UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de una aplicación para la exploración e indexación del contenido disponible en la DHT del protocolo BitTorrent

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Jorge Sánchez Sancho

Tutor: José Salvador Oliver Gil

Curso 2019-2020

Resumen

BitTorrent es uno de los protocolos más empleados en la distribución y compartición de archivos a través de la red. El principal problema del protocolo es el descubrimiento del contenido disponible en la red BitTorrent. En este trabajo se realiza un estudio en profundidad del protocolo para después elaborar una aplicación autónoma que permita encontrar todo el contenido disponible. Haciendo uso de la DHT del protocolo es posible explorar el contenido de la red e indexarlo en una base de datos. El objetivo de la aplicación es el de crear una herramienta que explore la red en busca de contenido con el fin de que un usuario pueda ver el todo el contenido disponible de la red. De esta forma, esta aplicación pretende solucionar el principal problema del protocolo.

Palabras clave: BitTorrent, protocolo, nodos, pares, DHT, Bencoding

Abstract

BitTorrent is one of the most used protocols for file distribution and sharing through the network. The main problema of the protocol is the discovery of the content avaliable on the BitTorrent network. In this Project, a study of the protocol is carried out to develope an autonomous application that allows the possibility of find all the avaliable content. Using the DHT of the protocol, it is possible to explore the content of the network and index it in a database. The purpose of the application is to créate a tool that explores the network in order to find find content so that a user can visualize all the content avaliable on the network. This application aims to solve the main problema of the protocol.

Keywords : BitTorrent, protocol, node, peers, DHT, Bencoding.

Tabla de contenidos

Índice de figuras

1. Introducción

Internet se ha convertido en una red de redes a la que millones de usuarios se conectan a diario. Su éxito se debe principalmente a que permite compartir o intercambiar contenido con otros usuarios de forma prácticamente inmediata. Las personas siempre han tenido necesidad de compartir información con las personas. Es por eso que Internet se adapta a esa necesidad de compartir archivos que tienen los usuarios. Hace años, la forma de compartir archivos era con medios físicos, como los tradicionales discos compactos o los disquetes, pero desde la aparición y la constante evolución de Internet la compartición de archivos ha ido mejorando, hasta tal punto que en la actualidad se está utilizando cada vez menos los medios físicos para compartir contenido.

El volumen de tráfico diario de datos a través de Internet está siendo ocupado cada vez más por la transferencia de contenido multimedia entre usuarios. La visualización de contenido multimedia también ha hecho incrementar el volumen de datos que se envía a la red, debido principalmente a las mejoras en los equipos informáticos y en la infraestructura de Internet. Cada vez se cuenta con mejores prestaciones y capacidad de almacenamiento en los equipos junto con un mayor ancho de banda para los usuarios. Las mejoras mencionadas propiciaron la aparición de las nuevas aplicaciones de red y nuevos protocolos que mejoran la experiencia del usuario a la hora de compartir contenido.

Desde la aparición del protocolo BitTorrent, su popularidad ha ido aumentando entre la multitud de usuarios de Internet, al haberse convertido en un protocolo que permite compartir archivos entre estos. En la actualidad hay un gran número de aplicaciones de usuario que implementan este protocolo. Sin embargo, uno de los principales problemas del protocolo es la dificultad de encontrar el contenido disponible que se está compartiendo, siendo normalmente necesario visitar páginas web para buscar el contenido a obtener. Por ello, se pretende desarrollar un componente que permita descubrir todo el contenido que se está compartiendo en la red para solucionar este problema.

1.1 Motivación

En la actualidad, existen múltiples aplicaciones que sirven para compartir archivos con el resto de usuarios que solucionan muchos de los problemas tecnológicos a la hora de compartir contenido. Hace unos años, cuando se quería compartir fotos o vídeos con

otra persona, había que recurrir a medios físicos como un álbum de fotos o los discos compactos, mientras que hoy en día es muy fácil compartir archivos; basta con tener la aplicación instalada. Una de las que destaca en su campo es BitTorrent que permite compartir archivos de forma sencilla.

BitTorrent cuenta con un protocolo complejo, pero con documentación suficiente para poder entender su funcionamiento. Por esto, este proyecto tiene la intención de desarrollar una aplicación que pueda ayudar a las personas a descubrir todos los archivos que se están compartiendo en la red. Por otro lado, también se pretende obtener conocimientos a bajo nivel de cómo se puede enviar archivos a través de la red mediante el intercambio de mensajes.

1.2 Objetivos

El objetivo principal del proyecto es el de desarrollar una aplicación para ordenadores de escritorio que permita descubrir todo el contenido multimedia que está siendo compartido en la red de BitTorrent, haciendo uso de su protocolo. Para lograr esto, será necesario desarrollar un componente que se encargue de descubrir el contenido de la red y que gestione la persistencia de la información, cumpliendo así con el objetivo marcado. Para conseguir esto es necesario alcanzar los siguientes requisitos.

- Desarrollar un mecanismo de búsqueda que permita obtener la información de la red de forma automática.
- Crear un buscador de contenido capaz de ir añadiendo el contenido encontrado a la base de datos de forma autónoma.
- Obtener información detallada de los archivos encontrados en la red. En dicha información se debe incluir al menos el nombre del archivo y su tamaño expresado en la unidad correspondiente.
- Desarrollar un buscador rápido y eficiente para que se pueda ir poblando la base de datos con la mayor velocidad posible.
- Intentar diseñar una aplicación con una arquitectura simple para evitar la complicación del desarrollo.

1.3 Estructura de la memoria

El contenido de la memoria está organizado por apartados en los que se explican aspectos importantes del proyecto. El primer apartado describe la situación actual en la que se encuentra el protocolo BitTorrent y de las aplicaciones que lo implementan. En el siguiente apartado se realiza un análisis del marco legal y ético del proyecto en que se aportan argumentos para determinar la legalidad del mismo. A continuación, se hace una introducción a los conocimientos técnicos necesarios para entender el funcionamiento de este tipo de aplicaciones, en la que se deriva un análisis en profundidad de los aspectos más importantes del protocolo BitTorrent. En los siguientes apartados se realiza un análisis, diseño y desarrollo de la aplicación, concluyendo con una serie de pruebas que derivarán en las conclusiones del proyecto y proponiendo futuras mejoras.

1.4 Convenciones

Se va a proceder a marcar con letra cursiva las palabras que cumplan con al menos uno de los requisitos siguientes para que destaquen respecto al resto del texto.

- El nombre de los actores de la red de BitTorrent.
- Los nombres de las funciones de código empleadas.
- Las claves que contienen los mensajes que se envían a los nodos
- Los ejemplos de codificación.
- La explicación de las siglas de los acrónimos.

2. Situación actual

En la actualidad se pueden encontrar numerosas aplicaciones que implementan el protocolo BitTorrent, ya sean tanto gratuitas como de pago. Las versiones de pago se diferencian de las gratuitas por la eliminación en la aplicación de todos los anuncios que aparecen en estas últimas. Lo realmente importante es que todas ellas permiten realizar el intercambio de contenido multimedia a través de la red. Según el último informe que ha presentado BitTorrent, el número de usuarios activos durante el periodo de un mes supera los cien millones. En esta medición se ha tenido en cuenta los usuarios que emplean el cliente BitTorrent como uTorrent, ambos de la misma empresa. Desde el mismo informe confirman que el uso de red P2P está presente en 220 países, lo que se traduce en unas 400.000 descargas diarias de contenido.

Desde hace un tiempo hay determinados dispositivos que incluyen este protocolo desde su fabricación, como son los servidores NAS. Son pequeños servidores que de forma nativa permiten utilizar el protocolo y compartir archivos. Es por esto, que teniendo en cuenta las cifras de los usuarios mensuales junto con las descargas diarias y con dispositivos que integran este protocolo, que sea uno de los más eficientes y usados para compartir y distribuir contenido.

Un problema vigente de las aplicaciones es que no cuentan con un buscador propio que permita encontrar los archivos disponibles en la red, obligando a los usuarios a visitar portales web en los que realizar la búsqueda del archivo que desean obtener. Cabe destacar que en estos portales normalmente no se encuentra todo el contenido disponible de la red.

Es necesario desarrollar una herramienta que permita encontrar todo el contenido de la red. Motivo por el cual este proyecto puede solucionar uno de los problemas actuales de las aplicaciones que implementan el protocolo BitTorrent.

3. Análisis del marco legal y ético

Surge la duda sobre la legalidad de BitTorrent. Esta cuestión sobre la legalidad de las aplicaciones que implementan el protocolo BitTorrent siempre ha sido un tema de debate muy controvertido, debido a que en internet no existe ninguna identidad u organismo que pueda regular el contenido que se puede encontrar en su red.

El objetivo del protocolo es la de crear un medio para poder compartir archivos a través de la red de una forma descentralizada y eficiente. El problema ético y legal empieza cuando los usuarios hacen uso del protocolo para descargar contenido ilegal o protegido por derechos de autor, siendo por estas actuaciones cuando el protocolo se convierte en el punto de mira. El comportamiento de los usuarios es lo que ha creado la idea de que el protocolo es ilegal, cuando en realidad lo que es ilegal es la acción que realiza el usuario al descargar ese tipo de contenido de la red.

Un caso parecido es el del navegador Tor [24], que permite al usuario obtener una mayor privacidad y anonimato cuando se navega por la red. Lo negativo empieza cuando los hackers o ciber-delincuentes hacen uso de esta herramienta para realizar ataques a través de la red. Como es una herramienta que usan con dudosas intenciones por parte de algunos usuarios, es considerada una herramienta que facilita los ataques. Esto provoca que su uso se asocie a un comportamiento negativo. Se está culpando a un navegador web por las acciones de un grupo de personas.

Lo mismo sucede con BitTorrent, se asocia el uso ilegal que hacen los usuarios de las aplicaciones que implementan el protocolo. Por esto, se reafirma en la legalidad del protocolo, siendo ilegal la acción que realiza el usuario al descargar contenido protegido. Se trata de “penalizar” al usuario por el uso incorrecto de la herramienta y no a la herramienta misma.

Por otro lado, hay que dejar claro que el proyecto en ningún momento descarga contenido protegido por derechos de autor, únicamente obtiene los metadatos de los archivos. Los datos que obtiene la aplicación están formados por el nombre y el tamaño del archivo, y esta información se almacena de forma local en el ordenador, que en ningún momento es compartida en internet, es información exclusiva del usuario.

Con la finalidad de que los usuarios puedan descargarse contenido que no esté protegido por derechos de autor, han aparecido varios portales web en los que se puede encontrar contenido que no está protegido. Normalmente, se trata de programas de código abierto o contenido libre y gratuito, por lo que si un usuario descarga este tipo de archivos está obteniendo contenido completamente legal.

4. Introducción teórica a BitTorrent

En este apartado se va a realizar una introducción a conceptos teóricos relacionados con la arquitectura y diseño de BitTorrent, necesarios antes de profundizar en el estudio del protocolo BitTorrent, con el objetivo de sentar las bases previas de conocimiento del desarrollo del proyecto.

4.1 Aplicaciones de red

Las aplicaciones de red son programas que se ejecutan en distintos terminales o hosts y que se comunican entre sí haciendo uso de la red. Los encargados de desarrollar dichas aplicaciones utilizan una de las dos arquitecturas predominantes en las aplicaciones de red actuales: la arquitectura cliente-servidor o la arquitectura P2P.

4.1.1 Arquitectura P2P

En una arquitectura cliente-servidor siempre existe un elemento activo, denominado servidor. El servidor es el elemento central de esta arquitectura, encargado de dar servicio a las peticiones de otros hosts, conocidos como clientes. En esta arquitectura, los clientes no se pueden comunicarse directamente entre ellos y en la mayoría de situaciones, un solo servidor es incapaz de dar servicio a todas las solicitudes de sus clientes, por lo que se suele recurrir a la agrupación de servidores para crear un servidor virtual capaz de responder a todas las peticiones.

La comunicación entre el cliente y el servidor se realiza mediante el intercambio de mensajes a través de la red. En aplicaciones de compartición de archivos, el cliente solicita al servidor un archivo y este es el encargado de enviársela a través de la red. Cada cliente recibe una copia del archivo que solicita al servidor por lo que, si hay muchos usuarios realizando peticiones al servidor, el tiempo de recepción del archivo en el cliente puede aumentar debido a que el servidor dispone de un ancho de banda que no puede superar.

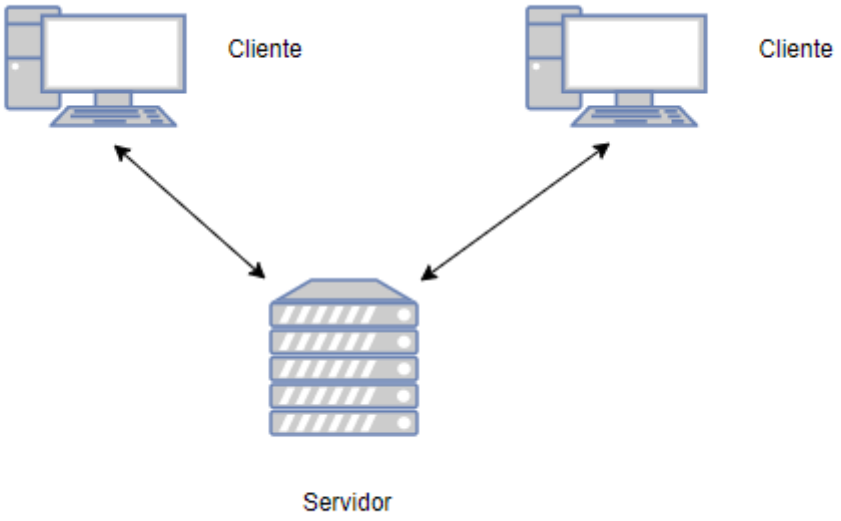

Figura 1. Arquitectura cliente-servidor

Las aplicaciones que usan esta arquitectura normalmente necesitan una gran infraestructura y hacer frente a los costes de las interconexiones y al ancho de banda de los servidores para enviar y recibir los archivos a través de Internet.

Esta arquitectura no es la mejor opción para las aplicaciones de red dedicadas a la compartición de archivos por los sobrecostes en el aumento del número de servidores. Además de este inconveniente, el ancho de banda del servidor está limitado ya que, estos cuentan con un ancho de banda concreto y no lo pueden superar para atender a las peticiones de los clientes.

4.1.2 Arquitectura P2P

La arquitectura P2P elimina el concepto de servidor central propio de la arquitectura cliente-servidor. Para lograr esto, cada host funciona al mismo tiempo como cliente y como servidor, permitiendo que cada host pueda redistribuir cualquier parte del archivo que ha recibido a cualquier otro host, y de esta manera, ayuda al host servidor en el proceso de distribución del archivo.

En el modelo cliente-servidor, el servidor tiene que mandar una copia del archivo a cada uno de los clientes, dando lugar a una gran sobrecarga en el servidor y un consumo enorme de su ancho de banda. Este problema se soluciona con la arquitectura P2P, ya que cada host actúa como cliente y como servidor.

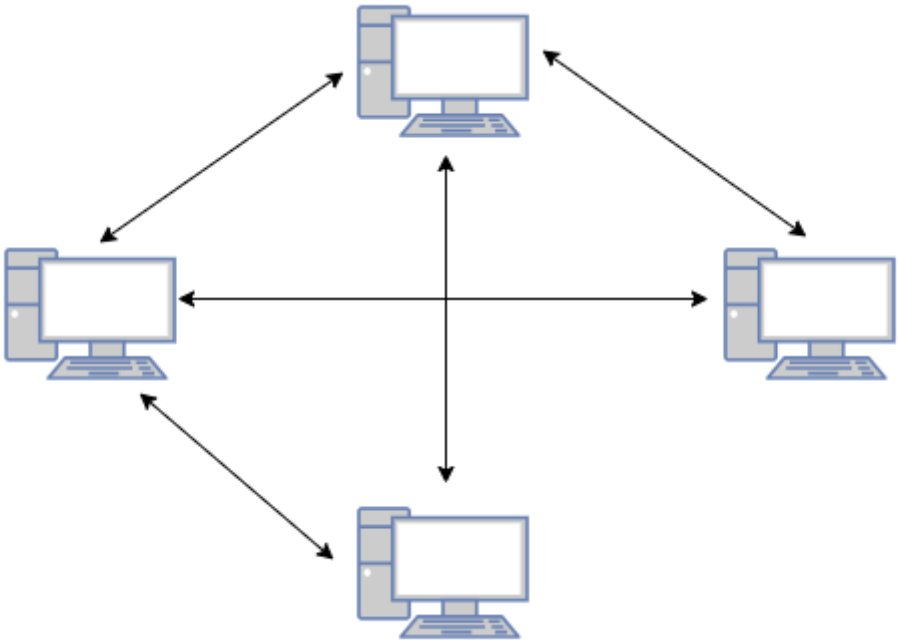

Figura 2. Arquitectura P2P

En esta arquitectura la comunicación se realiza entre hosts sin la necesidad de un elemento central. Esto permite reducir retardos en la compartición de archivos, y al mismo tiempo, hace que la principal ventaja sea la auto escalabilidad, ya que los hosts actúan como redistribuidores y consumidores de contenido. En el libro Kurose[16], se hace una comparación entre ambas arquitecturas en la distribución de un fichero en una red conforme aumenta el número de clientes. Dicha comparación demuestra la ventaja de la arquitectura P2P sobre la cliente-servidor, siendo la primera la más adecuada para las aplicaciones de compartición de archivos en red.

4.2 Redes P2P

Las redes P2P aparecieron por primera vez en la década de los años 90. Son redes resultado de desarrollar aplicaciones con la arquitectura P2P. A lo largo de su existencia, se han usado para varios propósitos como la telefonía VoIP, para hacer Streaming en tiempo real o la compartición de contenido multimedia. El uso más común en la actualidad es la compartición de archivos multimedia.

Una de las primeras aplicaciones de red con arquitectura P2P fue Napster [8], creada por Shawn Fanning en 1999. Napster era una aplicación que permitía la distribución de archivos de música en formato MP3. La red que creó esta aplicación, no era descentralizada puesto que, usaba su servidor central para localizar archivos y usuarios. La arquitectura de Napster tenía un problema, si su servidor central se detenía o se

cerraba, su servicio se vería interrumpido. Así fue como se cerró Napster después de ser denunciada.

Las redes P2P se pueden clasificar en dos categorías principales siendo estas la centralizada y la descentralizada.

4.2.1 Redes P2P centralizadas

Este tipo de red cuenta con uno o más servidores para crear un único servidor virtual lo suficientemente potente para que todos los hosts puedan ser atendidos. El servidor central sirve para localizar recursos en la red. Por otro lado, la transferencia de archivos se hace directamente entre los pares, sin la intervención del servidor.

Esta característica de un elemento central, convierte al servidor en un elemento del que depende todo el sistema y, por tanto, este servidor es el principal punto de fallo. Si el servidor deja de funcionar, el servicio de la aplicación se verá interrumpido hasta que se recupere el funcionamiento del servidor central.

Este tipo de redes cuentan con un problema de escalabilidad, el cual se produce cuando el número de hosts aumenta notablemente y se produce un cuello de botella en el servidor, pudiendo afectar a la capacidad de dar servicio a los pares.

Un ejemplo de este tipo de redes sería la red de la aplicación Napster [8] en la que su servidor principal servía para encontrar archivos y usuarios en la red.

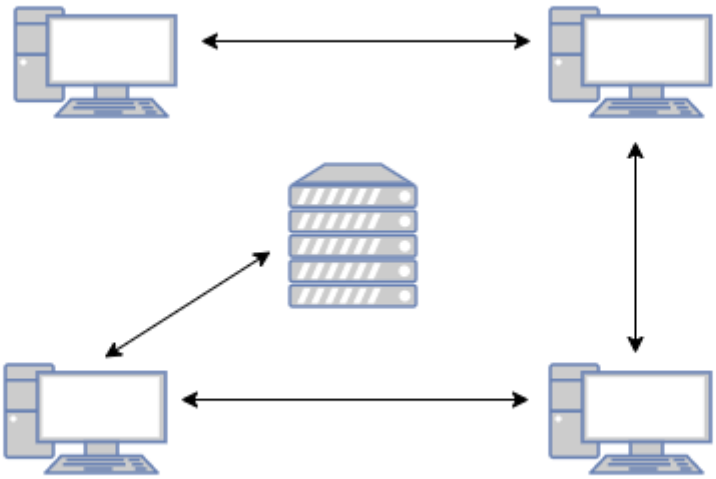

Figura 3. Arquitectura P2P centralizada

4.2.2 Redes P2P descentralizadas

Este tipo de red, también conocida como red P2P pura, no cuenta con ningún servidor para localizar recursos y usuarios en la red. Todos los hosts que participan en ella son iguales, hacen los papeles de cliente y servidor. Los hosts tienen una visión parcial de la red y de los recursos que se están compartiendo en ella.

Cuando los hosts necesitan buscar un archivo en la red, tienen que ir preguntando a los nodos que participan en la red para encontrar el archivo, lo que puede provocar una inundación de mensajes en la red conocida como flooding [1]. En algunos casos, estas búsquedas no logran encontrar los recursos en la red. Hay aplicaciones que implementan una tabla hash distribuida para localizar los recursos de la red de una forma más eficiente con el fin de evitar la inundación de mensajes en la red.

La principal ventaja es que son más robustas y escalables que las centralizadas porque no dependen de un servidor central para encontrar recursos y usuarios. En general, presenta mejores prestaciones que la centralizada.

Un ejemplo de aplicación que usa una red P2P pura o descentralizada es BitTorrent [2] cuando hace uso de la tabla hash distribuida mientras que, si hace uso de elementos centrales como los *trackers*, se puede entender que es centralizada.

5. Protocolo BitTorrent

En este apartado se va a realizar una explicación en profundidad de los conocimientos teóricos más importantes del protocolo BitTorrent. El objetivo es entender el funcionamiento del protocolo para poder desarrollar una aplicación completamente funcional que cumpla los objetivos especificados.

5.1 Protocolo

El protocolo BitTorrent fue creado en el año 2001 por Bram Cohem. Es un protocolo P2P diseñado para la distribución de archivos entre pares a través de Internet y se ha convertido en uno de los protocolos más usados para la trasferencia de archivos.

El protocolo permite que los pares participen en la distribución de los archivos, haciendo los papeles de cliente y de servidor para así, reducir el impacto que tendría en una arquitectura cliente-servidor al descargar todos los pares del mismo servidor. Por lo tanto, es una alternativa al modelo basado en servidores.

Para poder compartir archivos empleando este protocolo es necesario disponer de un cliente BitTorrent. Un cliente BitTorrent es una aplicación de red que implementa el protocolo BitTorrent y que por lo tanto se puede unir a dicha red. En la actualidad, existen numerosos clientes BitTorrent como por ejemplo BitTorrent [2], µTorrent [28], Vuze[29] o Transmission[26], siendo la inmensa mayoría de ellos gratuitos.

Cuando un par quiere descargar un archivo que se está compartiendo en la red, lo hacen descargando fragmentos del archivo, siendo el tamaño típico de 256 KBytes. Cada fragmento de un archivo recibe el nombre de pieza. Cuando un par comienza la descarga de un archivo, no dispone de ningún fragmento y, conforme avanza el tiempo, va obteniendo más fragmentos que puede redistribuir a otros pares que estén compartiendo el mismo archivo. Cuando un par quiere descargar fragmentos de un archivo, utiliza una técnica conocida como primero el menos común. La idea de esta técnica es determinar qué fragmentos no tiene el par y al mismo tiempo son los menos comunes entre los pares de los que está obteniendo fragmentos.

A diferencia de otras redes de intercambio, BitTorrent no dispone de un método de búsqueda de archivos para que después los pares puedan descargarlos y los usuarios de este protocolo tienen que buscarlos por sus propios medios. Normalmente, hay páginas web que publican los archivos disponibles para descargar, por lo que el objetivo de este TFG es el desarrollo de una herramienta que permita encontrar los archivos disponibles en la red.

5.2 Actores en la red BitTorrent

En la red de BitTorrent podemos encontrar los siguientes actores:

- *Peers:* Es el nombre que reciben todos los usuarios que están en la red. También son conocidos como pares.
- *Leechers:* Es el nombre que reciben todos los *peers* que están descargando un archivo y que todavía no tienen el archivo descargado completamente, es decir, no tienen todas las piezas o fragmentos del archivo.
- *Seeders:* Son los pares de la red que poseen todos los fragmentos de un archivo. Por cada archivo que se comparte en la red, debe haber un par que tenga el archivo completo para que el resto de pares se lo puedan descargar.
- *Trackers:* También conocidos como rastreadores, son servidores especiales que permiten que los pares se comuniquen unos con otros. Sirven para localizar a otros pares que tienen piezas del archivo que se está compartiendo. Los *trackers* siguen la pista de los pares que están participando en la compartición de un archivo.

5.3 BEP

BEP es un término inglés que significa propuesta de mejora de BitTorrent. Un BEP es un documento de diseño que informa de las características del protocolo BitTorrent o propone una nueva característica a modo de mejora o ampliación del protocolo. La idea de este documento es que pueda servir como un mecanismo para documentar el protocolo y proponer mejoras. Todos los documentos BEP se encuentran almacenados en un repositorio de GitHub [3].

Cuando se presenta una nueva propuesta, se le asigna un número de BEP, con la consecuente actualización del índice de los documentos BEP, siendo el BEP_0000 [10] el encargado de indexar todas las propuestas. Cada documento tiene un número que lo identifica y, además, dicho número nunca cambia. El historial de cualquier documento se encuentra almacenado en el repositorio de GitHub.

Las propuestas de mejora empiezan primero siendo borradores, pudiendo promocionar a aceptadas y finalmente al estado final. Los BEP en el estado aceptado, describen características que se han implementado en alguna aplicación cliente de BitTorrent y que han demostrado ser de utilidad. El estado final es que se ha aceptado como estándar y parte del protocolo. Con el tiempo también se puede reemplazar alguna característica del BEP como, por ejemplo, al realizar una mejora en el algoritmo del BEP original seguiría siendo el mismo BEP pero reemplazando parte de su contenido.

Un ejemplo de propuesta que se encuentra en estado de aceptada es el protocolo DHT [18] que está implementado en numerosas aplicaciones del protocolo BitTorrent.

5.4 Bencoding

Los mensajes que se mandan entre los participantes de la red BitTorrent están codificados, empleando para ello un método de codificación conocido como Bencoding [23]. Esta codificación admite cuatro tipos de datos, números enteros, cadenas de texto, listas y diccionarios, estando completamente detallado en la propuesta BEP_0003 [4]

- Los enteros se codifican mediante una "i" seguida del número en base decimal y la letra "e". Esta codificación permite representar números negativos. Los números en base decimal no pueden empezar con un cero en la izquierda salvo la codificación del número cero, mientras que los enteros no tienen una limitación de tamaño en lo que a dígitos se refiere. (Ejemplos de números válidos: *i5e, i-5e, i0e*. Ejemplos de números no válidos: *i05e, i-0e).*

- Las cadenas se codifican con la longitud de la propia cadena en base decimal seguido de dos puntos y la cadena. (Por ejemplo, *4:casa*, representa una cadena de longitud cuatro que es casa).

- Las listas se codifican con la letra "l" seguida de sus elementos codificados y al final del último elemento se añade la letra "e" para indicar el final de la lista. (Un ejemplo es la lista *["lunes","martes"]* se representa como *l4:lunes4:martese).*

- Los diccionarios se codifican con la letra inicial "d" seguida de una lista de claves y sus valores correspondientes con la letra "e" para indicar el final del diccionario. Las claves del diccionario tienen que ser cadenas. (Ejemplo: el diccionario *{"semana":2,"mes":"enero"}* se representa como *d3:mes5:enero6:semanai2ee).*

Esta codificación forma parte del estándar del protocolo y, por ello, todas las aplicaciones que emplean el protocolo BitTorrent implementan un codificador y decodificador para su correcto funcionamiento. En internet [23] hay varias implementaciones que permiten a los desarrolladores comprobar los mensajes que procesan sus aplicaciones.

5.5 Archivos de meta información

Estos archivos también son conocidos como archivos .torrent o torrents. Son diccionarios codificados con bencoding. Estos ficheros son importantes porque contienen información detallada sobre un archivo que se está compartiendo en la red. Todas las cadenas del archivo, que contienen texto, deben estar codificadas en UTF-8. Estos diccionarios cuentan con las siguientes claves:

- La clave *announce* almacena el valor de la URL del *tracker* principal.
- La clave *announce-list* es opcional y su valor es una lista de cadenas que contiene las URL de trackers auxiliares al principal. Esta clave se añadió en el BEP_0012 [12].
- La clave *info* es un diccionario que contiene información sobre el archivo que se está compartiendo. Contiene las siguientes claves:
  - *name*: Es una cadena de texto que contiene el nombre sugerido para guardar el archivo compartido.
  - *piece length*: Es un entero que representa el número de bytes en cada pieza de las que se ha dividido el archivo.
  - *pieces:* Es una cadena que concatena el resultado de aplicar una función hash SHA1 sobre cada una de las piezas.
  - *length* o *files:* Se utiliza una de las dos claves, pero nunca las dos al mismo tiempo. Se emplea la clave *length* cuando el archivo torrent contiene únicamente un fichero a compartir. En este caso, el valor de la clave *length* es un entero que representa el tamaño del archivo en bytes. En el caso de que el archivo torrent contenga más de un fichero, se usa la clave *files* y su valor es una lista de diccionarios que contiene por cada fichero una clave *length* con el valor del tamaño del fichero y la clave *path* con la ruta del fichero en formato UTF-8.

Cabe destacar que las URL de los *trackers* que figuran en el archivo torrent sirven para que los pares se conecten a ellos y puedan encontrar a más pares que estén descargando el archivo.

5.6 DHT

Como se ha explicado previamente, la red de BitTorrent no es descentralizada, ya que cuenta con elementos centrales conocidos como trackers que se usan para poner en contacto a los pares.

El gran inconveniente de esto es que, si los *trackers* dejan de funcionar, la propia red dejaría de funcionar ante la imposibilidad de poder descubrir pares con los que obtener y compartir la información.

Para solucionar esto, BitTorrent implementa una tabla hash distribuida (DHT) para almacenar información de contacto de los pares de la red para los archivos torrent que no emplean *trackers*. Este nuevo protocolo, detallado en el BEP_0005 [18], está basado en Kademlia [15] e implementado sobre UDP (a diferencia de los clientes que normalmente usan TCP.) A efectos prácticos, cada par de la red se convierte en un *tracker*. Ahora los pares pueden hacer la función de cliente, servidor y de *tracker*.

Para evitar futuras confusiones hay que tener en cuenta que el término “par” (*peer*) hace referencia a un cliente o servidor funcionando en un puerto TCP que implementa el protocolo BitTorrent. Y el término “nodo” hace referencia a un cliente o servidor funcionando en un puerto UDP que implementa la DHT.

5.6.1Funcionamiento de la DHT

La DHT está formada por nodos que almacenan información de localización y contacto de los pares. Cada aplicación cliente de BitTorrent incorpora un nodo DHT, el cual es usado para contactar con otros nodos de la DHT, permitiendo obtener la información de contacto de los pares de los que se puede obtener el archivo.

Cada nodo tiene un identificador único asignado, el cual ha sido asignado por el mismo nodo, haciendo uso de una función hash conocida como SHA1 [21]. El identificador del nodo está formado por 160 bits, el mismo número de bits que el espacio de los infohashes, siendo estos los identificadores únicos de los archivos que se comparten en la red. Este infohash [7] es el resultado de aplicar la función hash SHA1 sobre la clave *info* de los archivos torrent. Es por esto que podemos identificar a los archivos de la red y a los nodos con hashes.

La DHT utiliza la distancia métrica para comparar dos identificadores con el objetivo de calcular su proximidad. Los nodos deben mantener una tabla de enrutamiento con la información de contacto de un grupo reducido de nodos. La idea es que almacene muchos nodos con un identificador cercano al suyo, conocidos como vecinos, y unos pocos con un identificador lejano al suyo.

En Kademlia, la distancia métrica se calcula con una XOR de los hashes y el resultado es un entero sin signo. Cuanto más pequeño sea el número significa que están más cerca uno del otro.

Cuando un nodo intenta buscar información de contacto de los pares que están compartiendo un determinado archivo, primero comunicará con sus vecinos para obtener información.

En el momento que un nodo necesita la información de contacto de los pares de un torrent, usa la distancia métrica para comparar el infohash del torrent con cada uno de los nodos de su tabla de enrutamiento. Después, contacta con los nodos de su tabla más cercanos al infohash del torrent y pregunta por la información de contacto de los pares que están compartiendo el archivo del torrent. Si el nodo que recibe la petición conoce a pares del torrent, responde con la información de contacto de dichos pares. En caso contrario, el nodo responde con la información de contacto de los nodos de su tabla de enrutamiento más cercanos al infohash del torrent. El nodo original repite este proceso hasta que no pueda encontrar nodos más cercanos.

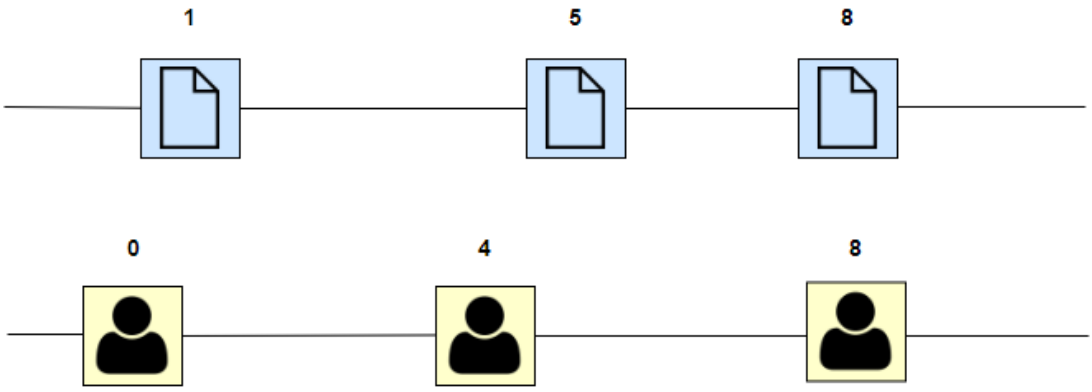

Figura 4. Relación entre los hashes de los archivos y los nodos

5.6.2 Protocolo KRPC

El protocolo KRPC [18] es un mecanismo RPC sencillo que consiste en el envío de diccionarios codificados con bencodig sobre UDP. En el intercambio de mensajes, únicamente se envía un paquete de consulta y se recibe un único paquete de respuesta. En este protocolo existen tres tipos de mensajes: consultas, respuestas y errores.

Un mensaje del protocolo está formado por un diccionario con tres claves fijas en todos los mensajes y con claves adicionales en función del tipo de mensaje. Las claves comunes son:

- La clave *t* es una cadena que representa el identificador de la transacción. Dicho identificador es generado por el nodo que hace la consulta y es devuelto en el mensaje de respuesta. Esta clave sirve para que exista una relación entre el mensaje de consulta y el de respuesta, es decir, para saber a qué mensaje de consulta hace referencia la respuesta recibida.
- La clave *y* es un carácter cuyo valor indica el tipo de mensaje. El valor "q" indica que es una consulta, el valor "r" indica que es una respuesta y el valor "e" nos informa de que es un mensaje de error.
- La clave *v* es una cadena de caracteres que informa de la versión de la aplicación cliente. Se usan dos caracteres para identificar al cliente según está especificado en el BEP_0020 [11] seguido de dos caracteres con el identificador de la versión.

5.6.3 Información de contacto

La información de contacto de los pares está formada por la dirección IP y el número del puerto TCP. Dicha información está codificada como una cadena de texto de seis bytes, destinando los primeros cuatro bytes a la dirección IP y los dos últimos al puerto TCP. Esta información también se conoce como dirección compacta.

La información de contacto de los nodos de la DHT está codificada como una cadena de texto de 26 bytes. En este caso, los primeros 20 bytes corresponden al identificador del nodo en la red seguido de cuatro bytes para la dirección IP y 2 bytes para el puerto UDP. Esta información se conoce como información compacta del nodo.

5.6.4 Tipos de mensajes

En este protocolo contamos con tres tipos de mensajes que son los mensajes de consulta, de respuesta y de error.

- Los mensajes de consulta contienen dos claves adicionales. La clave *q* es una cadena de texto que especifica el nombre del método de la consulta a realizar. La clave *a* es un diccionario que contiene los argumentos necesarios del método. Cada argumento tiene su clave y su valor.
- Los mensajes de respuesta contienen una única clave adicional. La clave *r* es un diccionario que contiene los valores que retorna la función. Para cada valor tiene una clave asociada. Los mensajes de respuesta se mandan únicamente si el mensaje de consulta ha tenido éxito.
- Los mensajes de error contienen una clave adicional. La clave *e* es una lista en la que el primer elemento es un número entero que representa el código del error [18]. El segundo elemento es una cadena de texto con una descripción del error. Los mensajes de error se envían cuando no se ha podido completar con éxito una consulta.

```
Mensaje de error: {"t":"ab", "y":"e", "e":[204,"Method Unknown"]}
Mensaje codificado: d1:eli204e14:Method Unknowne1:t2:ab1:y1:ee
```

Figura 5. Ejemplo de mensaje de error

5.6.5 Tipos de consultas en la DHT

En la DHT se pueden realizar cuatro tipos de consultas. Son las consultas de *ping*, *find_node*, *get_peers y announce_peer*.

Ping: es la consulta más simple y su función es comprobar si un nodo está conectado y en funcionamiento. El valor de la clave *q* es *ping*. Esta consulta tiene un argumento que es el identificador del nodo que hace la consulta. Este argumento se representa mediante la clave *id* y su valor es una cadena de texto de 20 bytes.

```
Consulta de ping: {"t":"ac", "y":"q", "q":"ping", "a":{"id":"anc8dbfgteism584sf5q"}}
Consulta codificada: d1:ad2:id20:anc8dbfgteism584sf5qe1:q4:ping1:t2:ac1:y1:qe
Respuesta de ping: {"t":"ac", "y":"r", "r": {"id":"jdtasdna45bsd8sa5q32"}}
Respuesta codificada: d1:rd2:id20:jdtasdna45bsd8sa5q32e1:t2:ac1:y1:re
```

Figura 6. Consulta de tipo *ping*

Find_node: esta consulta se usa para obtener la información de contacto de un nodo dado su identificador. Tiene dos argumentos, el argumento *id* contiene el identificador del nodo que hace la consulta y el argumento *target* que contiene el identificador del nodo del que se quiere obtener información. Cuando un nodo de la red recibe esta consulta, debe responder con el argumento *nodes* cuyo valor es una cadena de texto con la información de contacto del nodo objetivo o de *n* nodos cercanos al objetivo que tenga en su tabla de enrutamiento.

```
Consulta de find_node: {"t":"aa", "y":"q", "q":"find_node", "a": {"id":"anc8dbfgteism584sf5q",
                       "target":"jdtasdna45bsd8sa5q32"}}
Consulta codificada: d1:ad2:id20:anc8dbfgteism584sf5q6:target20:jdtasdna45bsd8sa5q32e1:q9:
                     find_node1:t2:aa1:y1:qe
Respuesta de find_node: {"t":"aa", "y":"r", "r": {"id":"dsf5df61ew98f23a221f", "nodes":
                        "azd486..."}}
Respuesta codificada: d1:rd2:id20:dsf5df61ew98f23a221f5:nodes9:azd486...e1:t2:aa1:y1:re
```

Figura 7. Consulta de tipo *find_node*

Get_peers: esta consulta permite obtener los pares que están compartiendo un archivo torrent identificado por su infohash. Tiene dos argumentos. El argumento *id* contiene el identificador del nodo que hace la consulta y el argumento *infohash* contiene el identificador del torrent. Si el nodo que recibe la consulta conoce a pares de ese infohash, contesta con una clave *values* cuyo valor es una lista de cadenas de caracteres con la información de contacto compacta de los pares. Cada elemento de la cadena tiene la información de un par. En caso contrario, si el nodo que recibe la consulta no conoce a pares para ese infohash, devuelve la clave *nodes* con *n* nodos cercanos a ese infohash que tiene el nodo consultado en su tabla de enrutamiento. En cualquiera de las dos situaciones, el nodo consultado devuelve una clave *token* para una futura consulta de *announce_peer*.

```
Consulta de get_peers: {"t":"bg", "y":"q", "q":"get_peers", "a":{"id": "abcdefghij0123456789",
                       "info_hash": "dsf5df61ew98f23a221f"}}
Consulta codificada: d1:ad2:id20:abcdefghij01234567899:info_hash20:dsf5df61ew98f23a221fe1:q9:
                     get_peers1:t2:bg1:y1:qe
Respuesta con pares: {"t":"bg", "y":"r", "r":{"id": "abcdefghij0123456789", "token": "aoeusnth",
                     "values": ["axjegu","idhtnm"]}}
Respuesta codificada: d1:rd2:id20:abcdefghij01234567895:token8:aoeusnth6:values16:axjegu6:idhtnme
                     e1:t2:bg1:y1:re
Respuesta con nodos cercanos: {"t":"bg", "y":"r", "r":{"id":"abcdefghij0123456789", "nodes":
                              "def456...","token":"aoeusnth"}}
Respuesta codificada: d1:rd2:id20:abcdefghij01234567895:nodes9:def456...5:token8:aoeusnthe1
                     :t2:bg1:y1:re
```

Figura 8. Consulta de tipo *get_peers*

*Announce*_peers: sirve para avisar de que el par que controla al nodo que hace la consulta está descargando el torrent en un puerto. Esta consulta tiene cuatro argumentos. El argumento *id* con el identificador del nodo que hace la consulta, el *infohash* que contiene el infohash del torrent, el *port* que contiene el número del puerto y el *token* recibido en la respuesta previa de un get_peers. Hay un argumento opcional conocido como *implied_port* cuyo valor es el uno si el servidor debe omitir el puerto del parámetro *port* y utilizar el puerto de origen del datagrama UDP. Este parámetro es muy útil para los pares que se encuentran detrás de un NAT.

```
Consulta announce_peers: {"t": "be", "y": "q", "q": "announce_peer", "a": {"id": "anc8dbfgteism
                         584sf5q", "implied_port": 1, "info_hash": "dsf5df61ew98f23a221f", "por
                         t": 6881, "token": "aoeusnth"}}
Consulta codificada: d1:ad2:id20:anc8dbfgteism584sf5q12:implied_porti1e9:info_hash20:dsf5df61ew
                     98f23a221f4:porti6881e5:token8:aoeusnthe1:q13:announce_peer1:t2:be1:y1:qe
Respuesta announce_peers: {"t": "be", "y": "r", "r": {"id": "dsf5df61ew98f23a221f"}}
Respuesta codificada: d1:rd2:id20:dsf5df61ew98f23a221fe1:t2:be1:y1:re
```

Figura 9. Consulta del tipo *announce_peer*

5.7 Implementaciones de BitTorrent

A lo largo del tiempo se han ido desarrollando e implementando un gran número de aplicaciones cliente y bibliotecas de BitTorrent. A continuación, se enumeran las más usadas.

5.7.1 Aplicaciones cliente

- BitTorrent [2]
- µTorrent [28]
- Transmission [26]
- Vuze [29]

5.7.2 Bibliotecas

- Libtorrent [17]
- Ttorrent [27]

6. Análisis y diseño de la aplicación

En este apartado se va a realizar un análisis de los requisitos de la aplicación para que se puedan satisfacer los objetivos expuestos con un diseño adecuado.

6.1 Planificación inicial

Para desarrollar una aplicación que cumpla con los objetivos mencionados previamente, se necesita un elemento que busque contenido en la red de BitTorrent y otro elemento que sea capaz de buscar información detallada sobre el contenido encontrado. A todo esto, hay que añadir una base de datos que permita el intercambio de información entre los dos elementos y el almacenaje de la información de forma persistente.

Por ello, este proyecto está formado por dos elementos principales separados. Cada uno con una finalidad distinta pero complementaria.

En primer lugar, está el componente encargado de buscar contenido mediante la exploración de la red de BitTorrent para intentar descubrir todo el contenido que se está compartiendo en dicha red. Todo el contenido que descubra se almacenará en una base de datos para su posterior procesado por el segundo componente. En segundo lugar, está el buscador de nombres, encargado de procesar el contenido descubierto por el primer componente y obtener información detallada para cada uno de los archivos encontrados. Después de encontrar dicha información, deberá almacenarla de nuevo en la base de datos.

Debido a que los dos componentes requieren una base de datos para almacenar información y procesarla, se ha optado por un diseño de componentes que incluya una base de datos principal a la que ambos componentes puedan acceder.

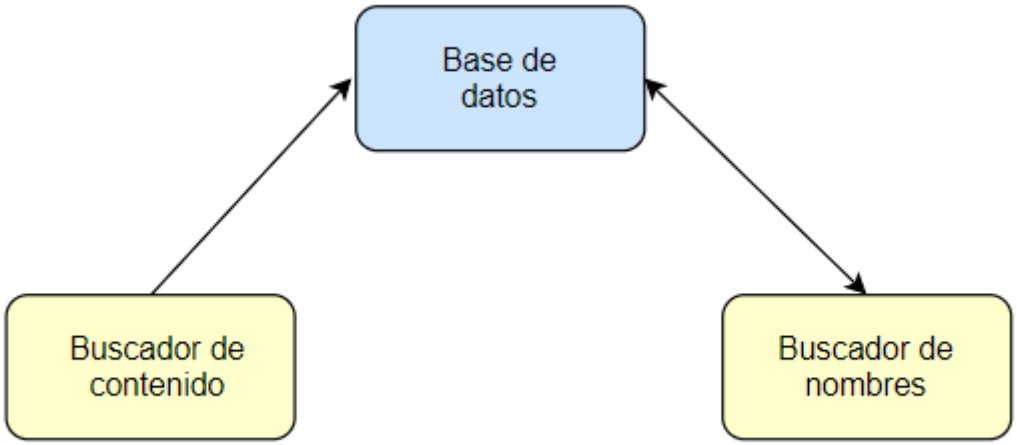

Figura 10. Diseño de la arquitectura de la aplicación

Como se puede apreciar en la figura 10, la relación entre la base de datos y el buscador de contenido es unidireccional. Esto es debido a que el buscador únicamente almacena datos en la base de datos y nunca hace consultas para obtener información de ella. Por el contrario, la relación de la base de datos con el buscador de nombres es bidireccional ya que, debe obtener información sobre los archivos encontrados, procesar la información obtenida y buscar información sobre los archivos para su posterior inserción en la base de datos.

6.2 Buscador de contenido

En la actualidad, tanto el protocolo de BitTorrent como el protocolo de la DHT no tienen ninguna implementación o mejora que permita descubrir contenido en la red. Esto ha provocado que aparezcan páginas web en las que se puede encontrar contenido que se está compartiendo en la red. Estas páginas muestran una pequeña parte de todo el contenido que realmente está disponible.

La comunidad de desarrolladores es conocedora de este problema, y es por ello que existe una propuesta de mejora [6] que permitiría a los nodos de la DHT obtener una serie de muestras de infohashes que otros nodos tienen almacenados. Esta mejora permitiría recorrer todos los nodos de la DHT para obtener los archivos almacenados en los nodos y, por lo tanto, descubrir la totalidad del contenido disponible en la red. Sin embargo, esta propuesta está en un proceso de consideración para la estandarización, pero todavía no se ha implementado por lo que, no es una solución al problema de la búsqueda de contenido.

La posible solución es la implementación de algún componente que permita realizar un seguimiento de los mensajes que se intercambian en la red para intentar indexar el contenido encontrado en dichos mensajes.

Anteriormente, se ha explicado que los nodos de la DHT tienen una vista parcial de la red y una lista reducida de nodos cercanos también conocida como nodos vecinos. La idea es que el buscador de contenido tiene que hacer el papel de nodo de la DHT y, paulatinamente, recibir información de otros nodos sobre los archivos de la red. Esta idea presenta un problema de rendimiento porque se obtendría información de un grupo reducido de nodos. Al estar limitando el rendimiento de la aplicación, habrá que buscar una forma de obtener información sobre un número mayor de nodos.

6.2.1 Algoritmo

Dos investigadores de la universidad de Helsinki realizaron una investigación publicada bajo el nombre “Real-World Sybil Attacks in BitTorrent Mainline DHT” [30] sobre los ataques en la DHT de BitTorrent. En dicha investigación se habla del avance tecnológico que han supuesto las versiones de los clientes BitTorrent con la tabla hash distribuida. Esta variante ha conseguido que el papel del *tracker* no sea un elemento tan crítico en la red, al conseguir que los pares hagan la función de un *tracker*. Esto es una ventaja puesto que, si los *trackers* de la red dejasen de funcionar, la red podría seguir funcionando, ya que los *trackers* estarían sustituidos por los propios pares. No obstante, esta nueva versión distribuida incorpora algunos problemas de seguridad. En la investigación explica dichos problemas de seguridad de una forma bastante detallada.

Cabe destacar que la red P2P de BitTorrent puede resistir a muchos de los ataques que menciona la investigación gracias a la redundancia de datos existente entre los nodos.

Para este trabajo, nos interesa explicar un problema concreto relacionado con un ataque que se puede hacer a la red DHT. El ataque se conoce como ataque horizontal y consiste en rellenar las tablas de enrutamiento de los nodos de la red con nodos ficticios para provocar un mal funcionamiento del nodo.

La intención de la aplicación a desarrollar no es la de provocar un mal funcionamiento de los nodos de la DHT. Lo que se pretende es utilizar parte del algoritmo para obtener más información de la red sin provocar un mal funcionamiento. Para lograr esto, se va a limitar el número de mensajes que se envía a los nodos de la red y se modificará el algoritmo original.

Es un algoritmo sencillo de implementar, cuyo funcionamiento se basa en que por cada mensaje que entra en la aplicación, realiza una distinción sobre el tipo de mensaje recibido. Si ha recibido un mensaje de tipo *ping*, entonces responde con un mensaje de

pong con un identificador de nodo aleatorio. Si el mensaje es un *find_node*, responde como el propietario del identificador que busca el nodo que envía el mensaje. En el caso de recibir un mensaje de *get_peers*, guarda el infohash y no responde al nodo que envió el mensaje.

En un comportamiento normal, los nodos de la red reciben llamadas de los otros nodos, procesan los mensajes recibidos y responden con normalidad. En una situación de ataque, el nodo que está llevando a cabo el ataque, inunda la red de mensajes en los que se hace pasar por otro nodo distinto para introducirse en sus tablas de enrutamiento. También responde de forma distinta a los mensajes que recibe, llegando incluso a no responder a una gran parte de ellos.

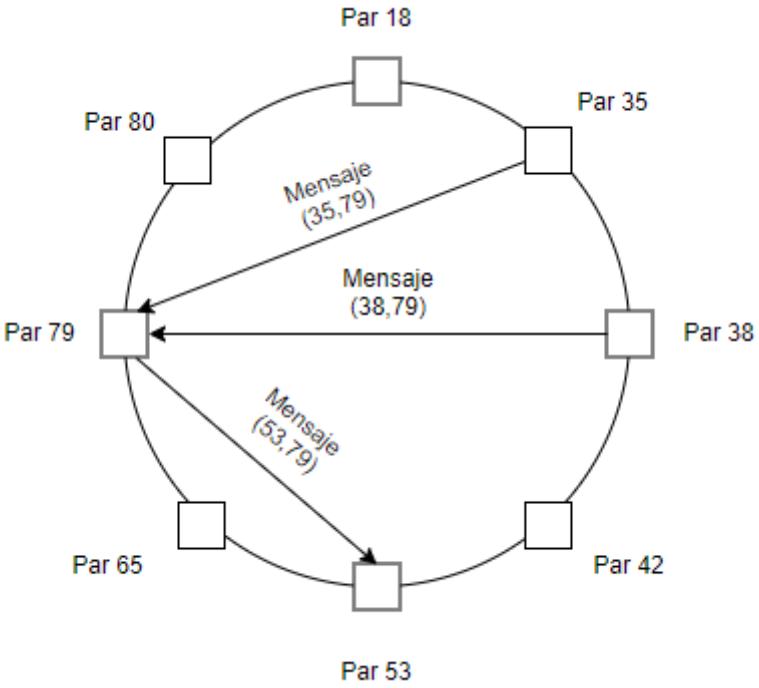

Figura 11. Comportamiento normal de un nodo en la DHT

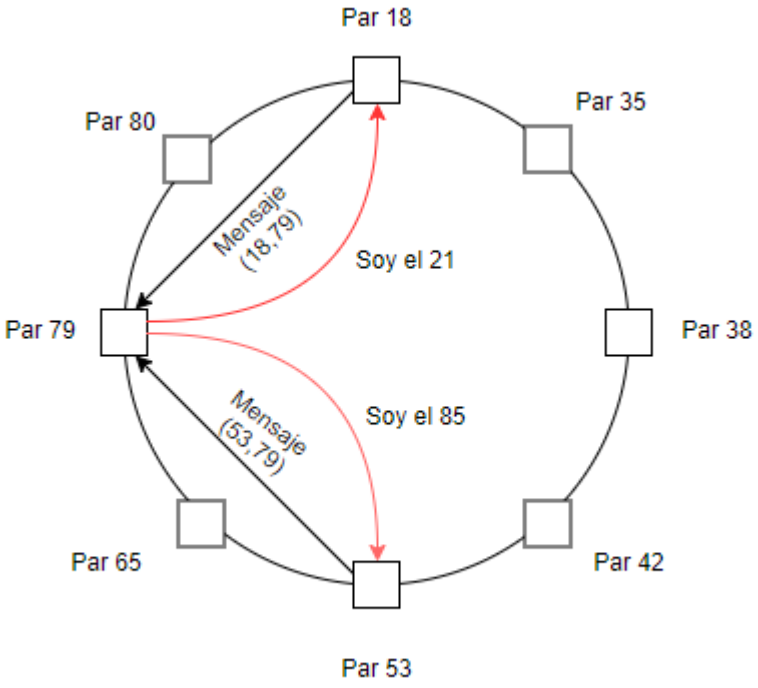

Figura 12. Comportamiento de un nodo que realiza el ataque

A pesar de recibir más información de otros nodos, no se garantiza que la información que se pueda recibir sea de gran utilidad. No obstante, este algoritmo es suficiente para desarrollar la aplicación y que el buscador de contenido tenga un buen rendimiento.

6.3 Buscador de nombres

El problema del buscador de contenido es que únicamente obtiene los identificadores de los archivos que se están compartiendo en la red, no obteniendo ningún tipo de información adicional. Por ello, este componente tiene un papel importante en la aplicación a desarrollar, al ser el encargado de obtener información detallada sobre los archivos que se han encontrado.

Es posible que no se pueda obtener información a partir de los identificadores de los archivos porque en ese momento no se estén compartiendo. No obstante, obtendrá la información de aquellos archivos que sí que se estén compartiendo en el momento de hacer la búsqueda.

La implementación de este componente es relativamente sencilla al existir una librería [5] de la aplicación WebTorrent que permite realizar consultas a los nodos de la red que participan en la DHT. Esta librería permite obtener información detallada de un archivo que se esté compartiendo a partir de su identificador único y global.

6.4 Modelo de datos

El diseño del modelo de datos de la aplicación no va a ser demasiado complejo. El uso de la base de datos es para almacenar la información básica del contenido encontrado en la red por el buscador de nombres y la información detallada de los archivos encontrados.

La tabla archivo almacenará la información básica de un archivo que se está compartiendo en la red. Concretamente, almacenará el identificador del archivo, un posible nombre y la fecha junto con la hora en la que ha sido encontrado. Además de esos datos, almacena también la dirección ip y el número de puerto del par que nos ha proporcionado la información.

La tabla info contendrá la información detallada de los archivos descubiertos, almacenando información relativa al nombre real en la red, el tamaño del propio archivo, su identificador y la fecha junto con la hora en la que se descubrió.

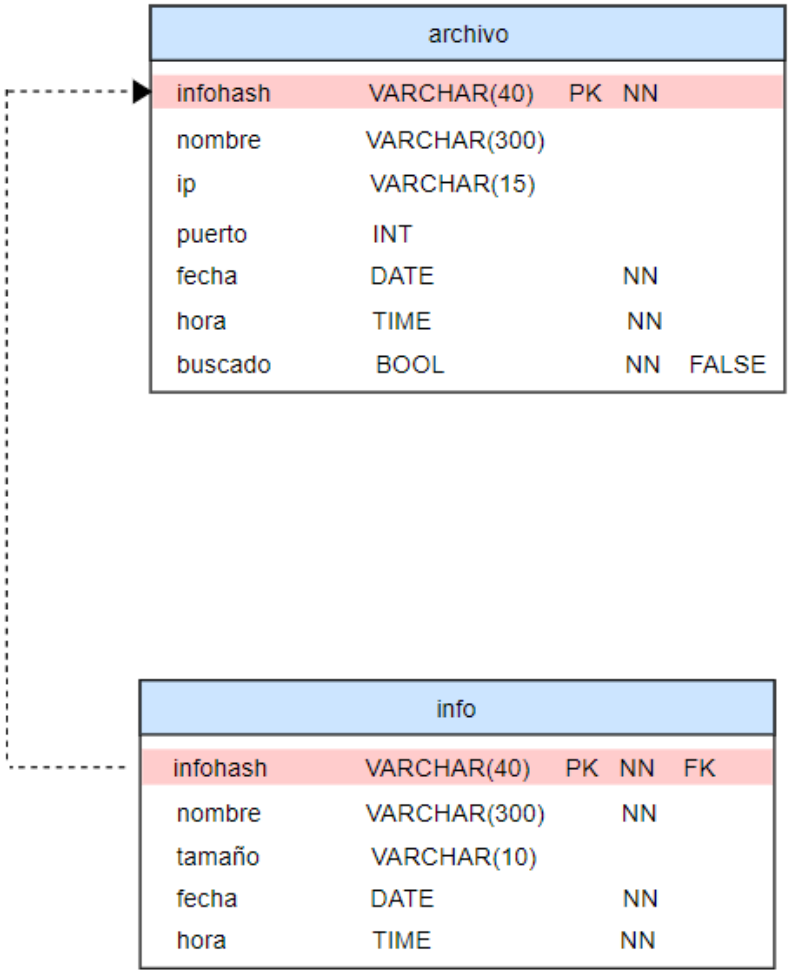

Figura 13. Modelo de datos de la base de datos relacional

7. Implementación de la aplicación

En este apartado se va a explicar el desarrollo de los componentes que conforman la aplicación desarrollada en este trabajo. Primero, se va a explicar las tecnologías empleadas para el desarrollo y después se detallarán los aspectos más importantes de cada componente.

7.1 Tecnología empleada

El objetivo de este apartado es el de detallar las tecnologías empleadas para el desarrollo y el funcionamiento de la aplicación.

7.1.1 Java

Java [13] es un lenguaje de programación orientado a objetos, que permite ejecutar programas en cualquier contexto y máquina, puesto que se ejecuta sobre una máquina virtual. Esto permite destacar su principal ventaja, la portabilidad del código. Java hereda gran parte de su sintaxis del lenguaje de programación C lo que se traduce en un aprendizaje rápido del lenguaje. Aunque es un lenguaje de alto nivel, permite interactuar con sus clases a bajo nivel.

7.1.2 JavaScript

JavaScript [14] es un lenguaje de programación que sigue un paradigma asíncrono. Permite ejecutar programas en navegadores web o mediante intérpretes suyos. Es un lenguaje débilmente tipado y dinámico lo que implica que favorece la creación de un entorno asíncrono.

7.1.3 MySQL

- MySQL [19] es un gestor de bases de datos que funciona con el lenguaje SQL y que permite crear, gestionar y modificar bases de datos de una forma sencilla y gratuita. En este proyecto se ha usado para almacenar e intercambiar la información entre los dos componentes principales de la aplicación.

7.1.4 Bibliotecas empleadas en Java

- BeeCoder [22] es una librería que permite codificar y decodificar cadenas de texto que usan un tipo de codificación conocida como bencoding, permitiendo procesar correctamente los mensajes que se intercambian en la red BitTorrent.

- MySQL Connector [20] es una librería que permite configurar y crear una conexión con la base de datos. De esta forma, es posible conectarse a una base de datos para realizar operaciones sobre la misma.

7.1.4 Bibliotecas empleadas en JavaScript

- MySQL Connector [20] es una librería que permite conectar la aplicación desarrollada en JavaScript con la base de datos para poder realizar las consultas y modificaciones necesarias.

- Infohash-to-metadata [5] es una implementación de la propuesta bep_0009 que permite obtener información detallada de un archivo dado su identificador. Cabe destacar que esta biblioteca es parte de la aplicación WebTorrent.

7.2 Buscador de contenido

El buscador de contenido es el componente más importante de la aplicación, además de presentar mayor complejidad. Antes de la explicación de su desarrollo se va a presentar la arquitectura del buscador.

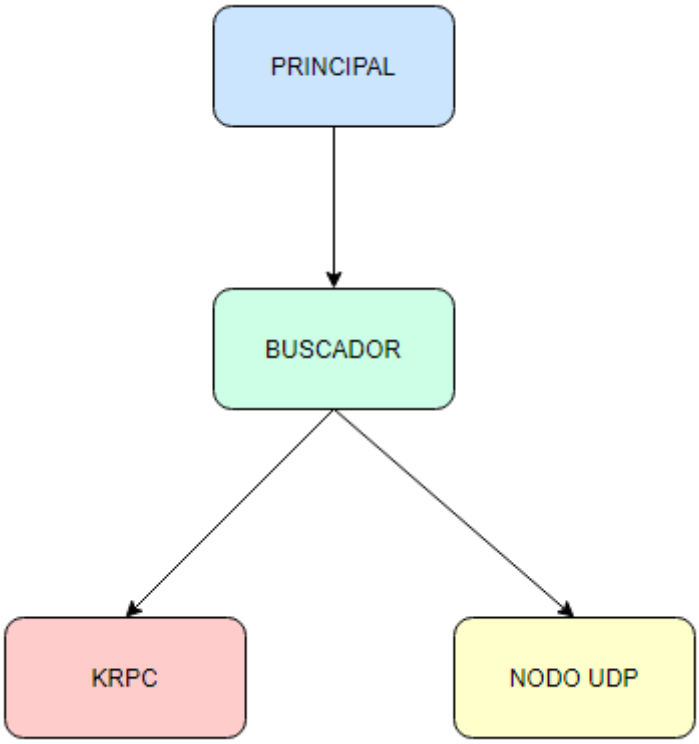

Figura 14. Estructura del buscador de contenido

La clase principal implementa un algoritmo que determina los ciclos de búsqueda, indicando al buscador cuándo tiene que encontrar vecinos y cuándo debe procesar los mensajes recibidos.

El buscador se encarga de descubrir nuevos nodos dentro de la red BitTorrent y de iniciar el intercambio de mensajes con cada uno de ellos. Hace uso de la clase KRPC y de los objetos del tipo nodo para comunicarse con el resto de nodos de la red.

La clase KRPC implementa los métodos necesarios para poder interaccionar con los nodos de la red. También implementa los métodos para poder procesar los mensajes recibidos y enviar las respuestas correspondientes.

Cada objeto del tipo nodo contiene la información de contacto de un nodo de la red. Los nodos son los elementos empleados por el buscador de contenido y la clase KRPC.

La clase KRPC permite interactuar con los nodos de la red, por lo que el nodo que implementa este componente va a poder reaccionar de forma distinta a los mensajes que reciba, independientemente de si son peticiones o respuestas. El método más importante de esta clase es el encargado de procesar los mensajes que el nodo recibe. Dicho método, al recibir un mensaje, determina el tipo de mensaje recibido e invoca al método destinado a contestar ese tipo de mensaje.

```
1  public static void procesar_mensaje(DatagramSocket s, List<Entry<String, Object>>
2      diccionario, TablaEnrutamiento t,InetAddress ipRemota, int puertoRemoto,
3      byte[] nidLocal, String mensajeOriginal,Connection cn) {
4
5          List<Entry<String, Object>> dic = diccionario;
6          Iterator<Entry<String, Object>> itr = dic.iterator();
7          String clave = "", clave_t = "", clave_y = "", clave_q = "";
8          Object valor;
9          List<Entry<String, Object>> dic_a = null;
10         List<Entry<String, Object>> dic_r = null;
11         while(itr.hasNext()) {
12             Entry<String,Object> e = itr.next();
13             clave = e.getKey();
14             valor = e.getValue();
15             switch(clave) {
16                 case "t":
17                     clave_t = (String) valor;
18                     break;
19                 case "y":
20                     clave_y = (String) valor;
21                     break;
22                 case "q":
23                     clave_q = (String) valor;
24                     break;
25                 case "a":
26                     dic_a = (List<Entry<String, Object>>) valor;
27                     break;
28                 case "r":
29                     dic_r = (List<Entry<String, Object>>) valor;
30                     break;
31                 default:
32                     break;
33             }
34             switch (clave_y) {
35                 case "e":
36                     //System.out.println("Se ha recibido un mensaje de error");
37                     //System.out.println(dic);
38                     break;
39                 case "r":
40                     respuesta_find_node(dic_r, t);
41                     break;
42                 case "q":
43                     if(clave_q.equals("ping")) {
44                         peticion_ping(s,clave_t,ipRemota, puertoRemoto);
45                     } else if (clave_q.equals("get_peers")) {
46                         peticion_get_peers(s,clave_t,dic_a,ipRemota,puertoRemoto,
47                         nidLocal,mensajeOriginal,cn);
48                     } else if (clave_q.equals("find_node")) {
49                         peticion_find_node(s,clave_t, dic_a, ipRemota, puertoRemoto);
50                     } else if (clave_q.equals("announce_peer")) {
51                         peticion_announce_peer(s,clave_t,dic_a, ipRemota, puertoRemoto,
52                         nidLocal,mensajeOriginal,cn);
53                     } else {
54                         System.out.println("No se ha procesado un mensaje por el" +
55                         "campo q:" + clave_q");
56                     }
57                     break;
58             }
59         }
60     }
```

Figura 15. Método dedicado al procesamiento de los mensajes que recibe el nodo

Los métodos encargados de enviar las respuestas a los nodos de la red son los métodos descritos en el apartado 3.2.1.

Después de implementar la clase KRPC, el buscador puede procesar los mensajes recibidos, crear los mensajes de respuesta y enviarlos a los nodos correspondientes. Para poder empezar a recibir mensajes de otros nodos, lo primero que tiene que hacer nuestro nodo es unirse a la red.

Para unirse a la red basta con enviar un mensaje de ping a los nodos de arranque. Son nodos que siempre están encendidos y se emplean para que un nodo pueda empezar a participar en la red. El objetivo de la aplicación es el de obtener mensajes de otros nodos por lo que en vez de enviar un mensaje de ping se enviará un mensaje del tipo *find_nodes*. Al enviar este tipo de mensaje a los nodos de arranque se consiguen dos cosas. La primera es que nos podemos unir a la red y la segunda es que obtenemos la información de contacto de otros nodos de la red. Dicha información de contacto será utilizada por el algoritmo para enviar mensajes a los nodos para después recibir mensajes con información.

```
1  NodoArranque na = new NodoArranque("router.utorrent.com", 6881);
2  NodoArranque na1 = new NodoArranque("router.bittorrent.com", 6881);
3  NodoArranque na2 = new NodoArranque("dht.transmissionbt.com", 6881);
4  NodoArranque na3 = new NodoArranque("dht.libtorrent.org", 25401);
```

Figura 16. Nodos de arranque

El algoritmo de la aplicación está formado por un bucle infinito en el que en cada ciclo hace varias comprobaciones. En un primer lugar, comprueba que en la tabla de enrutamiento hay información de contacto de nodos. En el caso de que no disponer información de contacto de otros nodos de la red, debe ponerse en contacto con los nodos de arranque para realizar una consulta que devuelva información de contacto de los nodos de la red.

```
public static void primerosVecinos(byte[] nid, NodoArranque[] listaArranque, DatagramSocket socket) {
    for(int i = 0; i < 4;i++) {
        List<Entry<String,Object>> dic = new LinkedList<Entry<String,Object>>();
        Entry<String,Object> y = new AbstractMap.SimpleEntry<String,Object>("y", "q");
        Entry<String,Object> q = new AbstractMap.SimpleEntry<String,Object>("q", "find_node");
        Entry<String,Object> t  = new AbstractMap.SimpleEntry<String,Object>("t", "bb");
        List<Entry<String,Object>> dic_a = new LinkedList<Entry<String,Object>>();
        Entry<String,Object> id = new AbstractMap.SimpleEntry<String,Object>("id", new String(nid));
        Entry<String,Object> target = new AbstractMap.SimpleEntry<String,Object>("target",
            new String(Nodo.generar_nid_aleatorio()));
        dic_a.add(id);
        dic_a.add(target);
        Entry<String,Object> a = new AbstractMap.SimpleEntry<String,Object>("a", dic_a);
        dic.add(y);
        dic.add(q);
        dic.add(t);
        dic.add(a);
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        BeeCoder.Utils.encodeObject(dic, out);
        byte[] buffer = out.toByteArray();
        DatagramPacket p = new DatagramPacket(buffer,buffer.length,listaArranque[i].get_dirIP(),
            listaArranque[i].get_nPuerto());
        try {
            socket.send(p);
        } catch (IOException e) {
            System.out.println("Se ha producido un error enviando la petición de " +
                "find_nodes desde primerosVecinos");
        }
    }
}
```

Figura 17. Método encargado de obtener nodos de contacto

En este momento, hay información de contacto de otros nodos en la tabla de enrutamiento. El siguiente paso del algoritmo consiste en enviar mensajes a los nodos de la tabla para que conozcan la existencia del nodo del buscador. Hecho esto, el buscador empezara a recibir mensajes de los nodos.

```
public static void hacerVecinos(byte[] nid, TablaEnrutamiento tabla, DatagramSocket socket) {
    TablaEnrutamiento taux = tabla;
    Nodo n;
    for(int i = 0; i < taux.get_numElemen(); i++) {
        n = taux.get_Nodo(i);
        byte[] nid_vecino = new byte[20];
        byte[] nid_nodo = n.get_nid();
        System.arraycopy(nid_nodo, 0, nid_vecino, 0, 15);
        System.arraycopy(nid, 0, nid_vecino, 15, 5);

        List<Entry<String,Object>> dic = new LinkedList<Entry<String,Object>>();
        Entry<String,Object> y = new AbstractMap.SimpleEntry<String,Object>("y", "q");
        Entry<String,Object> q = new AbstractMap.SimpleEntry<String,Object>("q", "find_node");
        Entry<String,Object> t  = new AbstractMap.SimpleEntry<String,Object>("t", "aa");
        List<Entry<String,Object>> dic_a = new LinkedList<Entry<String,Object>>();
        Entry<String,Object> id = new AbstractMap.SimpleEntry<String,Object>("id",
            new String(nid_vecino));
        Entry<String,Object> target = new AbstractMap.SimpleEntry<String,Object>("target",
            new String(Nodo.generar_nid_aleatorio()));
        dic_a.add(id);
        dic_a.add(target);
        Entry<String,Object> a = new AbstractMap.SimpleEntry<String,Object>("a", dic_a);
        dic.add(y);
        dic.add(q);
        dic.add(t);
        dic.add(a);
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        BeeCoder.Utils.encodeObject(dic, out);
        byte[] buffer = out.toByteArray();
        try {
            DatagramPacket p = new DatagramPacket(buffer,buffer.length,
                InetAddress.getByName(n.get_dirIP()), n.get_nPuerto());
            socket.send(p);
        } catch (IOException e) {
            System.out.println("Se ha producido un error enviando la petición de " +
                "find_nodes desde hacerVecinos");
            System.out.println("ipEnvio: " + n.get_dirIP() + "   puertoEnvio: " +
                n.get_nPuerto());
        }
    }
}
```

Figura 18. Método que interactúa con los nodos encontrados

En este momento, el nodo empezará a recibir mensajes de otros nodos de la red. Al procesar dichos mensajes, es posible obtener información de algún archivo que se está compartiendo.

```
while(true) {
    try {
        if(tabla.get_numElemen() == 0) {
            KRPC.primerosVecinos(nidLocal, listaArranque, socketUDP);
        }
        KRPC.hacerVecinos(nidLocal, tabla, socketUDP);
        tabla = new TablaEnrutamiento(1000);

        HiloEntrada hiloSecundario = new HiloEntrada("secundario",
        socketUDP,tabla, nidLocal, nAnnounce);
        hiloSecundario.start();
        Thread.sleep(6000);
        hiloSecundario.detenerHilo();
        if(hiloSecundario.isAlive()) {
            byte[] buffer = "Mensaje de desbloqueo".getBytes();
            DatagramPacket p = new DatagramPacket(buffer, buffer.length,
            socketUDP.getLocalAddress(), socketUDP.getLocalPort());
            socketUDPaux.send(p);
        }
        hiloSecundario.join();
    } catch (Exception e) {
        System.out.println(e);
        e.printStackTrace();
    }
}
```

Figura 19. Bucle principal del buscador de nombres

La clase *HiloEntrada* permite a la aplicación escuchar los mensajes que se reciben en el socket del nodo. Cuando recibe un mensaje, lo descodifica para su posterior procesado mediante el método dedicado a ello de la clase *KRPC*.

```
1  public void run() {
2      while(continuar) {
3          datosEntrada = new byte[1024];
4          DatagramPacket respuesta = new DatagramPacket(datosEntrada,
5          datosEntrada.length);
6          try {
7              socket.receive(respuesta);
8              String s = new String(respuesta.getData());
9              if(!s.contains("Mensaje de desbloqueo")) {
10                 InetAddress ipRemota = respuesta.getAddress();
11                 int puertoRemoto = respuesta.getPort();
12                 String mensaje = new String(respuesta.getData());
13
14                 ByteArrayInputStream in =
15                     new ByteArrayInputStream(respuesta.getData());
16                 Object o = BeeCoder.Utils.decodeObject(in);
17                 List<Entry<String,Object>> l =
18                     (List<Entry<String, Object>>) o;
19                 KRPC.procesar_mensaje(this.socket, l, this.tabla,
20                 ipRemota,puertoRemoto,this.nidLocal, mensaje, this.cn);
21             } else {
22                 //System.out.println("Recibido mensaje de desbloqueo");
23             }
24         } catch (Exception e) {
25             System.out.println(e);
26             e.printStackTrace();
27         }
28     }
29 }
```

Figura 20. Clase que recibe los mensajes de los nodos de la red

7.2.1 Obtención de la información

Los archivos que se comparten en la red se identifican mediante un identificador único y global. Dicho identificador es un hash, resultado de aplicar la función SHA1 sobre la información del archivo. Cuando se recibe un mensaje del tipo *get_peers*, se recibe el identificador del archivo del que desea obtener pares que lo están compartiendo. Ocurre lo mismo con los mensajes del tipo *announce_peer*, se recibe el identificador del archivo que en este caso se va a descargar el nodo.

Por lo tanto, cuando se recibe un identificador mediante uno de estos dos tipos de mensajes, existe una posibilidad de que el identificador recibido pertenezca a un archivo que se está compartiendo en la red y, es por esto por lo que el buscador de nombres busca información sobre los identificadores recibidos. Es posible que no pueda encontrar información para ese identificador debido a que no exista ningún nodo que lo esté compartiendo en ese momento o que el identificador todavía no tenga ningún archivo asociado.

Las aplicaciones cliente del protocolo BitTorrent realizan esta búsqueda de información para todos los archivos que quieren obtener antes de empezar con su descarga.

7.2.2 Estructuras de datos

Para desarrollar el buscador de contenido se han creado una serie de estructuras para poder gestionar la información de una forma más adecuada y sencilla. El conjunto de estructuras desarrolladas está formado por la tabla de enrutamiento, los nodos y los nodos de arranque.

```
1  public class NodoArranque {
2
3      private InetAddress dirIP;
4      private int nPuerto;
5
6      public NodoArranque (String dirIP, int nPuerto) throws UnknownHostException {
7              this.dirIP = InetAddress.getByName(dirIP);
8              this.nPuerto = nPuerto;
9      }
10
11     public InetAddress get_dirIP() {
12         return this.dirIP;
13     }
14
15     public int get_nPuerto() {
16         return this.nPuerto;
17     }
18 }
```

Figura 21. Código del objeto nodo de arranque

Los nodos de arranque almacenan la información de contacto, estando formada por su dirección IP y el número de puerto. Estos objetos permiten acceder de forma rápida a la información de los nodos para poder enviarles mensajes de petición y obtener nuevos nodos con los que contactar.

```java
public class Nodo {

    private byte[] nid;
    private String dirIP;
    private int nPuerto;

    public Nodo (byte[] nid, String dirIP, int nPuerto) {
        this.nid = nid;
        this.dirIP = dirIP;
        this.nPuerto = nPuerto;
    }

    public byte[] get_nid() {
        return this.nid;
    }

    public String get_dirIP() {
        return this.dirIP;
    }

    public int get_nPuerto() {
        return this.nPuerto;
    }

    public String toString() {
        return "nid: " + new String(this.nid) + " dirIP: " + this.dirIP +
        " nPuerto: " + this.nPuerto;
    }


    public static byte[] generar_nid_aleatorio() {
        byte[] res = new byte[20];
        new Random().nextBytes(res);
        return res;
    }

    public static Nodo crear_Nodo_aleatorio (String dirIP, int nPuerto) {
        byte[] nid = Nodo.generar_nid_aleatorio();
        Nodo res = new Nodo(nid, dirIP,nPuerto);
        return res;
    }
}
```

Figura 22. Código del objeto nodo

Un objeto de tipo nodo permite almacenar la información de contacto de un nodo de la red. La información que guarda está formada por el identificador del nodo, una dirección IP y un número de puerto. Además, la clase cuenta con métodos de utilidad que permiten crear nodos ficticios de forma aleatoria.

```java
public class TablaEnrutamiento {

    private int capacidad;
    private Nodo[] lista;
    private int nElemen;

    public TablaEnrutamiento (int cap) {
        this.capacidad = cap;
        this.lista = new Nodo[cap];
        this.nElemen = 0;
    }

    public int get_capacidad() {
        return this.capacidad;
    }

    public Nodo[] get_lista() {
        return this.lista;
    }

    public int get_numElemen() {
        return this.nElemen;
    }

    public Nodo get_Nodo(int pos) {
        return this.lista[pos];
    }

    public boolean esta_llena() {
        return this.nElemen == capacidad ;
    }

    public boolean anyadir(Nodo n) {
        if (!this.esta_llena()) {
            this.lista[this.nElemen] = n;
            this.nElemen++;
            return true;
        } else {
            return false;
        }
    }
}
```

Figura 23. Código del objeto tabla de enrutamiento

La tabla de enrutamiento sirve para que la aplicación almacene una lista con los nodos a los que la aplicación puede enviar mensajes para intentar obtener infohashes nuevos. La tabla cuenta con un indicador de capacidad que limita el número de elementos que puede almacenar en ella con el objetivo de controlar la memoria principal que consume la aplicación en el ordenador.

7.3 Buscador de nombres

Dada la naturaleza del buscador de nombres, se ha decidido adoptar un paradigma asíncrono [9]. Dicho componente realiza múltiples conexiones con los nodos de la red de forma simultánea, y la mejor forma de gestionar las conexiones es mediante un paradigma asincrónico. A diferencia de un paradigma síncrono, en el asincrónico no es necesario esperar a que finalicen las llamadas a los métodos para realizar las siguientes. No hay ninguna necesidad de que todo siga un orden síncrono.

El buscador de nombres, encargado de encontrar la información detallada de los archivos que han sido encontrados, representa una parte importante de la aplicación. La implementación de este componente es más sencilla que la del buscador de contenido.

Su función consiste en conectarse a la base de datos de la aplicación, obtener los identificadores de los archivos descubiertos, buscar información detallada de dichos archivos en la red e insertar en la base de datos la información encontrada.

Este componente tiene que poder conectarse a la base de datos de la aplicación, por lo que el primer paso consiste en crear una conexión que permita hacer consultas a la base de datos. Al emplear MySQL como base de datos, es necesario importar el módulo de mysql al buscador de nombres. La importación del módulo va a permitir al componente conectarse a la base de datos y realizar todo tipo de consultas.

```
1  const mysql = require('mysql');
2
3  var connection = mysql.createConnection({
4      host: '192.168.0.21',
5      user: 'Jorge',
6      password: 'TFGJava2020-User',
7      database: 'tfg',
8      port:'3306'
9  });
```

Figura 24. Configuración de la conexión a la base de datos

Como se puede observar en la figura 24, se importa el módulo de mysql y después se utiliza para crear una conexión con la base de datos. Los parámetros para crear dicha conexión son los mismos que nos podríamos encontrar con otros conectores de bases de datos. En el caso más simple, basta con introducir la dirección del host, el usuario con los privilegios necesarios para hacer consultas, la contraseña del usuario, la base de datos a la que se va a realizar la conexión y el puerto del servidor de la base de datos. El número de puerto no es un campo obligatorio, ya que el conector usa el mismo número de puerto en caso de omitir el argumento.

Después de configurar la conexión, hay que realizar la propia conexión. Para ello, basta con ejecutar el método *connect* de la conexión definida. En el método se puede emplear una función de callback que permita avisar al usuario de que la conexión ha tenido éxito o, en su defecto, que ha habido un problema al realizar la conexión.

```
connection.connect(function(error){
    if (error){
        console.log('Error al establecer la conexion');
        throw error;
    } else {
        console.log('Conexion establecida');
    }
});
```

Figura 25. Establecimiento de la conexión a la base de datos

Una vez está configurada y creada la conexión con la base de datos, la aplicación debe realizar una consulta a la base de datos para obtener los identificadores de los archivos encontrados. Para evitar la posibilidad de obtener varias veces el mismo identificador, la consulta selecciona los identificadores que no se han buscado todavía y que no se ha obtenido información de ellos. Esta consulta devuelve dos identificadores cada vez que se ejecuta y de forma aleatoria de todos los que están pendientes. Se ha establecido el límite a dos identificadores para no saturar al router al realizar las conexiones con los nodos de la red. Los identificadores son devueltos de forma aleatoria para asegurarse de que todos tienen la misma posibilidad de ser buscados en ese momento y de que no se reciben siempre los primeros identificadores, ya que en la base de datos están ordenados por el identificador.

```
function busca () {
    connection.query('select * from archivo where infohash not in ' +
        '(select infohash from info) and buscado is false order' +
        'by rand() limit 3', function(error, result){
            if (error){
                throw error;
            } else {
                for(var i = 0; i < result.length; i++){
                var dia = result[i].fecha.getDate();
                var mes = result[i].fecha.getMonth() + 1;
                var anyo = result[i].fecha.getFullYear();
                var fecha = anyo + "-" + mes + "-" + dia;
                buscarMetadatos(result[i].infohash, fecha, result[i].hora);
            }
        }
    });
}
```

Figura 26. Método que recupera los identificadores de la base de datos

Ahora ya tenemos los identificadores de los archivos que podemos buscar en la DHT. Llegados a este punto, es necesario importar un segundo módulo que es el encargado de obtener la información detallada de los archivos al realizar consultas sobre

los nodos. El módulo a importar tiene el nombre de *infohash-to-metadata* y es el que va a permitir realizar las llamadas para obtener información de los archivos a los nodos de la DHT.

La función encargada de obtener la información detallada de los archivos tiene como primer argumento el identificador del archivo, junto con la fecha y hora en la que fue descubierto por el buscador de contenido. Esta función realiza consultas en los nodos de la red DHT. En caso de obtener información detallada, esta es insertada en la base de datos, en caso contrario, no se inserta en la base de datos. Tanto si se ha encontrado información como si no se ha encontrado, el identificador del archivo se marca como buscado para evitar que se vuelva a buscar. Esto garantiza que los otros identificadores que no han sido objetos de búsqueda, puedan ser buscados. En las situaciones en las que hay información que insertar en la base de datos, se calcula el tamaño del archivo con su correspondiente unidad de tamaño.

```
function buscarMetadatos(infohashI, fechaI,horaI){
metadatos(infohashI, { maxConns:30, fetchTimeout:10000, socketTimeout:8000})
.then(metadata => {
    connection.query(
        "INSERT INTO info (infohash,nombre,tamaño,fecha, hora) VALUES" +
        "('" + infohashI + "','" + metadata.name +"','" +
        calculaTamanyo(metadata.length) + "','"+ fechaI + "','" + horaI+"')",
            function(error,result){
            if(error){
                console.log('error al insertar');
                console.log(error);
            } else {
                console.log('insertado');
                console.log(metadata.length);
                console.log(metadata.name);
                console.log(calculaTamanyo(metadata.length));
            }
            actualizarBuscado(infohashI);
        });
    }).catch(error => {
        console.log('No se ha encontrado');
        actualizarBuscado(infohashI);
    });
}
```

Figura 27. Función que busca la información detallada en la DHT

Una vez descritas las funciones necesarias para que este componente pueda funcionar, falta una última función para que todo esto funcione de forma cíclica y sin parar. Es una función destinada a establecer la frecuencia de ejecución de las funciones descritas, que por defecto, está establecida a once segundos para dar tiempo a buscar la información detallada de los archivos. Después de implementar todo esto, el buscador de nombres es capaz de obtener información detallada de los archivos encontrados en la red.

7.4 Base de datos

Para poder almacenar la información respetando el modelo de datos de la aplicación, basta con crear una base de datos en MySQL con dos tablas. Una tabla destinada a contener la información básica de los archivos de la red y la otra destinada a almacenar la información detallada de los archivos que encuentre el buscador de nombres.

```
CREATE TABLE `archivo` (
  `infohash` varchar(40) COLLATE utf8_spanish_ci NOT NULL,
  `nombre` varchar(300) COLLATE utf8_spanish_ci DEFAULT NULL,
  `ip` varchar(15) COLLATE utf8_spanish_ci DEFAULT NULL,
  `puerto` int DEFAULT NULL,
  `fecha` date NOT NULL,
  `hora` time NOT NULL,
  `buscado` tinyint NOT NULL DEFAULT '0',
  PRIMARY KEY (`infohash`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_spanish_ci
```

Figura 28. Creación de la tabla archivo en la base de datos

El código de la figura 28 muestra las instrucciones SQL que permiten crear la tabla que almacena la información básica de los archivos.

```
CREATE TABLE `info` (
  `infohash` varchar(40) COLLATE utf8_spanish_ci NOT NULL,
  `nombre` varchar(300) COLLATE utf8_spanish_ci NOT NULL,
  `tamaño` varchar(10) COLLATE utf8_spanish_ci DEFAULT NULL,
  `fecha` date NOT NULL,
  `hora` time NOT NULL,
  PRIMARY KEY (`infohash`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_spanish_ci


ALTER TABLE info
ADD CONSTRAINT CAj-infohash
FOREIGN KEY (infohash)
REFERENCES archivo (infohash)
ON DELETE NO ACTION
ON UPDATE NO ACTION ;
```

Figura 29. Creación de la tabla info en la base de datos

Por otro lado, el código de la figura 29 crea la tabla que almacena la información específica de los archivos y establece la relación que existe con la tabla que guarda la información básica. Se trata de una clave ajena que relaciona la información detallada con un archivo.

8. Equipos para pruebas

Para la realización de las pruebas se ha decidido poner en funcionamiento la aplicación durante 30 días. Durante este periodo, se podrán obtener datos para realizar estadísticas de qué tipo de mensaje se obtiene más identificadores, ver el rendimiento de la aplicación o incluso estudiar el comportamiento de los routers bajo la carga que pueda suponer la aplicación.

El entorno de pruebas está formado por dos ordenadores conectados al mismo router. El ordenador principal tiene la base de datos de la aplicación y el buscador de nombres. Por lo tanto, el ordenador secundario implementa el componente encargado de obtener la información detallada de los archivos.

El ordenador principal cuenta con una velocidad de subida de 300 Mbits/s y con 150 Mbits/s de bajada. Por otro lado, el ordenador secundario cuenta con una velocidad de subida de 600 Mbits/s y con 180 Mbits/s de bajada. Las velocidades de ambos ordenadores son más que suficientes para que la aplicación pueda enviar todos los mensajes sin problemas.

El ordenador principal cuenta con un procesador Intel Core i5-7500 acompañado de 16 GB de memoria RAM con un disco de estado sólido de 480 GB de capacidad. La conexión a internet se realiza mediante un receptor wifi.

El ordenador secundario cuenta con un procesador Intel Core i5-6200U acompañado de 4GB de memoria RAM con un SSD de 250 GB de capacidad. La conexión a internet se realiza mediante un cable Ethernet.

9. Conclusiones

En este apartado se va a comentar los resultados de la aplicación que ha estado funcionando durante 30 días de forma ininterrumpida.

La aplicación a desarrollar tiene como objetivo ser completamente autónoma, por ello, se ha dejado en funcionamiento durante un determinado periodo de tiempo para comprobar si era capaz de funcionar sin errores y sin la intervención de un usuario. La prueba ha sido un éxito en ese aspecto, dado que no ha sido necesario intervenir durante su funcionamiento y en el momento de realizar la revisión no se han detectado errores.

Durante el periodo de la prueba, el buscador de contenido ha sido capaz de interceptar durante el intercambio de mensajes con los nodos la cantidad de 3.437.977 identificadores de archivos. Aunque la cifra es bastante elevada, cabe destacar que no todos esos identificadores tienen un archivo asociado o no se está compartiendo con más nodos que el que el propio nodo que lo está buscando. Durante este periodo, el buscador de nombres ha sido capaz de buscar información de 1.194.170 archivos, encontrando información detallada de 23.120 archivos. Con estas cifras, se puede estimar que el buscador de nombres habrá encontrado información detallada de 68.700 archivos cuando haya procesado el resto de identificadores descubiertos por el buscador de contenido.

Existen portales web [25] que llevan en funcionamiento durante años que tienen en sus bases de datos entorno a unos 15 millones de torrents indexados. Algunos portales incluso tienen más torrents indexados. La mayoría de estos portales cuentan con la ayuda de sus usuarios que pueden ir aportando más contenido a la base de datos.

Teniendo en cuenta estas cifras y los portales que llevan años en funcionamiento, podemos considerar que el rendimiento de la aplicación es bastante elevado.

Por otro lado, podemos afirmar que hemos cumplido con todos los objetivos propuestos para el desarrollo de la aplicación. La base de datos relacional de la aplicación es sencilla y ha almacenado la información de forma correcta. La información detallada que se almacena de los archivos incluye tanto el nombre del archivo como el tamaño. El buscador de contenido es rápido y eficiente como se ha podido ver con los resultados.

infohash	nombre	tamaño	fecha	hora
286d2e5b4f8369855328336ac1263ae02a7a60d5	ubuntu-18.04.4-desktop-amd64.iso	1.98 GB	2020-07-11	09:48:25
NULL	NULL	NULL	NULL	NULL

Figura 30. Información detallada de un archivo almacenado

10. Relación del trabajo realizado con los estudios cursados

Los estudios cursados en el grado de ingeniería informática han determinado en cierta medida el desarrollo de este proyecto de la siguiente manera.

En primer lugar, el desarrollo y la población de la base de datos se ha llevado a la práctica atendiendo a los conocimientos aprendidos en las dos asignaturas relacionas con bases de datos. La asignatura de Bases de datos y sistemas de información ha influenciado en la forma de realizar las consultas a la base de datos del proyecto y ha servido para optimizar las consultas, para que sean más eficientes. La otra asignatura que ha influenciado en el desarrollo de la base de datos ha sido Tecnología de bases de datos, y su conocimiento ha permitido optimizar la indexación de los datos almacenados en la base de datos. Al ser una base de datos de reducida complejidad, los conocimientos han sido suficientes para afrontar cualquier problema.

Los conocimientos teóricos de la asignatura Sistemas y servicios en red han sentado las bases para comprender la arquitectura P2P que emplea el protocolo BitTorrent. En dicha asignatura se realizó un estudio sobre las características y el funcionamiento del protocolo, lo que despertó un interés en estudiar en profundidad el protocolo para comprender un poco más su funcionamiento.

Para el desarrollo del buscador de nombres se ha hecho uso de los conocimientos adquiridos en la asignatura de Tecnologías de sistemas de información. Ha permitido crear un componente con un comportamiento asíncrono completamente funcional y ha servido para crear un componente que fácilmente puede funcionar en varios equipos de forma simultánea.

Por otro lado, la asignatura de Redes de computadores ha sentado las bases para llevar a cabo el intercambio de mensajes entre ordenadores conectados a la red. Los conocimientos adquiridos han permitido entender mejor la elaboración de los mensajes y su envío a través de la red.

Por último, la integración de diversos componentes de red existentes con el proyecto ha sido posible con los conocimientos de la asignatura Integración de aplicaciones. Ha permitido adaptar los componentes desarrollados para que sean capaces de interactuar con elementos externos al proyecto que no podemos modificar.

11. Trabajos futuros

Esta versión del proyecto cumple con los requisitos especificados en el principio, no obstante, otra versión del mismo podría incluir una serie de mejoras e incluso introducir nuevas funcionalidades. Las siguientes propuestas no se incluyeron como objetivos al no disponer de suficiente tiempo para implementarlas y realizar pruebas de funcionamiento.

Una mejora a considerar sería la de agrupar los tres componentes bajo una misma aplicación de usuario. El resultado sería una aplicación que implementaría los tres componentes con una puesta en marcha y manejo más sencillo para el usuario. Sería una mejora importante dado que, en la versión actual del proyecto, el control sobre los componentes se hace desde una consola.

Para visualizar el contenido descubierto en la red, es necesario realizar consultas a una base de datos. Esta acción puede resultar compleja para los usuarios. Es por esto, que se podría desarrollar una interfaz sencilla que permita la visualización de los archivos encontrados en la red.

Por otro lado, únicamente se puede obtener información de los archivos encontrados. No hay ninguna funcionalidad que permita a los usuarios realizar una descarga de un archivo. Una función que se podría añadir al proyecto sería, la posibilidad de generar el fichero *torrent* que, permita al usuario realizar la descarga del archivo que desee.

Otra mejora a desarrollar sería la obtener el número de pares que están compartiendo un archivo. Esta utilidad permitiría dar una idea al usuario de si es posible obtener el archivo de los pares de la red.

12. Referencias

[1] Ataque de denegación de servicio. Recuperado (2020, agosto 14) de https://es.wikipedia.org/wiki/Ataque_de_denegaci%C3%B3n_de_servicio

[2] BitTorrent. Recuperado (2020, agosto 14) de https://www.BitTorrent.com/es/

[3] BitTorrent Inc. BitTorrent, (2020, mayo 1). Recuperado de https://github.com/BitTorrent/BitTorrent.org

[4] Cohen, Bram. The BitTorrent Protocol Specification, (2008, enero 10). Recuperado de https://www.BitTorrent.org/beps/bep_0003.html

[5] Das, Bubun. Infohash-to-metadata, (2019, febrero 21). Recuperado de https://github.com/bubundas17/infohash-to-metadata

[6] DHT Infohash Indexing, (2016, diciembre 20). Recuperado de https://www.BitTorrent.org/beps/bep_0051.html

[7] Emetiel. ¿Qué es exactamente el info_Hash en un archivo torrent?, (2015, febrero 5). Recuperado de https://www.it-swarm.dev/es/hash/que-es-exactamente-el-info-hash-en-un-archivo-torrent/1051545894/

[8] García, José. (2019, agosto 31). Napster: inicio, auge y caída del servicio que puso en jaque a la industria musical. Recuperado de https://www.xataka.com/historia-tecnologica/napster-inicio-auge-caida-servicio-que-puso-jaque-a-industria-musical

[9] Giancarlo, Corzo. Programación Asíncrona, (2017, mayo 23). Recuperado de https://medium.com/laboratoria-how-to/programacion-asincrona-cea3bad7c3c6

[10] Harrison, David. Index of BitTorrent Enhancement Proposals, (2008, enero 10). Recuperado de https://www.BitTorrent.org/beps/bep_0000.html

[11] Harrison, David. Peer ID Conventions, (2008, febrero 27). Recuperado de https://www.BitTorrent.org/beps/bep_0020.html

[12] Hoffman, John. Multitracker Metadata Extension, (2008, febrero 7). Recuperado de https://www.BitTorrent.org/beps/bep_0012.html

[13] Java. Recuperado (2020, agosto 14) de https://www.java.com/es/

[14] JavaScript. Recuperado (2020, agosto 8) de https://developer.mozilla.org/es/docs/Web/JavaScript

[15] Kademlia. Recuperado (2020, julio 24) de https://es.wikipedia.org/wiki/Kademlia

[16] Kurose, James F. (2010), Redes de computadoras, España, Editorial Pearson

[17] Libtorrent. Recuperado (2020, agosto 14) de https://www.libtorrent.org/

[18] Loewenstern, Andrew. DHT Protocol, (2008, enero 31). Recuperado de https://www.BitTorrent.org/beps/bep_0005.html

[19] MySQL Recuperado (2020, agosto 14) de https://www.mysql.com/

[20] MySQL Conncetor. Recuperado (2020, agosto 23) de https://www.mysql.com/products/connector/

[21] Secure Hash Agorithm. Recuperado (2020, agosto 23) de https://es.wikipedia.org/wiki/Secure_Hash_Algorithm

[22] Soloviev, Dimitry. Beecoder, (2014, septiembre 19). Recuperado de https://github.com/soulaway/beecoder

[23] Stoll, Adrian. Bencoding, (2016, abril 25). Recuperado de https://adrianstoll.com/bencoding/

[24] Tor Recuperado (2020, agosto 14) de https://www.torproject.org/es/

[25] Torrentz2 Recuperado (2020, agosto 14) de https://torrentz2.eu/

[26] Transmission. Recuperado (2020, agosto 14) de https://transmissionbt.com/

[27] Ttorrent. Recuperado (2020, agosto 14) de https://ttorrent.org/

[28] Utorrent. Recuperado (2020, agosto 14) de https://www.utorrent.com/intl/es/

[29] Vuze. Recuperado (2020, agosto 14) de https://www.vuze.com/

[30] Wang, Liang. Real-World Sybil Attacks in BitTorrent Mainline DHT. Recuperado (2020, agosto 23) de https://www.cl.cam.ac.uk/~lw525/publications/security.pdf

13. Glosario

Host: Es una palabra inglesa que hace referencia a un ordenador

Streaming: Es un término que hace referencia al hecho de reproducir contenido multimedia sin necesidad de descargarlos completamente. Se consigue mediante el uso de un buffer en la que se almacena una pequeña parte del contenido y que permite visualizarlo sin tener el archivo completo.

Flooding: Es un término inglés que significa inundación. En informática hace referencia a un tipo concreto de ataque, la denegación de servicios. Dicho ataque se consigue mediante el envío masivo de mensajes a un servidor.

GitHub: Es una plataforma que permite almacenar proyectos haciendo uso del control de versiones. Se utiliza principalmente para el desarrollo de código.

Cliente-servidor: Es una arquitectura software en la que el proveedor de recurso se conoce como servidor y los consumidores de recursos llamados clientes.

Bencoding: Es el tipo de codificación usado en el protocolo de BitTorrent por los participantes de la red al enviar y recibir mensajes.

Metadatos: Información de un archivo torrent.

Torrent: Es un fichero que contiene la información detallada de un archivo, también conocido como archivo .torrent

Infohash: Identificador de un archivo o de un par en la red

Pieza: Unidad de datos en la que se puede dividir un archivo en BitTorrent. El tamaño depende del propio archivo.

Peer: Es una aplicación cliente que implementa el protocolo BitTorrent y se encuentra conectado a la red BitTorrent

14. Lista de acrónimos

UTF-8: *Unicode Transformation Format*

DHT: *Distributed Hash Table*

UDP: *User Datagram Protocol*

TCP: *Transport Control Protocol*

SHA1: *Secure Hash Algorithm*

RPC: *Remote Procedure Call*

IP: *Internet Protocol*

P2P: *Peer To Peer*

VoIP: *Voice over IP*