calidades de imagen y ajustarse al ancho de banda que use el cliente.

Todo esto es gracias a la presencia de un *buffer* (o búfer, en español), que es un espacio de memoria dedicado al almacenamiento temporal de información en el que se guardan datos durante el tiempo de espera antes de ser procesados. En el caso del streaming de vídeo, el búfer funciona como un colchón: el vídeo se reproduce mientras se descarga y el búfer contiene los siguientes x segundos del vídeo de forma que, si la conexion se corta, se puede reconectar mientras el búfer reproduce los segundos de vídeo que tenía. Un búfer suele contener alrededor 64Kb de 4Mb de datos.

2.2.1. Tipos de streaming de vídeo

A continuación, se definirán los diferentes tipos de streaming de vídeo y sus correspondientes protocolos:

Streaming de vídeo en directo

El streaming de vídeo en directo, como su nombre indica, se basa en la codificación de audio y vídeo en el acto; estos retransmiten instatáneamente a Internet y el cliente recibe la retransminisón en tiempo real. Ejemplos de este tipo de streaming son las videoconferencias o directos en plataformas como Twitch y YouTube.

Para retransmitir este tipo de streaming no pueden utilizarse protocolos basados en TCP pues este es un protocolo orientado a la conexión y, en caso de que se produzca un error o se pierda un dato, este se vuelve a transmitir; es decir, si se utilizase este protocolo en un vídeo, a la minima pérdida de un paquete el ví-

deo se congelaría. Por tento, se utilizan protocolos basados en UDP. Uno de ellos es RTP, el cual proporciona servicio de entrega en la red desde el origen hasta el destino.

Streaming de vídeo bajo demanda

En el streaming bajo demanda, el contenido multimedia se aloja normalmente en un servidor y puede ser visualizado en cualquier instante. En este caso es el cliente quien controla la transmisión y recepcion del contenido.

Al contrario que el streaming de vídeo en directo, este no se transmite a tiempo real; es el cliente el que controla la recepción. Es posible usar protocolos basados en TCP, como HTTP y FTP. Estos protocolos aseguran que los paquetes lleguen a su destino y en secuencia. De esta forma, si se pierden paquetes por el camino, estos vuelven a retransmitirse pero no hay problemas en la recepción porque el cliente se encarga de todo.

2.3 Herramientas

El proyecto se llevará a cabo con JavaScript[2] y NodeJS[3] principalmente, así que en este capítulo se hará una lista junto con un breve resumen de las herramientas de estos lenguages que sirvan para crear redes P2P y para streaming de vídeo.

Empezaremos con heramientas dirigidas al P2P y continuaremos con streaming de vídeo.

2.3.1. Herramientas P2P

Principalemente, necesitaremos librerías capaces de crear sockets, conectarlos a un servidor y poder enviar mensajes entre ellos además de poder conectarse directamente a un cliente.

WebRTC

WebRTC[4] es una tecnología capaz de agregar comunicación en tiempo real a las aplicaciones donde se utilicen. Disponible en todos los navegadores modernos, lo está como API JavaScript regulares en todos los principales navegadores y además es de código abierto. Compatible con Apple, Google, Microsoft y Mozilla.

Figura 2.5: WebRTC

[2] https://developer.mozilla.org/es/docs/Web/JavaScript
[3] https://nodejs.org/es/
[4] https://webrtc.org/

Su funcionamiento es sencillo: accede a los medios de los dispositivos, abre conexiones entre pares y descubre pares para iniciar la transmisión. Un punto a favor de esta herramienta es la cantidad de tutoriales y ejemplos que Google nos ofrece; gracias a esto, lo que en principio es una herramienta complicada y poco intuitiva se convierte en una herramienta mucho más accesible. Además, hay varios proyectos que se pueden utilizar como base.

Esta herramienta es increiblemente útil pues junta todo: conexión P2P y envio de datos como vídeos o audio, de modo que es posible hacer streaming de vídeo en vivo y almacenado.

A menos que se quiera crear una red usando WebRTC de forma local, se necesitará un servidor. WebRTC no puede crear conexiones sin un servidor en medio[1], esto es lo que se llama 'signaling service', y en este caso se encarga de intercambiar información sobre la conexión y sobre los datos multimedia.

Estos servidores son servidores STUN/TURN:

1. **Servidores STUN**: Para poder conectar nodos entre si necesitamos ponerles algúna clase de identificación o obtener su dirección ip pública y el puerto, se ve claramente en la figura 2.6, un servidor STUN o (Sesión Traversal de User Datagram Protocol [UDP] a través de Network Address Translators [NAT]) se ocupa de eso, él cliente le envia una petición y el responde con la ip pública, una vez que tengamos nuestra dirección ip pública ya podemos enviarla a quién la necesite para comenzar una una llamada o streaming de vídeo en vivo.

2. **Servidores TURN**: Este servidor se encarga de resolver posibles incompatibilidades y permitir que cualquier dispositivo pueda unirse a las llamadas desde el navegador.

Normalmente un servidor STUN es suficiente para poder activar la conexión.

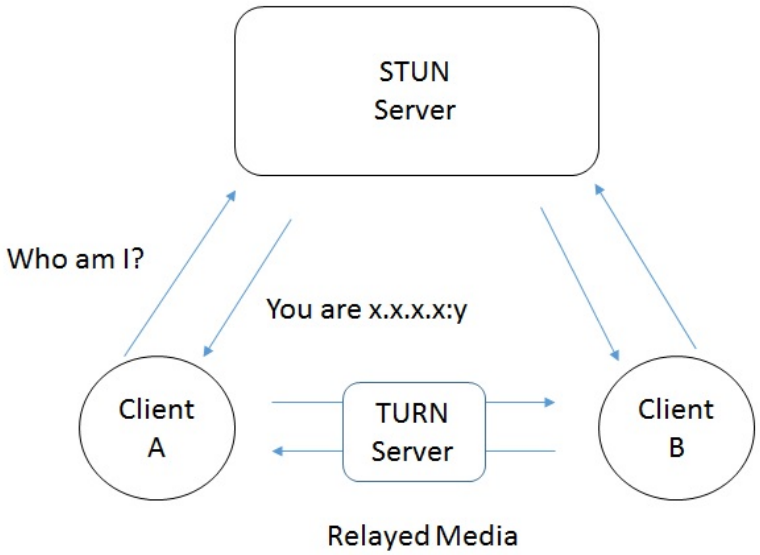

Figura 2.6: Funcionamiento de un servidor Stun y Turn
[12]

Para transimitir información multimedia se crea una Oferta por parte del cliente que servirá la información, y una Respuesta por parte del que consumirá, que contendrá el protocolo de descripción de sesión o (SDP), una pequeña muestra del flujo de unan Oferta y una Respuesta sería esta:

1. Cliente A, iniciara la conexión con una Oferta.
2. El servidor encargado de *signaling* enviará la Oferta al Cilente B.
3. Este creará una Respuesta en base a la Oferta recibida,
4. La Respuesta será devuelta al Cliente A.

En el caso del intercambio de datos sobre la conexión, esto se conoce como ICE candidates y define los métodos en los que el cliente está dispuesto a comunicarse (directamente o a través de un servidor TURN). Normalmente cada Cliente propone sus mejores opciones al principio; suele ser UDP ya que es el más rápido y una pérdida de paquete UDP no supone gran cambio en la reproducción del vídeo.

En este diagrama se muestra cómo funciona este intercambio de mensajes más claramente:

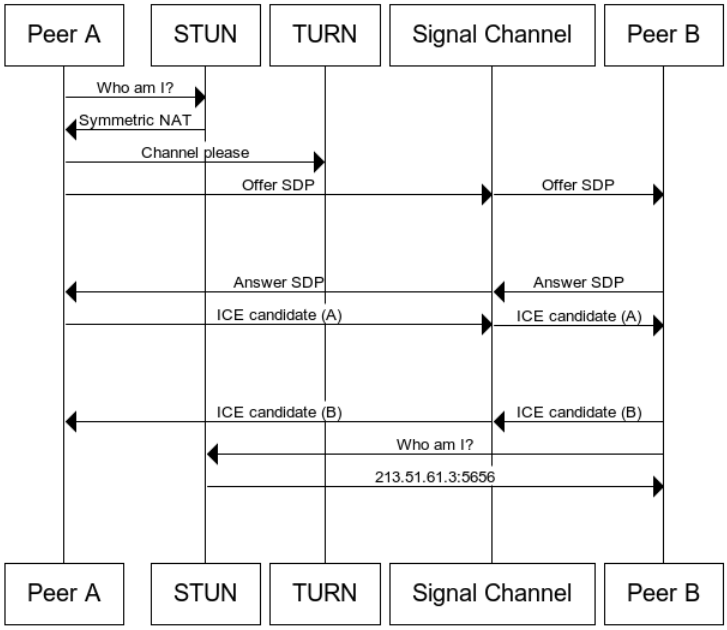

Figura 2.7: Diagrama de conectividad en WebRTC [12]

Socket.io

Socket.io es una librería capaz de crear un cliente en el navegador y un servidor. Su funcionamiento es simple aunque quizás resulte poco intuitivo si no se está acostumbrado a utilizar sockets. La afirmación de que es capaz de crear un cliente en el navegador se refiere a la posibilidad de importar la librería pegando el CDN en el mismo HTML y, a partir de ahí, abrir un script y utilizarla.

Figura 2.8: Socket.io

Crear un cliente desde el navegador es un gran punto a favor. De esta forma, el diseño de la red es mucho más compacto y aumenta la capacidad de interacutar con el servidor desde el cliente, que se conectará automaticamente al abrir el HTML. Con socket.io es muy fácil crear una arquitectura Cliente/Servidor y es útil también para redes P2P centralizadas pero, a parte de esto, socket.io ofrece una versión P2P capaz de enviar mensajes a otros clientes con el servidor como apoyo. En su github hay diferentes ejemplos, como un chat privado entre pares o un streaming de vídeo en vivo. Sin embargo, esta versión es vieja y no tiene apoyo desde hace 4 años.

PeerJS

Figura 2.9: PeerJS

Esta librería facilita el uso de WebRTC, es capaz de crear conexiones P2P configurables y tiene una API fácil de utilizar. Con solo la identificación de un cliente, esta librería puede crar una conexión y mandar datos como ví-

deo o audio a otro cliente. Para encargarse de las conexiones, esta librería necesita un servidor que lleve a cabo el intercambio de información entre pares, y es por eso que hay otra versión para programar y configurar un servidor propio, PeerServer. Si, por el contrario, solo se necesita un servidor básico para conectar ciertos nodos, pueden utilizarse servidores predefinidos que ofrece PeerJS.

PeerJS tiene una gran cantidad de opciones configurables, como utilizar SSL o poner un servidor detras de un proxy. Además, puede configurarse cómo se generan las identificaciones de los clientes. Esta librería, como socket.io, es capaz de crear un cliente en el mismo navegador.

En general es una librería muy útil y fácil de utilizar, con gran variedad de opciones a la hora de configurar y servidores que pueden usarse si no se pretende hacer uno propio. Sigue recibiendo soporte y utiliza WebRTC, lo que hace que sea capaz de crear una videollamada fácilmente entre dos clientes. El único problema es que es incapaz de enviar datos de Servidor a Cliente.

Sería perfecto para una red P2P híbrida pero no para una red P2P centralizada, ya que el servidor no puede enviar datos a los clientes.

Simple-Peer

Simple Peer es parecido a PeerJS: utiliza WebRTC, funciona en NodeJS y en el navegador, soporta streams de audio, vídeo y canales de datos binarios y, además, tiene opciones avanzadas como utilizar *Trickle ICE candidates*, una manera más rápida de montar el streaming de vídeo.

Básciamente, no espera a que se procesen todos los candidatos ICE. Pero esta librería tiene un fallo y es que no posee implementación para un servidor y debe montarse uno mismo el sistema de *signaling* para intercambiar información sobre la conexión; es una simplificación de WebRTC.

2.3.2. Herramientas streaming de vídeo

Ahora investigaremos sobre las herramientas que nos permitirán enviar un flujo de datos y recogerlo de un cliente a otro.

hls.js

Figura 2.10: HLS.js

Esta librería permite crear un streaming de vídeo. Implementa un cliente de streaming en vivo por HTTP, no necesita ningún reproductor externo pues funciona en el mismo elemento HTML <video>. La mayor ventaja de esta librería es su amplia comunidad: muchos servicios que ofrecen vídeo en streaming utilizan esta librería, como por ejemplo Twitter, Fox Sports, Canal+, etc. La gran cantidad de usuarios de esta librería es comprensible, pues es rápida, potente y extrema-

damente fácil de utilizar. Utiliza el protocolo HLS, que utiliza archivos .m8u3 con los que crea listas de segmentos que debe enviar.

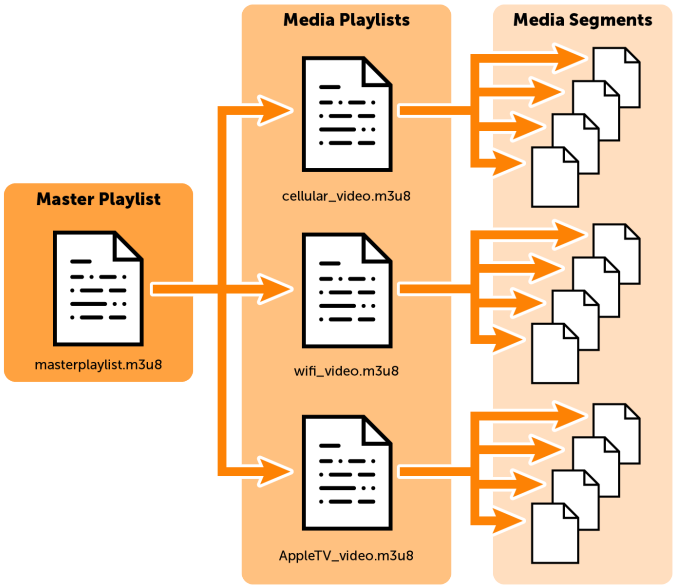

Figura 2.11: Diagrama de funcionamiento del protocolo HLS

Es de código abierto y puede encontrarse en su github.

VideoJS-HTTP-Streaming(VHS)

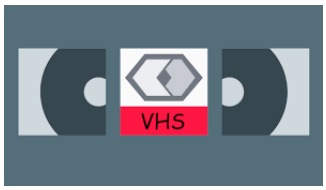

Figura 2.12: VHS

VideoJS es un librería que sirve para reproducir vídeo en un navegador, pero existe una versión de esta librería enfocada a reproducir un vídeo retransmitido via HTTP Live Streaming o HLS. También cuenta con varias opciones de configuración que hacen posible desde modificar el ancho de banda a comenzar la reproducción del vídeo con el bitrate más bajo. VideoJS también cuenta con una lista de plugins que añaden diferentes funcionalidades al reproductor creado por la librería, como por ejemplo el plugin 'videojs-contrib-quality-levels', que selecciona automaticamente la mejor calidad para reproducir el vídeo o permite hacerlo manualmente mediante el uso de eventos en el código.

Es una librería muy completa y, aunque tenga problemas con la compatibilidad, estos se resuelven fácilmente gracias a la gran variedad de extensiones que añaden funcionalidad o arreglan problemas.

Módulo http

Este es un módulo de NodeJS que ya viene integrado y es extremadamente útil, pues es capaz de transferir datos sobre HTTP. Esto quiere decir que es capaz de montar un servidor que escuche en un puerto elegido y responder a peticiones de clientes. Además, cuenta con cantidad de opciones de configuracion de respuesta, por ejemplo puede añadir una cabecera a la respuesta, guardar la petición del cliente como objeto, dividir la petición, etc.

Pero, en este proyecto, este módulo nos interesa para el streaming de vídeo, que es sorprendentemente fácil e increíblemente efectivo. Su funcionamiento para enviar vídeo por streaming se basa en dividir el vídeo en varias partes y enviarlos al cliente. Esto permite que el cliente pause el vídeo, lo adelante o retrase, pidiendo las partes de los vídeos que necesita en ese momento.

En la parte de desarrollo del proyecto se entrará más a fondo en el funcionamiento de este módulo con respecto al streaming de vídeo.

Módulo FileSystem(fs)

Este es otro módulo integrado que nos ofrece NodeJS: es capaz de leer, crear, escribir, borrar y renombrar archivos del sistema. Es un módulo básico pero potente y de grandísima utilidad. Normalmente se suele utilizar en la parte del servidor, y una de sus utilidades es leer un archivo HTML y enviarlo a un cliente en la respuesta; el cliente utiliza los datos que ha recibido para abrir el HTML que le ha servido el servidor. Este método de uso es muy comun en arquitecturas Cliente/Servidor.

Este módulo es también muy útil para leer el nombre de los archivos en una carpeta: es, de hecho, la manera en la que se integrará en nuestro proyecto.

Para más información de este módulo puedes consultar su API o diferentes ejemplos en esta página.

2.4 Documentos relacionados

Después de esta investigación sobre el tema a tratar en este proyecto, nos dirigimos a ver otros trabajos que tengan relación con el nuestro con la finalidad de crear una base de donde obtener ideas de optimización para nuestro proyecto. Los documentos encontrados en RiuNet que coinciden con los criterios previamente descritos son los siguientes:

- El primero es Propiedad Intelectual y Redes P2P[11], sus palabras clave son P2P, Propiedad intelectual, Ley y Protocolo. Este documento presenta información acerca de la legislación sobre porpiedad intelectual en lo que a redes P2P se refiere. Este trabajo se ha utilizado principalmente para
- Otro trabajo que cumplia los requisitos es Implementación de una solución de streaming bajo demanda usando DASH[5], sus palabras claves son DASH, MPEG, Streaming, Internet y Streaming bajo HTTP. Este trabajo se estudia el streaming adaptativo de vídeo y el estándar MPEG-DASH. Este proyecto se utilizó como ejemplo y fuente de información del protocolo DASH y el streaming de vídeo almacenado.
- El situiente proyecto que recogemos es Desarrollo de un sitema de live video streaming utilizando raspberry pi y el protocolo DASH[4], sus palabras clave son DASH, Streaming , Codificación de vídeo, Programación Web, Video encoding, Web programming. Este tfg crea un sistema de captura y transimision de vídeo de forma selectiva utilizando el protocolo DASH. Este documento se ha utilizado para el desarrollo del método de emisión de vídeo en nuestro proyecto.

A partir de aquí se muestra los documentos que no forman parte de los TFGs de la ETSINF pero se han utilizado como fuente de información para este proyecto:

- El primero en esta lista es P2P Sharing: Aplicación android P2P para compartición de archivos[8], sus palabras clave son P2P y WebRTC. Este proyecto estudia las redes P2P y crea una aplicación android donde puedes agregar a amigos, subir archivos y compartir esos archivos con tus amigos. Utilizamos este proyecto como fuente de información sobre la implementación de una red P2P.

- Otro documento utilizado es Diseño e implementación de un sitema de vídeo bajo demanda P2P[7], sus palabras clave son Vídeo bajo Demanda, Peer to Peer, Segmento, Vídeo Streaming, Round Trip Time(RTT). Este proyecto estudia diferentes arquitecturas de VoD y redes P2P, con la finalidad de crear una aplicación capaz de proporcionar vídeo bajo demanda dentro de

una arquitectura P2P. Este documento fue utilizado para estudiar las redes P2P y su funcionamiento y como integrar un reproductor de vídeo en streaming.

Avances en la tecnología permitieron crear redes P2P con streaming de vídeo que siguen siendo relevantes años después de su creación, como por ejemplo Pop-Corn Time y WebTorrernt.

Estos dos programas utilizan el protocolo BitTorrent, diseñado para el intercambio de archivos entre clientes o para redes P2P, pero estos dos programas no se quedan ahí: también son capaces de reproducir vídeos a medida que se descargan. Estos dos programas siguen en funcionamiento y se pueden utilizar, aunque Pop-Corn Time tuvo ciertos problemas que hicieron que el desarrollo se dividiera dado a problemas legales.

Además de estos programas también cabe destacar Napster y Audiogalaxy como ejemplos de redes P2P centralizadas relevantes.

Napster nace en Junio de 1999 y crea la primera red P2P de intercambio de archivos. Funcionaba de manera centralizada: un servidor guardaba índices a los clientes y, si un cliente quería descargar una canción, el servidor se encargaba de conectarlo con el cliente que la tenía.

Figura 2.13: Napster

Figura 2.14: Audiogalaxy

Después del cierre de Napster surgió Audiogalaxy, con mejoras como reanudar las descargas y una interfaz web que la hacía muy cómoda de utilizar, y el uso del puerto 80 (HTTP) le permitía saltarse firewalls y otros muros que capaban al resto de aplicaciones P2P.

2.5 Crítica a el estado del arte

Los documentos obtenidos en la fase de investigación que se utilizaron para informarse tienen fallos o cosas que se podrían haber hecho, y en esta sección haremos una pequeña crítica de cada uno de ellos.

El primer documento *Propiedad Intelectual y Redes P2P* no hace ninguna implementación de la red, sino que simplemente realiza un estudio de la arquitectura P2P y ahonda en la propiedad intelectual y las leyes que la afectan.

Otro proyecto es *Implementación de una solución de streaming bajo demanda usando DASH*. Este proyecto va más allá e implementa un sistema de streaming de vídeo adaptativo y, aunque prueba diferentes modos de streaming con diferentes formatos, lo hace bajo una arquitectura Cliente/Servidor.

El último documento que escogido es *Desarrollo de un sitema de live video streaming utilizando raspberry pi y el protocolo DASH* y utiliza el streaming de vídeo para crear un directo.

No se encontraron proyectos similares a este en el repositorio de la ETSINF, pero en la universidad Carlos III de Madrid existe un proyecto muy similar: *Diseño e implementación de un sitema de vídeo bajo demanda P2P*. Su objetivo es el mismo que el de este proyecto pero su desarrollo es diferente: trabajan con Java y crean una interfaz para el cliente. Además, es una red P2P híbrida, mientras que nosotros buscaremos una aproximación web y una arquitectura centralizada.

2.6 Propuesta

Nuestra propuesta es llenar un hueco en el repositorio de proyectos, pues durante la investigación en RiuNet no se han encontrado proyectos similares a este. De hecho, no se halló una implementación de una red P2P centralizada.

Se diseñará y creará una red P2P centralizada para streaming de vídeo almacenado desde un punto de vista web, en el que utilizaremos JavaScript y NodeJS principalmente. Adicionalmente, se hará uso de diferentes herramientas como módulos de NodeJS o librerías de JavaScript y el navegador funcionará como reproductor.

CAPÍTULO 3
Análisis de problema

Ahora que ya hemos finalizado el estudio del estado del arte y la fase de investigación, podemos empezar a ver cuál es el problema, pues no existe un proyecto dedicado a la implementación de una red P2P. Esto puede deberse a diferentes causas: a nadie se le había ocurrido aún, falta de herramientas o tecnología para llevarla a cabo o falta de conocimiento.

Esto hace que sea una gran oportunidad comenzar un proyecto para implementar una red P2P centralizada donde además los nodos podrán transmitir streaming de vídeo.

3.1 Análisis del marco legal y ético

Desgraciadamente, las redes P2P son más conocidas por la piratería que por sus ventajas sobre la arquitectura Cliente/Servidor, y esto nos obliga a analizar su marco legal y ético, pues un incorrecto uso de esta red puede conllevar problemas.

Uno de los principales problemas el la propiedad intelectual: en España se añadió el Real Decreto Legislativo 1/1996 el 12 de abril[13], que se convirtió en la ley de Propiedad Intelectual en 2014 debido a las diferentes modificaciones para adaptarse a las nuevas tecnologias.

Esta ley contiene varíos artículos donde define qué es un *autor* y quién se beneficia de esta ley. Resumiendo, la ley española todavía no acutúa contra el usuario final que consume contenido con derechos de autor ilegalmente. Las leyes son más duras con los usuarios en Estados Unidos, aunque en España seguramente cambiará próximamente e introducirá una búsqueda del usuario final.

3.2 Identificación y análisis de soluciones posibles

La solución es crear una red P2P centralizada capaz de emitir vídeo en streaming, y para eso podemos utilizar las herramientas que se han visto en el capítulo anterior.

Para crear una red P2P centralizada podríamos utilizar PeerJS pero, a la hora de probarlo, se vio que el servidor que implementaba no contaba con las funcionalidades necesarias y no era capaz de enviar mensajes directamente al cliente. Se probó también SimplePeer, pero no cuenta con un servidor y la fase de *signaling* debería implementarse con un servidor STUN. También se probó integrarlo con el servidor de Socket.io pero, al ser dos herramientas que servian para lo mismo, se decidió utilizar solo una y no mezclar. Se probó a usar la tecnología WRTC en JS sin ninguna herramienta, pero la complejidad y la falta de un servidor y fase de *signaling* hizo que recurriesemos a una herramienta más sencilla.

Con respecto a el streaming de vídeo, se diseñó y probó una implementación utilizando la herramienta hls.js y fluent-ffmpeg, de forma que el vídeo.mp4 se convertía a .m8u3 gracias al módulo ffmpeg y se retransmitía como torrente de datos mediante hls, pero sus largos tiempos de conversión y sus escasos resultados a la hora de retransmisión de vídeo hicieron que se buscasen otras alternativas.

3.3 Solución Propuesta

Habiendo diseñado y probado varias opciones, mezclando las herramientas elegidas se vio que la implementación más eficiente sería usar Socket.io para crear la red P2P centralizada y el módulo http de NodeJS para el streaming de vídeo. Adicionalmente, usaremos otros módulos como path, op o fs para completar las diferentes funcionalidades del cliente y servidor.

Los clientes se conectarán con el servidor y el servidor será capaz de hablar con cada uno de ellos repartiendo la dirección donde se esté hospedando la retransmisión del vídeo que el cliente ha pedido. El cliente, al recibir esto abrirá una ventana del navegador y comenzará a reproducir el vídeo.

CAPÍTULO 4
Diseño de la solución

El diseño de la red se basará en dos objetos clave, el **cliente** y **servidor**, que tendrán que poder comunicarse entre ellos y deberán ser capaces de producir streaming de vídeo. Además, el servidor implementara la fase de *signaling*.

A continuación se expondrá la arquitectura de estos dos objetos y más adelante entraremos más a fondo en las diferentes funcionalidades que tienen.

4.1 Arquitectura del Sistema

Dividiremos esta sección en dos, hablaremos primero del servidor y después del cliente.

4.1.1. Servidor

El servidor es piedra angular de esta red, pues todas las conexiones pasarán por el y es el que se encargará de la fase de *signaling*, administrando las conexiones y guardando a los clientes conectados.

Este servidor tiene que ser capaz de retransmitir vídeo pues en una red P2P centralizada el servidor también almacena archivos que pueden ser entregados a los clientes y, de esta manera, la próxima vez que otro nodo pida ese vídeo, este lo redigirá al nodo que recién descargó el archivo. Esta será nuestra forma de expandir las retransmisiones P2P.

Primero, expondremos la estructura del directorio del servidor. Puesto que es una red P2P centralizada, el servidor será capaz de almacenar archivos para compartirlos cuando sea necesario así que, dentro de la carpeta 'Servidor', donde se encuentra el código del servidor, habrá una carpeta que nombramos 'vídeos' donde se guardarán los vídeos que deseamos compartir.

El esquema de la estructura del servidor será la siguiente:

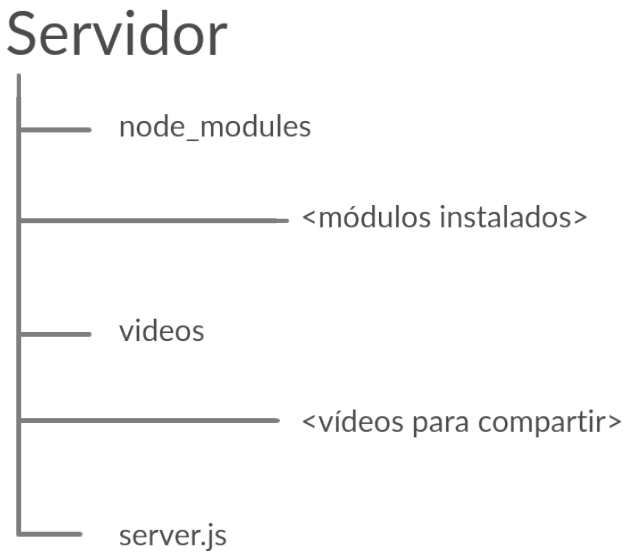

Figura 4.1: Estructura del directorio del servidor

4.1.2. Cliente

El cliente en esta red tiene que ser capaz de conectarse al servidor y poder comunicarse con él; para ello, necesitará ser capaz de compartir los archivos que tiene y enviar peticiones de búsqueda de vídeos al servidor. También tiene que poder reproducir y descargar el vídeo que estaba buscando. Por último, el cliente debe montar un streaming de vídeo cuando sea necesario.

La estructura del directorio del cliente será la misma que la del servidor, contrendrá una carpeta con los módulos instalados que utilizará NodeJS, otra donde guardaremos los vídeos que estemos dispuestos a compartir y el código del cliente.

Su estructura quedaría de la siguiente forma:

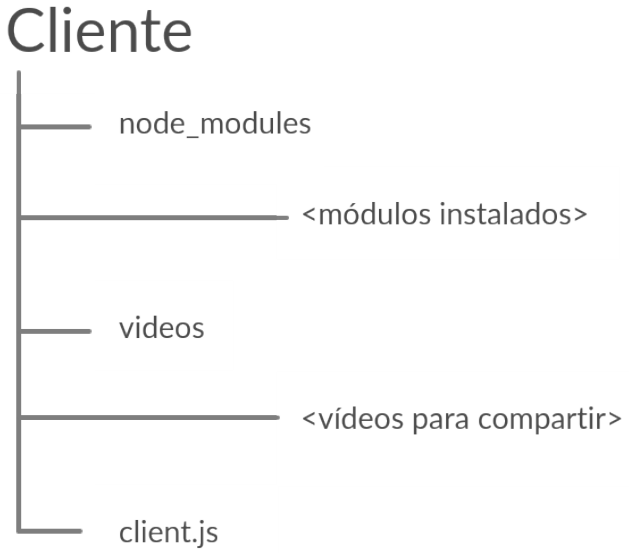

Figura 4.2: Estructura del directorio del cliente

4.2 Diseño detallado

En esta sección se ahondará en el funcionamiento del cliente y servidor, explicando algunas de sus funcionalidades como el envío de mensajes entre ellos, la fase de *signaling* y el streaming de vídeo, tanto para el cliente que lo hospeda como para el que lo recibe.

Puesto que el método de emisión de vídeo es el mismo para el servidor y para el cliente, dividiremos esta sección en tres: primero hablaremos de las funcionalidades del servidor, después continuaremos con el cliente y finalizaremos con el método de streaming de vídeo que se utilizará.

4.2.1. Servidor

Comenzaremos por el método de enviar y escuchar mensajes, para la fase de *signaling* y los mensajes que se envíen en la red en general se utilizará el puerto 3030.

El servidor utilizará la versión de Socket.io destinada a servidores para administrar las conexiones. El servidor, una vez conectado, esperará a conexiones de clientes; una vez se establezca una conexión, el servidor recibirá un objeto *socket* que contiene datos del cliente que se ha conectado.

Una vez que obtenga el objeto, *socket*, el servidor obtiene su identificador y su dirección IP y los guarda en un array, de forma que en la posición [identificador de socket] guarda la IP del cliente.

A continuación, el servidor se quedará esperando los mensajes que envie ese cliente, que pueden ser distintos; uno por cada acción que desee el cliente:

- El servidor espera una petición de compartir vídeos, este mensaje se identifica con el evento 'share' y viene con un array de nombres de vídeos que el cliente desea compartir. El servidor recogerá el array y lo introducira en el Map donde la clave es el nombre del vídeo y el valor es el identificador del cliente que lo tíene. Se guardarán sin la extenson del archivo.
- El servidor también esperará por una petición de ruta, se identifica con el evento 'route' esta es una funcion que permite al cliente saber cuantos clientes a parte del existen en la red y la ruta que hace el mensaje. El servidor enviara un mensaje a todos los demás clientes y recibira uno de vuelta que enviara al cliente que hizo la petición de ruta.
- La siguiente petición que espera el servidor es la más importante, la petición de búsqueda de vídeo y se identifica con el evento 'search'. Esta petición puede desenvocar en diferentes resultados dependiendo quiént tenga el vídeo que busca el cliente:

    1. El servidor tiene el vídeo: No existe ningun valor para el nombre del vídeo que se busca, esto significa que ningun cliente tiene el vídeo, así que el servidor buscará si el tiene ese vídeo, si es así comenzara el stream y le enviará un mensaje al cliente con la dirección donde el vídeo esta siendo emitido.

2. Un cliente tiene el vídeo: En ese caso el servidor encontrará el identificador del cliente que tiene el vídeo y enviará un mensaje al cliente que tiene el vídeo para informarle de que otro cliente quiere verlo, este mensaje se identificara por el evento 'set-stream' y contendrá el nombre del vídeo, su dirección ip, y el identificador del cliente que quiere ver el vídeo. El servidor entonces se quedará a la espera de recibir el mensaje de confirmación del cliente que monta el stream, este mensaje viene con el evento de 'stream-ready' y contiene la dirección donde el vídeo esta siendo emitido y la identificación del cliente que quiere verlo, en cuanto reciba esto el servidor enviara un mensaje al cliente que queria ver el vídeo con el evento 'listen-stream' con la dirección de la emisión.
3. Nadie tiene el vídeo: Simplemente después de comprobar que ningun cliente ni el servidor tiene el vídeo, envia un mensaje al cliente que pidió ese vídeo informandole que ese vídeo no se encuentra en la red.

En el siguiente diagrama de flujo se ve de manera más simple como funciona este evento.

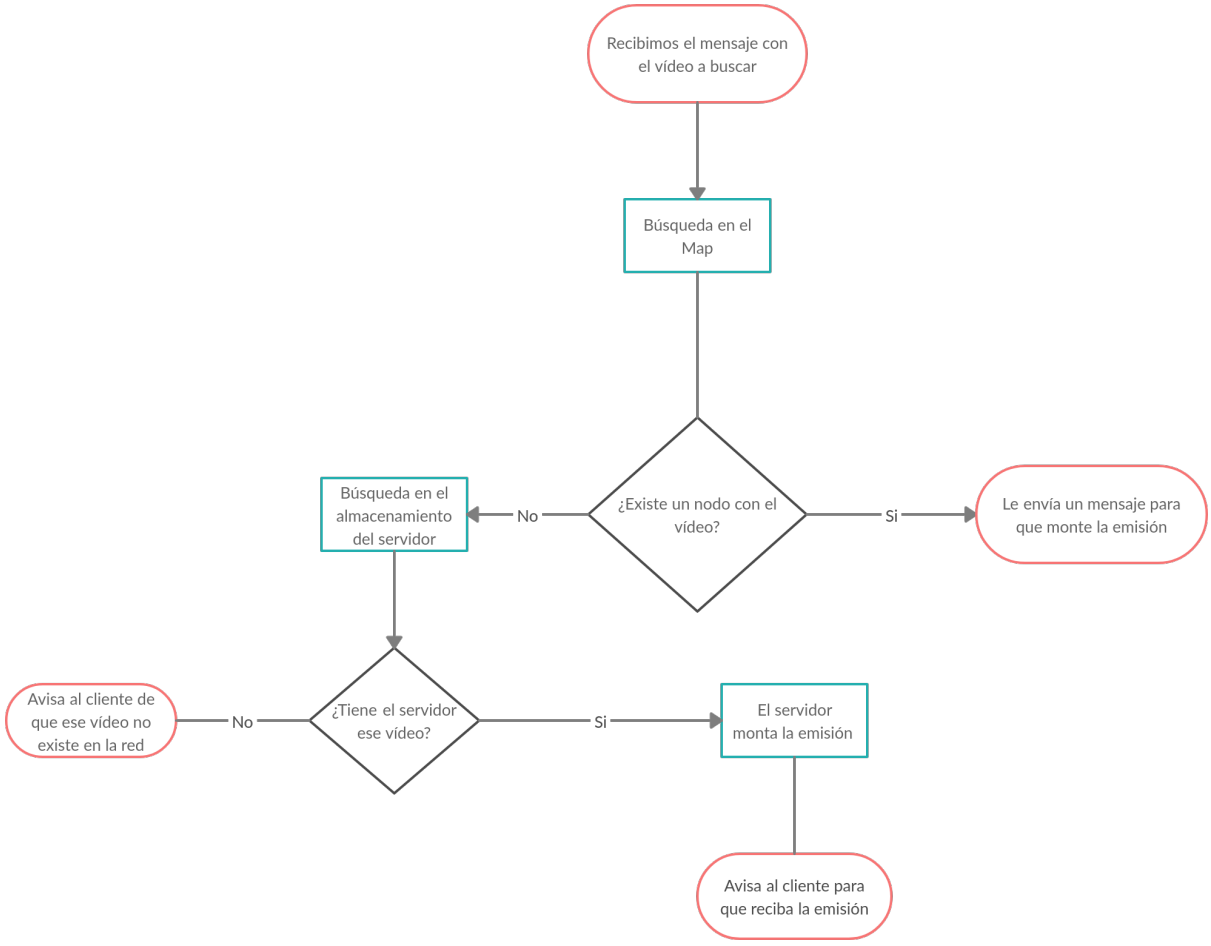

Figura 4.3: Diagrama de flujo para el evento 'Search'

La fase de *signaling* se puede ver claramente en la petición de búsqueda: el servidor guarda identificadores de clientes y su dirección IP para, más tarde, rediri-

gir a otros clientes en búsqueda de archivos que tienen y el servidor ha guardado para, de esta forma, conectarlos entre ellos.

4.2.2. Cliente

El cliente en esta red tiene que ser capaz de conectarse al servidor y poder comunicarse con el, necesitará ser capaz de compartir los archivos que tiene, y enviar peticiones de búsqueda de vídeos al servidor, también tiene que poder reproducir y descargar el vídeo que estaba buscando, por último el cliente deve montar un streaming de vídeo cuando sea necesario.

Para lograr que el cliente tenga todas estas funcionalidades, hemos elegido que se inicie en la terminal y se controle por comandos.

Para iniciar, el cliente abrirá una consola en el directorio, donde se encuentra *client.js*, y escribirá 'node client.js -h <dirección ip del servidor><comando>'. Cada comando iniciará una función diferente:

- share <lista de nombres>: Este comando, primero espera a que el servidor le envie el mensaje de confirmación de conexion, después manda la lista de nombres al servidor con el evento 'share'.
- test: Este es un comando para comprobar que el cliente tiene conexión con el servidor.
- search <nombre del vídeo>: Esta es la función principal de un cliente, con el argumento search, el cliente espera a recibir la confirmación de conexión, y después le envia un mensaje al servidor con el evento 'search' y el nombre del vídeo que desea ver.
- autoshare: Cuando quieres compartir una gran cantidad de vídeos o todos los que tienes, este comando hace una lista de todos los nombres de los archivos de la carpeta 'videos' y los envia al servidor con el mensaje 'share'.
- route: La finalidad de este comando es saber cuantos clientes hay conectados a la red, a parte de ti, además devuelve una ruta de los dispositivos por los que ha pasado el mensaje, el número de veces que te devuelva el mensaje será el número de clientes conectados, descontandote a ti.

A parte de esto, un cliente también está escuchando por si alguien necesita algún vídeo que él tiene, estará esperando a mensajes del servidor con el evento 'set-stream'. Una vez reciba un mensaje con este evento, ejecutará el método de streaming de vídeo y, cuando este finalice, enviará un mensaje al servidor avisando de que el stream está listo. Este mensaje irá con el evento 'stream-ready' y contendrá la dirección del stream y la identificación del cliente que desea consumir la emisión.

El cliente que espera a que le den la dirección del stream aguarda a recibir un mensaje del servidor con el evento 'listen-stream', que contiene la dirección donde se hospeda el stream del vídeo. Una vez recibido el mensaje, el cliente abrirá automáticamente una pestaña en el navegador predeterminado con la dirección recibida, e instantaneamente el vídeo comenzará a reproducirse.

4.2.3. Método de streaming de vídeo

El streaming de vídeo almacenado tiene ciertas características:

- Puedes ver el vídeo mientras lo descargas, gracias a un buffer que va guardando los datos descargados y los ordena, normalmente tienen un rángo, por ejemplo un buffer de 4Mb contendrá los siguientes 4Mb de vídeo y a medida que el vídeo avance el bufer ira descargando más.
- Gracias a que esta siendo descargado, el vídeo puede avanzarse, retrasarse o pausarse.
- Opcionalmente, el vídeo tiene diferentes canales con resoluciónes diferentes, de manera que el consumidor del vídeo puede ajustar la calidad conforme a su banda ancha, o puede ser automático.

El método de stream que hemos implementado, tanto en el servidor como en el cliente, funciona de la siguiente manera:

Se crea un servidor que trate peticiones y envíe respuestas usando el módulo http. Esto lo conseguimos pasándole como argumento una función con dos argumentos, uno para peticiones y el otro para respuestas.

Una vez dentro, cogemos las características del vídeo que vamos a emitir, más concretamente el tamaño total del archivo. Una vez tenemos el tamaño total, toca dividir el vídeo en partes. Para eso, el cliente nos enviará una petición con la cabecera 'Range'; esta cabecera indica la parte del vídeo que el servidor tiene que enviar. Además, en la misma cabecera se pueden pedir varias partes.

Formateamos la información que obtenemos de la cabecera para que nos sea útil y recibimos un array de partes para enviar al cliente. Una vez tenemos las partes que vamos a enviar, parseamos en valor decimal para tener un punto de inicio y de final. Si el final no se ha podido parsear, se utilizara el tamaño total del vídeo.

Ahora que tenemos un punto de inicio y otro de final para las partes, calcuamos el tamaño de una parte o 'chunk'.

Lo siguiente será crear un stream de lectura con el tamaño de chunk previamente definido directamente del vídeo almacenado y, a continuación, enviaremos un mensaje al cliente avisando de que aún queda más vídeo enviandole un código 206 y el comienzo y final del chunk para informarle de cuáles son las siguentes que debería pedir.

Por otro lado, si el servidor no recibe una cabezera definiéndole un rango, este le enviará todo el vídeo y una respuesta con un 200 OK informándole de que ese era el ultimo bloque. Esto suele suceder en casos donde el vídeo es más pequeño que el tamaño de los bloques que es capaz de descargar el cliente; de esta forma el streaming se adapta al ancho de banda del cliente.

Por último, para hacer que las respuestas se envíen después de enviar el vídeo, se ponen en una cola utilizando el método *pipe()* del módulo fs.

Durante la implementación de este método, nos dimos cuenta de varios problemas: el primero siendo que, una vez se comienza la emisión, cualquier cliente

puede entrar. Una solución para este problema es que, una vez se conecte un cliente, el puerto emisor se cierre a nuevas conexiones.

Otro problema es que, si varios clientes quieren ver un vídeo del mismo nodo, este no puede emitirlos mediante el mismo puerto y por eso se puso un contador y, cada stream que se cree, se sumará uno al contador de forma que, si dos clientes reciben vídeo de un mismo nodo, uno lo hará del puerto n y el otro del puerto n+1.

Se recomienda leer el segundo capítulo *Introdución al HTTP* de *Computer Networking: A Top-Down Approach*, pues explica detalladamente el funcionamiento del protocolo HTTP.

4.3 Tecnologia utilizada

En esta sección se expondrán las herramientas que hemos elegido para llevar a cabo el desarrollo del proyecto, hablaremos del dispositivo donde se ha llevado a cabo, el framework utilizado e incluso el porqué de la selección de las herramientas para la implementación de la red P2P centralizada y el streaming de vídeo.

El desarrollo se ha hecho en un portatil HP, en la partición con Ubunto 20.04 como sistema operativo.

Para programar se ha utilizado Visual Studio Code y JavaScript, debido a la cercanía y experiencia que tengo con ellos, además JavaScript cuenta con gran número de herramientas y una base de usuarion enorme.

Utilizamos NodeJS como entorno de ejecución por su potencia y utilidad y también debido a que facilita el añadir nuevas herramientas en forma de módulos.

Para la creación de la red P2P centralizada se utilizo Socket.io por su facilidad de uso y lo completa que es; además, fue la única que cumplió nuestras espectativas durante la ronda de pruebas de las herramientas investigadas.

Utilizamos el módulo http de NodeJS para el streaming de vídeo porque permite configurar una gran cantidad de cosas de las peticiones HTTP y hace muy fácil modificarlas para conseguir un streaming de vídeo como hha sido planeado.

Adicionalmente, se utilizaron varios módulos de NodeJS para llevar a cabo ciertas funciones del cliente y servidor. El cliente cuenta con el módulo op que permite abrir automáticamente un enlace, utilizado para abrir el enlace donde se hospeda el stremaming de vídeo. También utilizamos los módulos path y fs para obtener el camino hacia la carpeta donde se guardan los vídeos y leer los nombres de los vídeos en dicha carpeta.

CAPÍTULO 5
Funcionamiento

En este capítulo se pretende enseñar el funcionamiento de diferentes casos de la red diseñada, ahora que hemos dejado claro las funcionalidades del servidor y el cliente.

Para explicar el funcionamiento se harán varias trazas suponiendo diferentes situaciones y se acompañarán con imágenes mostrando el flujo de mensajes y streaming para un entendimiento más claro.

5.1 Primera situación: Ningún nodo tiene el vídeo

En este caso el servidor será el que monte la emisión de vídeo, y el flujo de datos será el siguiente:

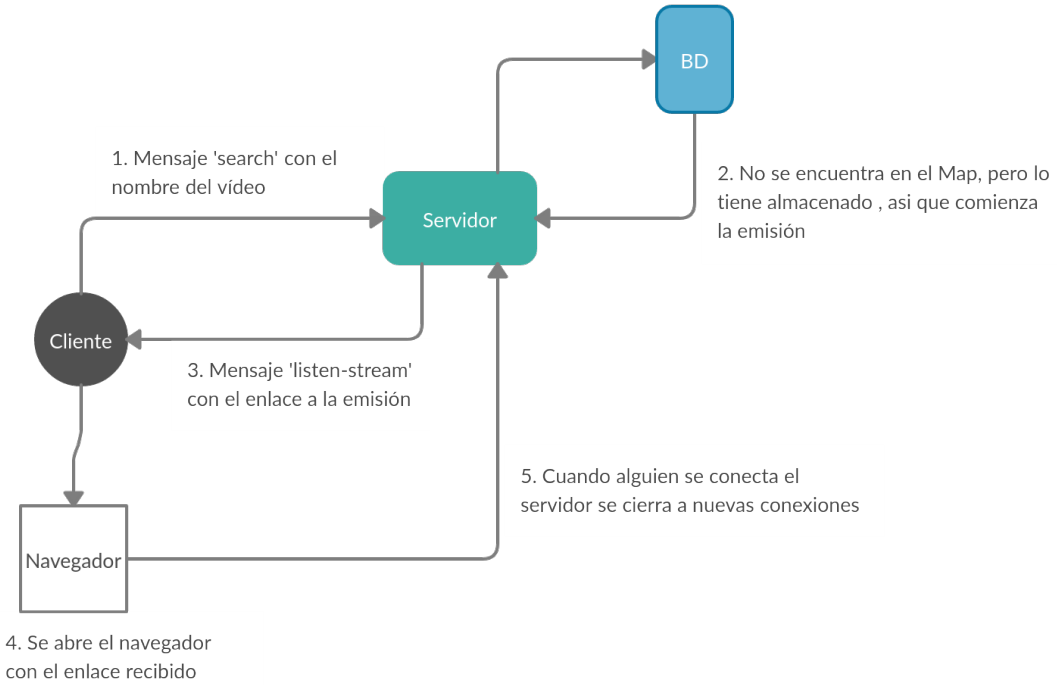

Figura 5.1: Diagrama de la primera situación

1. El cliente se conecta al servidor y le envia un mensaje con el nombre del vídeo que quiere ver.
2. El servidor busca en su base de datos si algún nodo tiene el vídeo, al ver que ninguno cuenta con el, buscará en su carpeta de vídeos y si lo encuentra, automáticamente montara el stream.
3. Ahora que el stream se ha montado, enviara un mensaje al cliente indicandole donde se aloja el stream.
4. El cliente recibira este mensaje y abrirá una pestaña del navegador predeterminado para visualizar el vídeo.
5. En el momento que el cliente abra el stream, el servidor parará de aceptar conexiones para que solo el cliente pueda entrar a ese vídeo.

5.2 Segunda situación: Otro nodo tiene el vídeo

Este será nuestro caso más común. Existirá otro nodo con el vídeo y este será el que monte el stream. El flujo de datos se ilustra en la siguiente imagen:

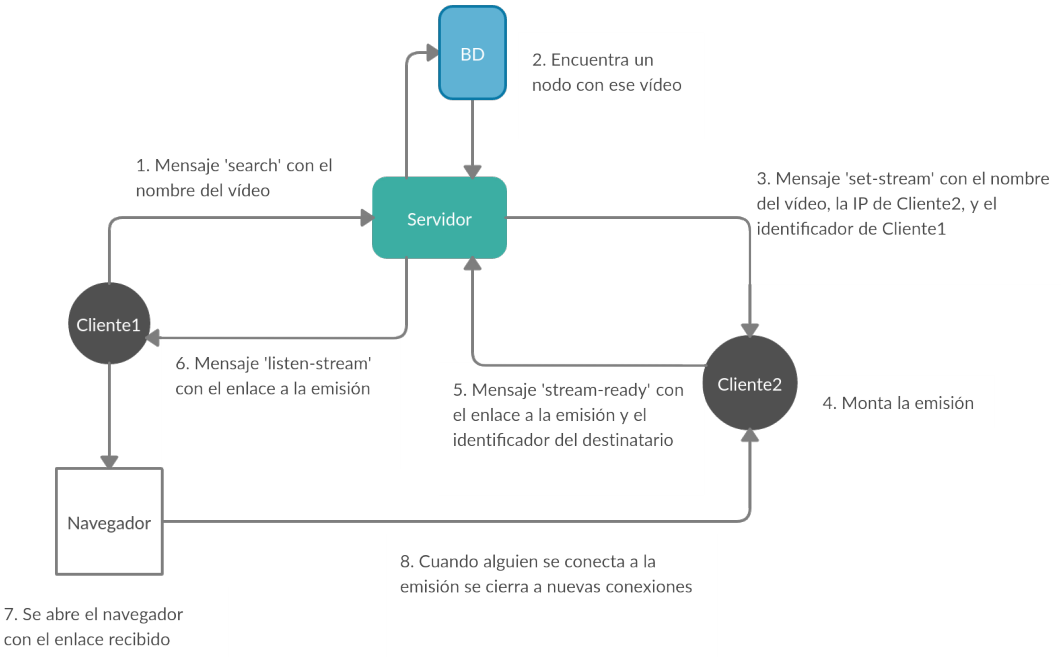

Figura 5.2: Diagrama de la segunda situación

1. El cliente conecta con el servidor y le envia un mensaje con el nombre del vídeo que quere ver.
2. El servidor encuentra un nodo que tiene el vídeo.
3. Cuando encuentra el nodo, le envia un mensaje avisandole de que tiene que montar un stream para cierto cliente.

4. El nodo con el vídeo recibe el mensaje y comienza a montar el stream.

5. Una vez se monte el stream, el nodo enviara un mensaje al servidor avisandole de que el stream esta listo y que tiene que notificar al cliente que queria ver el vídeo.

6. El servidor recibe el mensaje y le envia al cliente el enlace donde se hospeda el stream.

7. Finalmente el cliente recibe el enlace al vídeo y abre una ventana del navegador por defecto automáticamente que lleva al vídeo.

8. Una vez el solicitante entre en la emisión del vídeo, el nodo que ha montado la emisión cerrara ese puerto a nuevas conexiones.

CAPÍTULO 6
Desarrollo de la solución propuesta

El desarrollo de la red P2P comenzó haciendo pruebas con las diferentes herramientas que encontramos en la fase de investigación. Se crearon diferentes implementaciones pero la mejor en términos de simpleza de código y eficiencia era Socket.io así que comenzamos a pensar en las funcionalidades que necesitaban tanto el cliente como el servidor.

Al principio solo se implementó el evento 'share' y 'search', creamos la base de datos en el servidor que es un Map. Elegimos esta estructura de datos por su eficiencia en la búsqueda de un valor mediante su clave. También creamos el array donde se guardaría la IP de los clientes.

Debido a temas de pruebas, creamos los comandos del cliente 'test' y 'route' para poder ver el estao de la red y comprobar que todo funcionaba correctamente.

A continuacón, empezamos a hacer pruebas con las herramientas de streaming de vídeo. Comenzamos con hls.js y ffmpeg pero, debido a sus resultados subóptimos y problemas con el torrente de datos que retransmitian, decidimos cambiar el método de streaming, y es así como elegimos el módulo http.

Una vez implementamos el método de streaming y comprobamos que funcionase correctamente, nos dimos cuenta de que había que estructurar los directorios del servidor y cliente para poder tener un sitio donde se guardasen los vídeos, y así se creó la carpeta 'videos' y utilizamos los módulos path y fs para conseguir los nombres de los vídeos que se guardaban ahí.

Con el tiempo nos dimos cuenta de que era tedioso introducir los nombres de todos los vídeos de la carpeta cuando querias compartirlos, así que creamos el comando 'autoshare' para automatizar esta tarea.

Con todo esto implementado y una vez probado su funcionamiento, comenzamos a optimizar el sistema de diversas formas como, por ejemplo, hacer que el cliente abriese automáticamente una ventana de navegador cuando recibia el enlace, o hacer que el streaming se cerrase a nuevas conexiones una vez el cliente habia entrado. Tambíen se encontró un problema cuando dos clientes pedían un vídeo al mismo cliente, este intentaba utilizar el mismo puerto para retransmitir dos streamings diferentes, así que se creó un contador que sumaría uno cada vez que un cliente se conectase. De esta forma, el primer cliente tendria el streaming de vídeo en http://<dirección ip>:8000 y el segundo en http://<dirección ip>:8001 y así sucesivamente.

Se implementó también la capacidad de decidir el servidor al que se desee conectarse con el cliente a la hora de iniciarlo; se escribe la dirección IP del servidor al que se desea conectarse cuando se ejecuta el script de la siguiente forma 'node client.js -h <dirección IP del servidor><comandos>'.

CAPÍTULO 7
Pruebas

Para hablar de las pruebas que se realizaron con motivo de la validación del proyecto y ver si se llegó a lograr los objetivos que se propusieron, antes debemos hablar de los dispositivos que se utilizaron en estas pruebas:

- Servidor: Portatil HP, Procesador Intel Core i5, CPU 2.3GHz, RAM 4Gb, S.O Ubuntu 20.x de 64 bits. Este fue el servidor de la red en todas las pruebas.
- Cliente 1: Portatil acer, Procesador Intel Core i5, CPU 3.5GHz, RAM 8Gb, S.O Windows 10 de 64 bits. Este fue usuario utilizado en todas las pruebas.
- Cliente 2: Portatil HP, Procesador Intel Core i7, CPU 4.4GHz, RAM 8Gb, S.O Windows 10 de 64 bits.
- Cliente 3: MacBoock Air 13", RAM 8Gb, S.O IOS.
- Red 1: WiFi provisto por Vodafone a la que estaba conectado el Servidor durante todas las pruebas.
- Red 2: Datos del telefono.

A continuación se encuentra una lista de las pruebas realizadas para la comprobación del proyecto, seguida de un breve resumen del desarrollo de estas con sus resultados:

7.1 Primera prueba

Esta es la primera prueba y por tanto será la más básica, utilizando el comando de 'test' del cliente probaremos para ver si el cliente es capaz de conectarse al servidor. Para esto levantamos el servidor con el comando 'node server.js <dirección IP>' y una vez que el servidor este funcionando se iniciara el cliente con 'node client.js -h <dirección IP del servidor>test', se espera recibir por consola un mensaje confirmando que el cliente ha sido capaz de conectar con el servidor. El resultado de esta prueba fue exitosa.

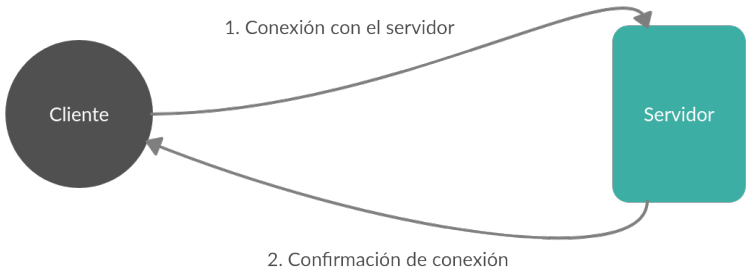

Figura 7.1: Diagrama de la primera prueba

7.2 Segunda prueba

En la segunda prueba probaremos que efectivamente se pueda pasar información en los mensajes de cliente a servidor. Necesitaremos el servidor y dos usuarios, consistira en utilizar el argumento 'route'. En un cliente y el resultado será un mensaje por consola que muestre los dispositivos que ha recorrido. El resultado fue exitoso.

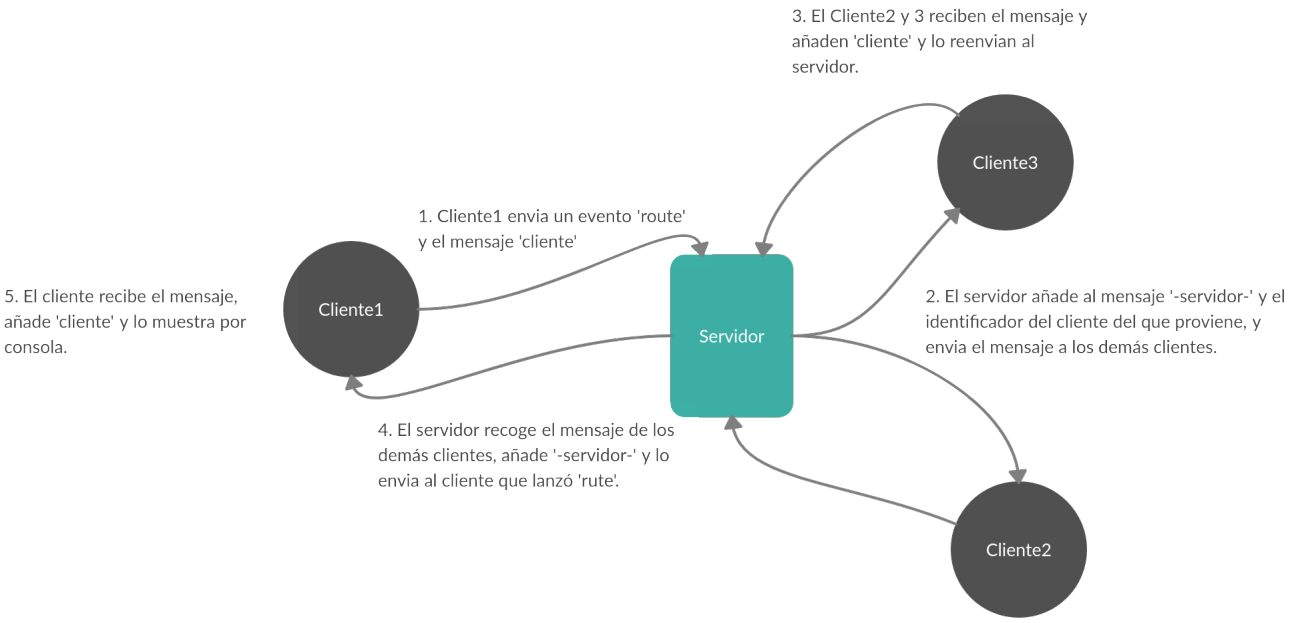

Figura 7.2: Diagrama de la segunda prueba

7.3 Tercera prueba

Esta prueba consistira en comprobar el streaming de vídeo, comprovaremos su eficacía y si se puede retroceder, adelantar, pausar y descargar. Utilizamos un script a parte en el servidor, este script localizaria el vídeo elegido y lo pondria a reproducir el vídeo en el puerto deseado, el resultado sería que el vídeo se reproduce y se puede manejar a traves del enlace http://localhost:8000 o http://<dirección ip>:8000. El resultado fue exitoso.

7.4 Cuarta prueba

Ahora comprobaremos que el servidor es capaz de montar un streaming de vídeo, para esto utilizamos el servidor y el cliente 1, el cliente enviara una petición de búsqueda al servidor, este montara un stream y enviara un mensaje al cliente 1 con el enlace, este lo recibirá y abrira una ventana del navegador con el vídeo, el resultado será que el video se retransmitira atuomáticamente una vez el servidor monte la emisión. El resultado fue exitoso.

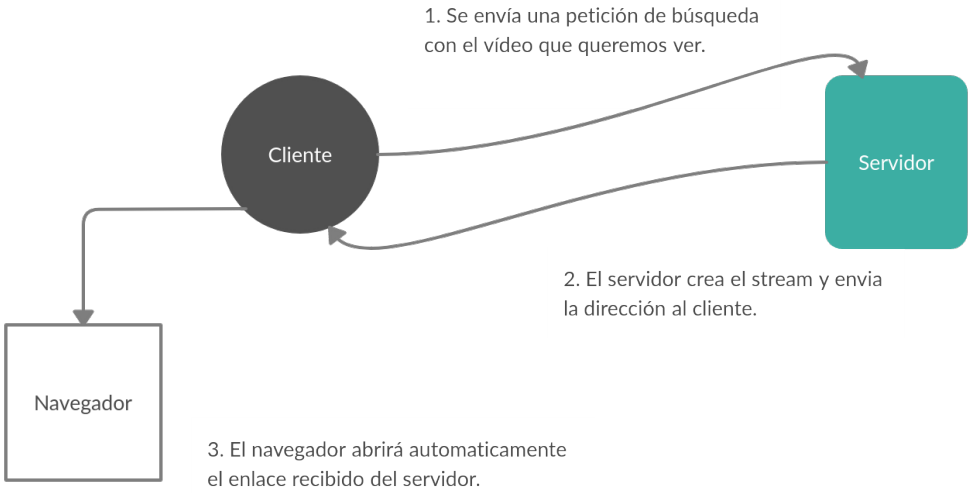

Figura 7.3: Diagrama de la cuarta prueba

7.5 Quinta prueba

Siguiente, comprobaremos que un cliente puede proveer un streaming de vídeo a otro cliente dentro de la red, este sería una prueba básica de la red P2P centralizada. Se utilizaron los dos clientes y el servior, Cliente 1 pregunta por 'Platano' un vídeo que tiene Cliente 2, el servidor completara la fase de *signaling* exitosamente mostrando por pantalla todas sus acciones, y Cliente 1 automaticamente comenzara a reproducir la emisión de vídeo cuando reciba el mensaje de confirmación del servidor. El resultado de esta prueba fue un éxito.

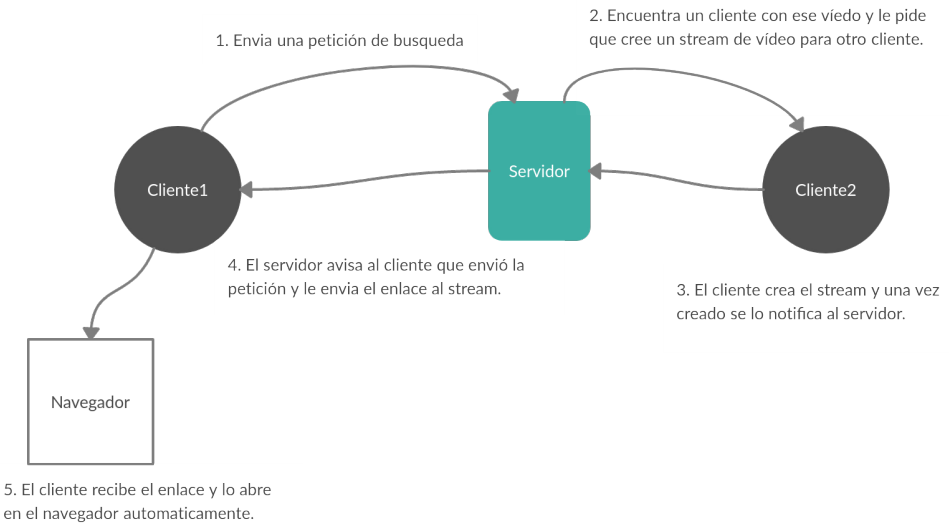

Figura 7.4: Diagrama de la quinta prueba

7.6 Sexta prueba

Ahora comprobaremos que nadie es capaz de entrar en la emisión de un cliente para otro más que el que ha solicitado ese vídeo, tanto si el servidor es el que emite como si lo es un clente. Se utilizaron 3 clientes y el servidor, hacemos que un cliente consuma un stream de otro o del servidor y con otro cliente intentamos conectarnos a la emisión, este cliente debera encontrarse con una página que informa de que no se encontro nada en ese enlace. El resultado fue exitoso.

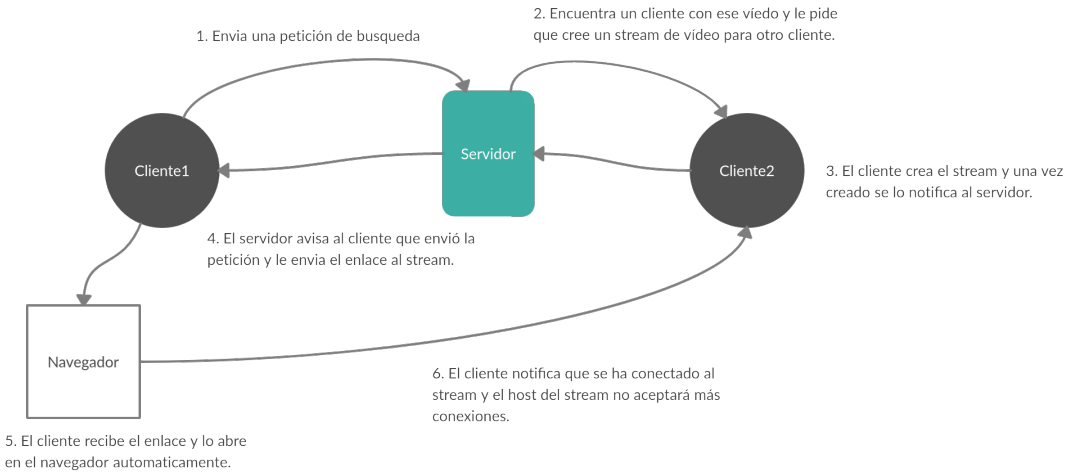

Figura 7.5: Diagrama de la sexta prueba

7.7 Septima prueba

La siguiente prueba tiene como finalidad comprobar que tanto servidor como cliente son capaces de crear diferentes emisiones para diferentesclientes. Se utilizaron los dos clientes y el servidor, primero dos clientes se conectarán al servidor y serán capaces de reproducir el vídeo que han solicitado y después dos clientes podrán conectarse a un tercer cliente que será capaz de retransmitir los diferentes vídeos que piden los solicitantes. El resultado fue exitoso.

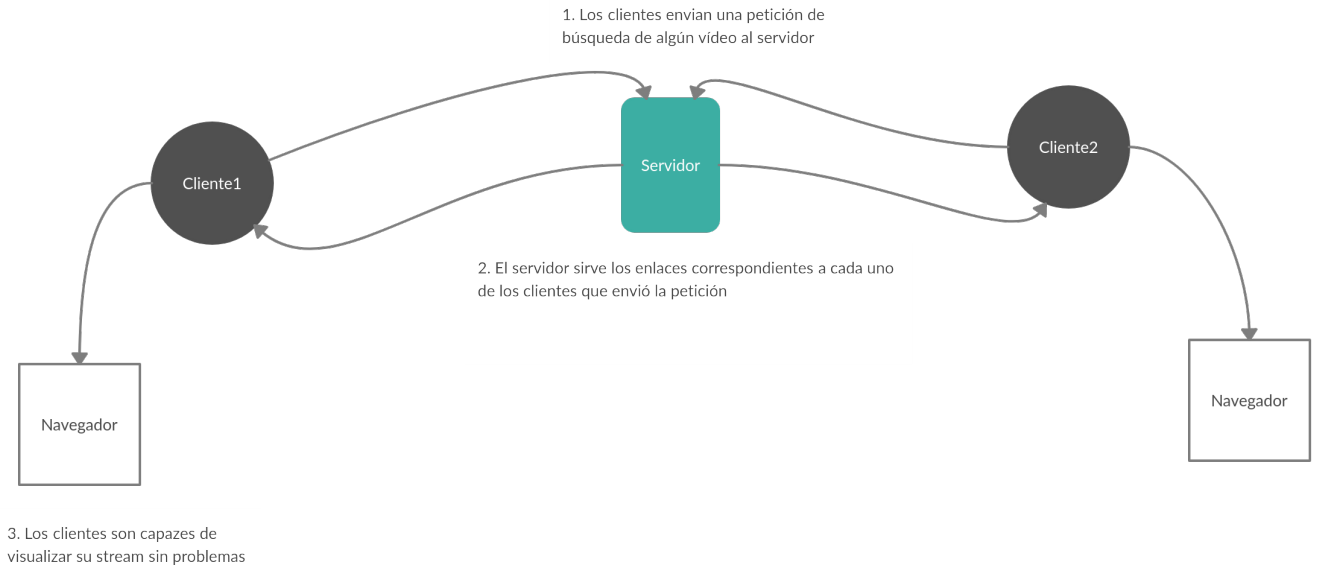

Figura 7.6: Diagrama de la séptima prueba con el servidor

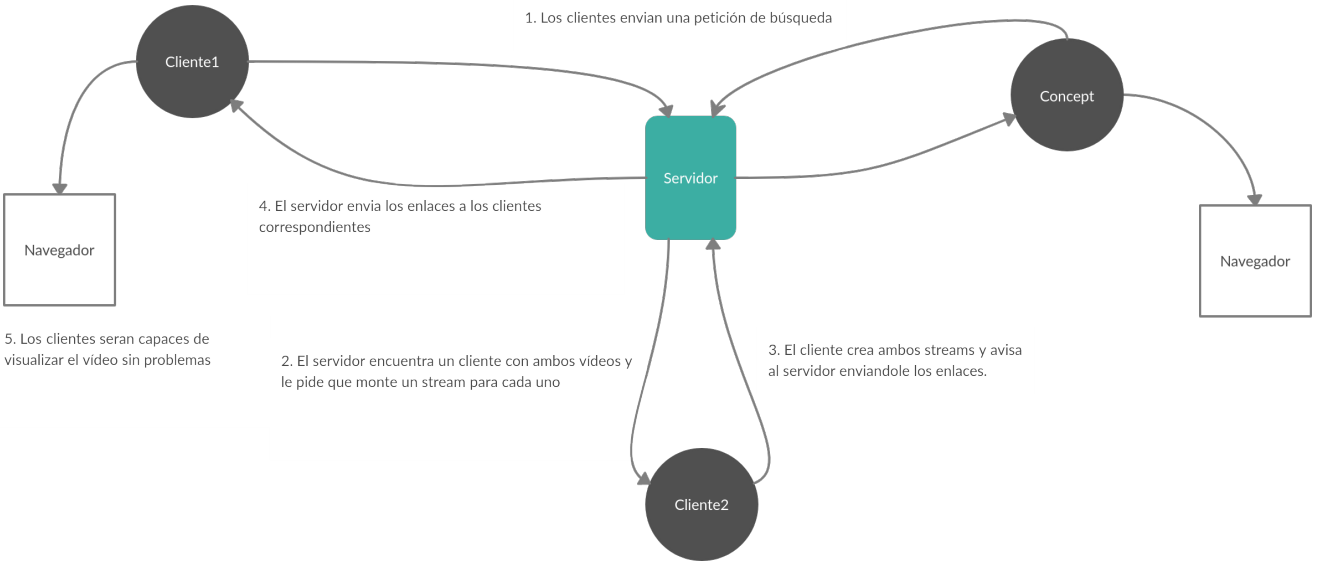

Figura 7.7: Diagrama de la séptima prueba con los clientes

CAPÍTULO 8
Conclusiones

Este proyecto tiene varios objetivos, pero el principal y el que le brinda nombre a este documento es la creación de una red P2P centralizada capaz de transmitir vídeo en streaming.

Para cumplir el primer objetivo, necesitábamos que nuestra arquitectura cumpliera ciertos requisitos, que eran:

1. La red debe tener las características de una red P2P centralizada que se explicaron en el primer capítulo, conexión con un servidor central que almacena archivos y transferencia de archivos entre nodos.
2. El streaming de vídeo será capaz de retrasar el vídeo, adelantarlo y pausarlo, además de descargalo.

Por las pruebas que se realizaron en el capítulo de validación, podemos concluir que esta red cumple con los requisitos de una red P2P centralizada y es capaz de transmitir vídeo en streaming satisfactoriamente.

Se creó un servidor que se encarga de administrar conexiones entre clientes gracias a la implementación de un método de *signaling* y se estudiaron varios métodos de streaming hasta dar con un metodo simple y eficaz.0 También se diseñó una estructura para cliente y servidor homogénea y se trabajó para la optimización de esta red haciéndola tolerante a diferentes fallos.

Poro otro lado, este proyecto tenía como objetivo plasmar todo lo que se ha aprendido y estudiado en una memoria con el fin de servir de guía o fuente de información para personas interesadas. Se explican una variedad de heramientas utilizadas y probadas, la útilidad y potencial que tienen además de nuestras experiencias trabajando con estas.

Se ha probado el funcionamiento dentro de una red y entre diferentes redes, pero solo ha funcionado dentro de la misma vlan. Esto se debe a que necesitaremos conseguir la IP pública de los clientes utilizando algún servicio o servidor STUN. A parte de eso, se probó el correcto funcinamiento de la red y el streaming de vídeo.

En resumen, en este proyecto se ha creado una red P2P centralizada para retransmisión de vídeo en streaming con una arquitectura básica, con ciertas limitaciones pero funcional y capaz de mostrar perfectamente cómo funciona una red P2P, concretamente una arquitectura P2P centralizada.

8.1 Relación de trabajo desarrollado con los estudios cursados

A lo largo del grado se estudia el funcionamiento de la red y diferentes protocolos; unos de ellos son BitTorrent y HTTP, y estos se han utilizado para estudiar las redes P2P y streaming de vídeo respectivamente. También se estudian diferentes tipos de estructuras de datos, que, durante la fase de diseño, se estudiaron para elegir cómo implementar la base de datos del servidor -acabó siendo un Map.

En el desarrollo de la fase de investigación y diseño también influyeron la formación en Ingeniería de Software y las diferentes formas de organizar un proyecto y diagramas.

En este proyecto se han utilizado gran variedad de recursos obtenidos durante la carrera, desde estructuras de datos como el Map que gurada la relación entre cliente y los vídeos que tiene, hasta sockets y web. Durante la etapa de planificación de este proyecto, también se puseron en práctica diferentes estrategias de Ingeniería de Software.

Durante la carrera también aprendes a elegir tus batallas, es por eso que se desecharon herramientas que prometían unos resultados excelentes, pero la curva de aprendizaje era demasiado grande y conseguir un dominio óptimo para desarrollar el proyecto requeria demasíado tiempo.

Con respecto a las competencias transversales diría que se pusieron en práctica varias:

1. **Innovación, creatividad y emprendimiento**: No hay ningun proyecto parecido en RiuNet y muy pocos relacionados con la creación de una red P2P. Al no haber mucho donde basarse, he tenido que salir con un diseño y un proyecto propio.

2. **Aprendizaje permanente**: El desarrollo de este proyecto ha hecho que estudie a fondo diferentes herramientas y coja experiencia con ellas, además el estudío sobre las redes P2P realizado fue un esfuerzo que costara olvidar.

3. **Planificación y gestión del tiempo**: Al tener que compenetrar el TFG con el trabajo he tenido que montar un horario y cumplirlo día tras día, esto me ha dado una mayor visión a la hora de organizar proyectos que duren cierto tiempo y una mejor idea de como planificar tareas.

4. **Análisis y resolución de problemas**: A lo largo de este proyecto nos hemos encontrado con variedad de problemas, pero siempre hemos sido capaces de encontrar varias soluciones, una ideal y otra de emergencia en caso de que la ideal no sea eficaz.

Bibliografía

[1] Como implementer *signaling* en una arquitectura WebRTC Consultado en https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/.

[2] Informe sobre redes distribuidas y su topologia de Paul Baran Consultado en https://www.rand.org/content/dam/rand/pubs/research_memoranda/2006/RM3420.pdf.

[3] Artículo sobre el funcionamiento de los servidores STUN/TURN Consultado en https://andrewjprokop.wordpress.com/2014/07/21/understanding-webrtc-media-connections-ice-stun-and-turn/.

[4] DESARROLLO DE UN SISTEMA DE LIVE VIDEO STREAMING UTILIZANDO RASPBERRY PI Y EL PROTOCOLO DASH Consultado en http://hdl.handle.net/10251/127818.

[5] Implementación de una solución de streaming bajo demanda usando DASH Consultado en http://hdl.handle.net/10251/128818.

[6] Historia, análisis y aplicaciones de las redes distribuidas Consultado en http://hdl.handle.net/10251/63883.

[7] DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE VÍDEOBAJO DEMANDA P2P Consultado en https://e-archivo.uc3m.es/bitstream/handle/10016/25275/PFC_JoseLuis_Aldovera_Escamilla.pdf.

[8] P2P Sharing: aplicación android P2P para compartición de archivos Consultado en https://eprints.ucm.es/48889/.

[9] Artículo sobre el consumo de VoD de los españoles Consultado en https://www.media-tics.com/noticia/8807/dips/6-de-cada-10-espanoles-utiliza-algúna-plataforma-de-streaming-de-pago.html.

[10] Porcentaje de internautas que emplearon tecnologías peer to peer (P2P) en la Unión Europea en 2015, por edad Consultado en https://es.statista.com/estadisticas/503800/porcentaje-de-internautas-que-usaron-tecnologias-p2p-en-la-ue-por-edad/.

[11] Propiedad Intelectual y Redes P2P Consultado en http://hdl.handle.net/10251/56040.

[12] Api de WebRTC, parte de su conectividad Consultado en https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Connectivity.

[13] Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia. Consultado en https://www.boe.es/buscar/act.php?id=BOE-A-1996-8930.

[14] Historia de napster y su servicio Consultado en https://www.xataka.com/historia-tecnologica/napster-inicio-auge-caida-servicio-que-puso-jaque-a-industria-musical.

[15] Audiogalaxy en el presente Consultado en https://www.rtve.es/noticias/20101014/retorno-audiogalaxy-ahora-permite-llevarse-musica-todas-partes/361918.shtml.

[16] Wikipedia Contributors *The World of Peer-to-Peer (P2P)*. Wikibooks.

[17] Saikat Basu *The Big Book of BitTorrent*. makeuseof.com.

[18] Quang Hieu Vu, Mihai Lupu y Beng Chin Ooi *Peer-to-PeerComputing: Principles and Applications*. Springer. Capítulos *Architecture of Peer-to-Peer Systems* y *P2P Programming Tools*

[19] James Kurose *Computer Networking: A Top-Down Approach*. Pearson, 6º edicion. Capítulo 2 *Introdución al HTTP*

APÉNDICE A
Abreviaciones

P2P: Peer-to-peer (Cliente a cliente)
VoD: Video on Demand (Vídeo por demanda)
HD: High Definition (Alta Definición)
TFG: Trabajo de Fin de grado
DHT: Distributed Hash Table (Tabla Hash Distribuida)
JS: JavaScript
WebRTC: Web Real-Time Comunication
API: Application Programming Interfaces
STUN: Session Traversal Utilities for NAT
NAT: Network Address Translation
TURN: Traversal Using Relay NAT
ICE: Interactive Connectivity Establishment
UDP: User Datagram Protocol
CDN: Content Delivery Network
HTML: HyperText Markup Language
SSL: Secure Sockets Layer
HTTP: Hypertext Transfer Protocol
HLS: Http Live Streaming
DASH: Dynamic Adaptative Streaming over Http
MPEG: Moving Picture Experts Group
RTT: Round Trip Time
ETSINF: Escuela Técnica Superior de Informática