Práctica opcional

Sistemas Distribuidos 19-20

Sistema de intercambio de ficheros mediante DHT con interfaz web

Francisco Pinto Santos
Héctor Sánchez San Blas

Índice

1. Introducción

Para la realización de esta práctica se solicita realizar el diseño de un sistema para compartir datos mediante un *DHT*.

Puesto que se otorga liberta en el diseño y la realización de este, se decidió llevar a cabo una implementación más funcional mediante el desarrollo de una aplicación. Es por ello por lo que se ha realizado la implementación de una pequeña aplicación de intercambio de ficheros, utilizando módulos basados en el protocolo *bittorrent.*
El objetivo principal es el de poder compartir archivos mediante su subida a través de la creación de un fichero *.torrent* del mismo y almacenarlo en el *DHT* para su posterior recuperación (descarga).
De esta forma, cualquier persona que posea dicho archivo .*torrent* puede descargarlo.

Teniendo el objetivo claro, se estableció que un lenguaje de programación adecuado para realizar el *backend* del proyecto era *Typescript* junto a *NodeJs*. Para realizar el *frontend* una interfaz web, basada en el *framework Vue.js* junto a el pack de componentes *Vuetify*.

Por último, cabe destacar que la comunicación entre el *frontend* y el *backend*, se ha utilizado una *API* utilizando el módulo *Express.js*.

A continuación, se van a explicar que módulos se utilizan, el funcionamiento de la aplicación y la explicación del código.

2. Proceso de búsqueda de módulos

Puesto que se usa *NodeJs*, como entorno en tiempo de ejecución, y se busca desarrollar una aplicación más funcional, en los primeros momentos del desarrollo se llevó a cabo una investigación y búsqueda de módulos con los cuales desarrollar la propuesta, para ello, se buscaban aquellos módulos que supusiesen un apoyo en la implementación de la funcionalidad asociada a el *DHT* y al tratamiento de los ficheros *.torrent*.

Durante este punto, se va a explicar el proceso de búsqueda, valoración y justificación (si se ha descartado o integrado) de módulos realizado hasta llegar a las bibliotecas que se creen idóneos para los requisitos, y por tanto, que se han utilizado:

- https://github.com/jeanlauliac/kademlia-dht : implementación en *javascript* para el protocolo *Kademlia* (entendido como el conjunto de algoritmos y estructuras de datos) en la implementación de *DHT*. Este módulo proporciona las funciones principales de *Kademlia*. Es decir, el direccionamiento basado en nodos y contenido, accesible mediante las dos primitivas de un *DHT*, *put* y *get*, utilizadas para almacenar y recuperar clave/valor respectivamente.
  Cabe destacar que se trata de una implementación adaptable al navegador.
  No obstante, tras unos días de desarrollo con este módulo, se descartó por los siguientes motivos:
  - La documentación es escasa, únicamente la generada de forma automática a través del código.
  - Es una implementación que trae demasiada funcionalidad para lo buscado y resuelve algunos desafios que gustaría afrontar como desarrolladores.

- https://github.com/khaosdoctor/node-dht: se trata de una implementación llevada a cabo en la *UFABC*, de un *DHT* basado en *RPC*, con unas reglas definidas establecidas con respecto a la teoría del funcionamiento de los *DHT*.
  A pesar de ser una implementación correcta de *DHT* fue descartada por los mismos motivos que el módulo anterior.

- https://github.com/libp2p/js-libp2p : módulo cuya finalidad es la de construcción de aplicaciones *P2P* que define interfaces que, una vez expuestas, permiten que otros protocolos y aplicaciones que las usen intercambien información, permitiendo su actualización y adaptabilidad en tiempo de ejecución.
  En sí, es un módulo bastante completo para desarrollar aplicaciones *P2P*, sin embargo, presenta los siguientes problemas:
  - El módulo está en desarrollo (versión *beta*) y tiene varios *issues* sobre el funcionamiento básico de este en *GitHub* sin resolver desde hace tiempo (por lo que se considera que ya no le dan soporte).
  - De nuevo, la documentación es escasa, únicamente la generada de forma automática a través del código. Además, en este modulo se ve agravado este problema, debido a que no es muy usado y se encuentran pocas soluciones al estilo *cookbook*, además de no haber casi preguntas en *StackOverflow*.

- https://github.com/webtorrent/webtorrent : cliente de *streaming* mediante *P2P*, basado en el protocolo *Bittorrent*, preparado para su uso en *NodeJs* y el propio navegador (cliente web).
  Esta basado en *TCP* y *UDP*, concretamente para la comunicación, en el caso de ser el cliente web, utiliza *WebRTC* (para el transporte *P2P*) y en cualquier otro caso utiliza *RPC*.
  El único problema que presenta es que este módulo incluye toda la funcionalidad necesaria para el intercambio de información mediante el protocolo *BitTorrent*, por lo que lo consideramos demasiado completo y perdía sentido incluirlo en el contexto de este proyecto, puesto que se iba a reducir demasiado el trabajo que se debía llevar a cabo.

- https://github.com/webtorrent/create-torrent y https://www.npmjs.com/package/parse-torrent: módulos que otorgan la funcionalidad necesaria para la creación y parseo de ficheros *.torrent*.
  Fueron descartados, porque no permitían la personalización del contenido del fichero *.torrent* tanto como fuera necesario, incluyendo información de *trackers* e intentando conectar con *trackers* por defecto en el caso de no estar presentes en el fichero.
  Debido a esto, investigamos el formato de los ficheros *.torrent* y vimos que se trataba de un *JSON* con unos campos determinados e implementamos el proceso que harían estos módulos, debido a la sencillez de este.

Por último, tras descartar los módulos anteriores y al investigar más sobre *WebTorrent*, descubrimos, que este a su vez, era una recopilación de módulos utilizados para el intercambio de información en *P2P*. Concretamente, utilizaba otro módulo para la implementación de la funcionalidad básica sobre el protocolo *Bittorrent* que si interesaba:

- https://github.com/webtorrent/bittorrent-dht : implementación del protocolo *BitTorrent* siguiendo las *RFC* establecidas, que trabaja con *Kademlia* a bajo nivel para direccionar por nodo y contenido.
  Se trata de un módulo robusto, probado y documentado que otorga la funcionalidad básica para llevar a cabo una implementación funcional siguiendo el protocolo *BitTorrent*.
  Entre las ventajas que otorga este módulo, están las siguientes:
  - Al saltarse la NAT, permite acceder a el *DHT* público asociado a este protocolo, pero también permite realizar la conexión entre *peers* de una misma red.
  - Ofrece las primitivas básicas asociadas con el intercambio de información *P2P* (*put* y *get*) permitiéndonos construir toda la funcionalidad de nuestra aplicación por encima.
  - Ofrece la funcionalidad para que al instanciar el objeto para trabajar con el *DHT*, indicarle un "*bootstrap peer*", para poder realizar el acceso a la información almacenada en el *DHT* sin utilizar *trackers*.

3. Manual de usuario

En este apartado se explicará de forma visual cual es el funcionamiento de la aplicación, para ello se explicarán las distintas vistas y su contenido:

- **Vista "Files":** En esta ventana nos encontramos dos opciones, subir archivo y almacenarlo en el *DHT,* o descargar un archivo, a partir de un fichero *.torrent* :

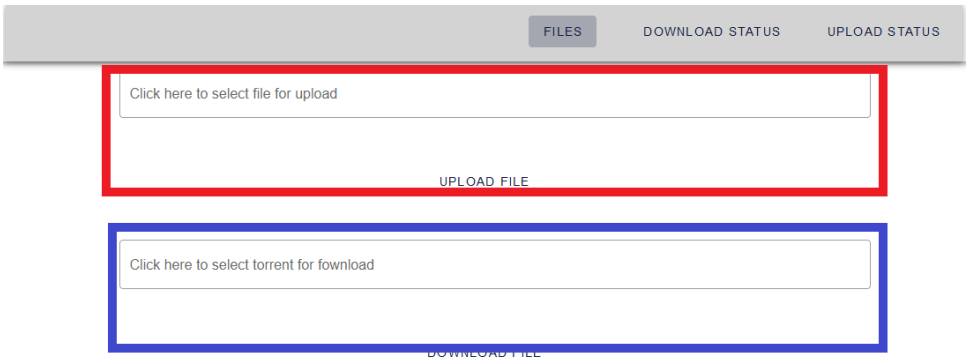

Figura 3.1

Delimitado en rojo se encuentra la zona para seleccionar un archivo y subirlo, y respectivamente, delimitado en azul, la zona para seleccionar un archivo *.torrent* y comenzar la descarga de su respectivo archivo.

- **Vista "Upload Status":** en esta ventana se pueden observar dos tablas, la primera nos mostrará el estado de la subida de los archivos (porcentaje subido), es decir, el porcentaje de *chunks* (fragmentos de fichero de 950 *bytes* o menos) que se han almacenado en el *DHT*. Esto se puede observar en la siguiente imagen:

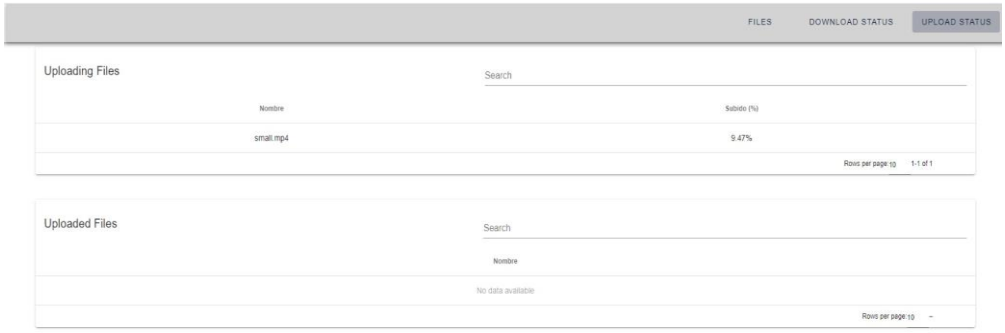

Figura 3.2

Cuando todos los *chunks* han sido almacenados, el archivo se mostrará en la tabla de archivos que se han subido completamente, de forma que nos dará la opción de obtener el archivo *.torrent* mediante el símbolo de descarga que aparecer al lado del nombre del archivo subido tal y como se puede observar a continuación:

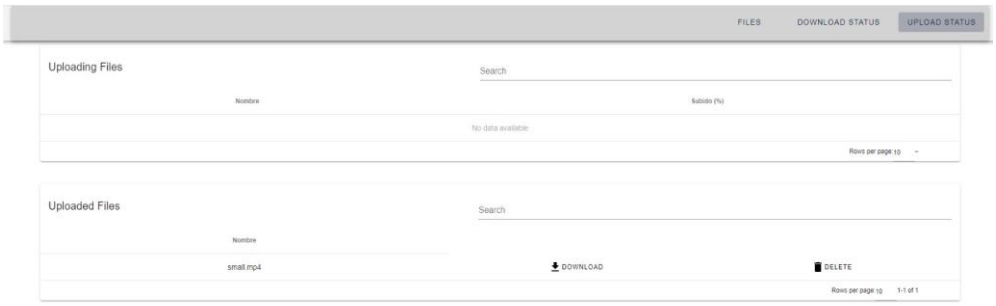

Figura 3.3

- **Vista "Download Status":** vista similar a la de *Upload* pero destinada a la obtención de archivos a partir del archivo *.torrent*. En esta ventana se pueden observar dos tablas, la primera nos mostrará el estado de la descarga de los archivos (porcentaje descargado), es decir, el porcentaje de *chunks* que se han recuperado del *DHT*. Esto se puede observar en la siguiente imagen:

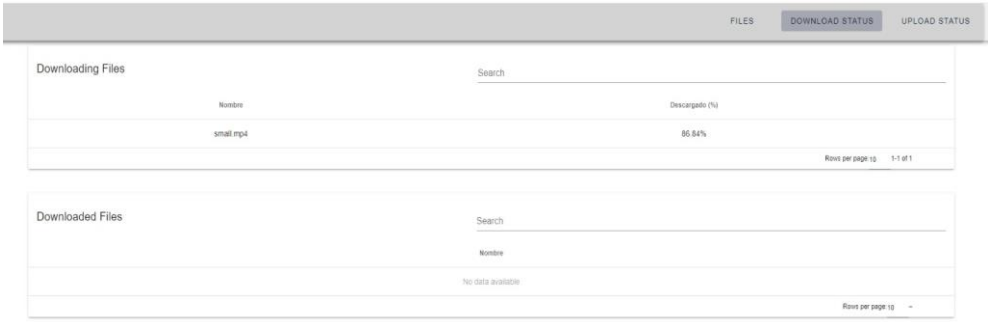

Figura 3.4

Cuando todos los *chunks* han sido obtenidos, el archivo se mostrará en la tabla de archivos que se ha descargado completamente, de forma que nos dará la opción de obtener el archivo mediante el símbolo de descarga que aparecer al lado del nombre del archivo subido tal y como se puede observar a continuación:

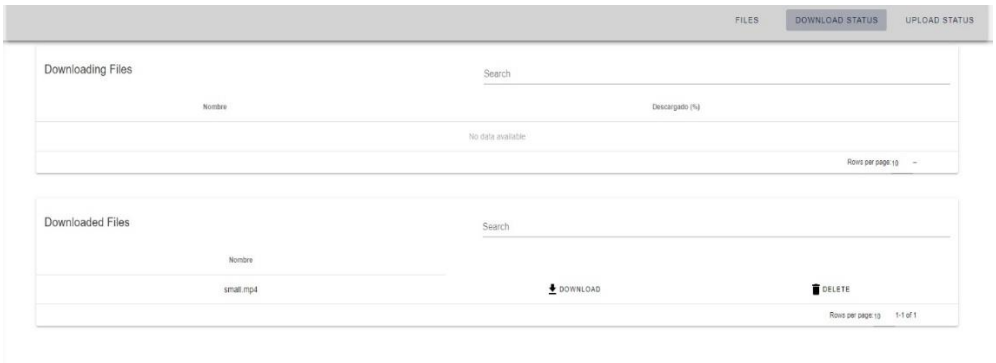

Figura 3.5

Una vez mostradas las ventanas, se puede explicar cuál es el funcionamiento de la aplicación. Cuando se desea compartir un archivo, se debe subir en mediante la opción mostrada en la vista *Files*, seleccionando el archivo a compartir y dándole al botón

Upload. Una vez hecho esto, si nos trasladamos a la vista *Upload Status* se puede observar la evolución del porcentaje almacenado en el *DHT*. Cuando el porcentaje es 100, es decir, está completamente subido, el nombre del archivo aparecerá en la segunda tabla dándo la opción del obtener el archivo *.torrent* generado.
Se debe obtener de esta manera puesto que en los requisitos del trabajo se especifica que no debe de ser un proceso centralizado, por tanto, si alguien quiere obtener un archivo subido por otra persona, debe de haber recibido de otro participante el archivo *.torrent* generado al subirlo para compartir.

En lo referente al proceso de descarga u obtención del archivo, en la vista *Files*, en el apartado de *Download* debemos seleccionar el fichero *.torrent* del cual se desea obtener el archivo original. Una vez seleccionado y habiendo hecho *click* en el botón de *Download*, si accedemos a la pestaña *Download Status* nos encontraremos con algo similar a lo sucedido en *Upload Status* pero adaptado de la descarga del archivo, es decir, la recuperacion de la información almacenada en el *DHT*. Cuando el proceso termina, si se utiliza el botón de descarga obtendremos el archivo completo.

Cabe destacar que, en ambos casos (*Upload* y *Download*), cuando un archivo ha terminado de subirse o descargarse, respectivamente, después de descargar el archivo generado, se puede eliminar cuando el proceso ha terminado mediante el botón que así lo indica.
Esto trae como consecuencia que los ficheros asociados a esa subida o descarga sean eliminados del servidor, por lo que, una vez hecho, no se puede obtener el archivo relacionado con este registro de la tabla.

Cabe destacar que, en el siguiente enlace, se ha añadido un video del funcionamiento del sistema, para mostrar el funcionamiento de este en vivo.
En el ejemplo de funcionamiento, con cada ventana del navegador se ha accedido a la interfaz web del sistema desplegado en máquinas distintas y los ficheros generados durante el proceso se han añadido en la carpeta *doc/example*.

https://streamable.com/3jd0u0

4. Aspectos relevantes del proyecto

En este apartado, se comentará de manera somera las distintas partes del proyecto de backend en materia de aspectos generales, aspectos de los controladores y modelos. Y, por último, algunos aspectos relevantes del despliegue.

4.1. Aspectos generales

- Configuración: para cargar la configuración en el fichero *config.ts*, se carga el fichero *config.json* (si está disponible) y, en otro caso, se carga el fichero *default-config.json*. Tras ello, el fichero cargado se parsea y se obtiene la configuración de la aplicación: puerto de HTTP, puerto del *peer DHT*, tamaño de *chunk*, *bootstrap peer*, nivel de *logging*, etc.
- Logging: en el fichero *log.ts* se ha definido un *logger* adaptado a las necesidades del sistema.
- En el fichero *index.ts* se inicia el modelo de *DHT* y la *API* de *Express* (*HTTP*), además, se registran las manejadoras para realizar una desconexión ordenada del *DHT* cuando muerta el proceso.
- Se ha registrado un *crashguard* en el fichero *crashGuard.ts*, para registrar las posibles excepciones no controladas.
- En el fichero *app.ts*, se puede encontrar la configuración de la *API* de *Express* y el de registrar los controladores y vistas para ser accesibles mediante esta interfaz.
- En el fichero *util/http.ts*, se han registrado constantes relacionadas con el protocolo *HTTP* como son los códigos de retorno, *content-type*, etc.
- En el fichero *models/types.d.ts* se han declarado para *Typescript*, los módulos que no tienen un *@types* declarado, como es *bittorrent-dht*.
- En este sistema solo hay dos características que no han podido ser optimizadas:
    - la subida y bajada de ficheros es lenta: ya que según se ha descubierto, la biblioteca únicamente permite realizar operaciones contra el *DHT* de manera secuencial por seguridad (evitar ataques de denegación de servicio), y estas tienen que estar confirmadas con acuse de recibo (ya sea positivo o negativo) de todos los *peers* implicados, por lo que subir ficheros de gran tamaño es un proceso lento.
    - Si se apaga un *peer*, aunque sea de forma ordenada, muchas veces se pierde la información que ha subido este. En la documentación no hay comentarios sobre esto y tampoco *issues* abiertos en *GitHub*, pero por lo leído en *StackOverflow*, la mayoría de la comunidad opina que se trata de una política de seguridad.

4.2. Controladores

- Hay dos controladores, uno para realizar descargas (*controllers/downloadsController*) y otro para realizar subidas (*controllers/uploadsController*).
- Ambos controladores comparten estructura:
    - Ambos tienen un modelo con un listado de descargas o subidas (*models/downloads* y *modes/uploads* respectivamente), en el que está registrado el estado de estas y los objetos necesarios para trabajar con ellas.
    - Ambos tienen cuatro operaciones disponibles, que son:
        - *status*: listar el estado de las descargas o subidas.
        - *create*: crear una descarga o una subida en el modelo y comenzarla.
        - *file o torrent*: obtener el fichero *.torrent* asociado a la subida o el fichero obtenido de la descarga.
        - *delete*: elimina la información asociada a una descarga o subida del modelo correspondiente.

4.3. Modelos

- *uploads.ts y downloads.ts*: son los modelos de subidas y descargas respectivamente. Almacenan un listado de las subidas y descargas. Además, en estos se definen las clases *Upload* y *Download* con la información de ficheros Torrent y regulares necesarios para trabajar con las subidas y descargas.
- *dht*.ts: es el modelo de *DHT* y define el funcionamiento necesario para trabajar con intercambio de información *P2P* mediante la biblioteca *bittorrent-DHT*:
    - Generación de id.
    - Iniciar conexión con otros *peers*, incluido encontrar el *bootstrap peer.*
    - Primitivas para añadir y recuperar información del *DHT*: *get* y *put.*
    - Registrar manejadoras de los distintos eventos relacionados con el *DHT* para imprimir información sobre estos.
- *chunk.ts*: aquí se define la clase *Chunk* que representa un fragmento de fichero compuesto por dos *Buffers* (contenido binario):
    - *cid*: contiene el id del fragmento dentro del DHT, el cual es su *hash* hallado mediante el algoritmo *SHA-1*.
    - *value*: contiene el propio valor del fragmento en forma binaria. Debido a las especificaciones de la biblioteca utilizada, tiene que ocupar menos de 1000 *bytes*, por lo que se ha establecido un tamaño de 950 *bytes* como máximo.

  Este modelo ofrece dos métodos principales que son *resolve* y *store*, que respectivamente recuperan y almacenan el *chunk* del *DHT*.
- *file.ts*: contiene dos clases principales: *FileBittorrent* y *Torrent*, que respectivamente contienen la funcionalidad necesaria para dividir ficheros en *chunks* y juntarlos (en el caso de *FileBittorrent*), y para parsear y generar ficheros *.torrent* a partir de los fragmentos de un fichero (en el caso de *Torrent*).

4.4. Despliegue

Para realizar el despliegue se ha adjuntado el fichero *deploy.bash* que realiza el despliegue del sistema, para lo cual hay que situarse sobre la carpeta *deploy*.
Para conocer los comandos disponibles sobre el script de despliegue basta con invocarlo con la opción *-h*.

5. Bibliografía

- https://github.com/jeanlauliac/kademlia-dht
- https://github.com/khaosdoctor/node-dht
- https://github.com/libp2p/js-libp2p
- https://github.com/webtorrent/webtorrent
- https://github.com/webtorrent/bittorrent-dht
- https://github.com/webtorrent/create-torrent
- https://www.npmjs.com/package/parse-torrent
- https://en.wikipedia.org/wiki/Torrent_file
- https://fileformats.fandom.com/wiki/Torrent_file
- https://stackoverflow.com/