Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

Creación de una herramienta para la investigación de la red BitTorrent

Autor: Raúl Calvo Laorden

Tutor: Manuel Sánchez Rubio

2019

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

Trabajo Fin de Grado

Creación de una herramienta para la investigación de la red BitTorrent

Autor: Raúl Calvo Laorden

Director: Manuel Sánchez Rubio

Tribunal:

Presidente: « Nombre y Apellidos »

Vocal 1º: « Nombre y Apellidos »

Vocal 2º: « Nombre y Apellidos »

Calificación: .....

Fecha: .....

A todos los profesores que se preocupan porque sus alumnos aprendan de verdad y hacen todo lo posible para conseguirlo...

"Sólo podemos ver poco del futuro, pero lo suficiente para darnos cuenta de que hay mucho que hacer"
Alan Mathison Turing

Resumen

BitTorrent, a pesar de ser una de las principales formas de compartir ficheros en red, su número de herramientas para la investigación es escaso. En este trabajo se realiza, en primer lugar, un estudio en profundidad del protocolo BitTorrent para después desarrollar una aplicación con el máximo número de funcionalidades encontradas. Esta aplicación tiene como objetivo ser útil a investigadores de modo que tengan todas las funcionalidades necesarias para cualquier investigación en este campo, desde encontrar los archivos .torrent que se están compartiendo de un fichero, a monitorizar uno o más torrents y recibir alertas por IP o rango.

Palabras clave: BitTorrent, herramienta, Protocolo BitTorrent, DHT, Golang.

Abstract

BitTorrent, despite being one of the main ways of sharing files on a network, its number of research tools is scarce. In this work, in the first place, an in-depth study of the BitTorrent protocol is carried out to later develop an application with the maximum number of functionalities found. This application aims to be useful to researchers so that they have all the necessary functionalities for any investigation in this field, from finding the .torrent files that are being shared from a file, to monitoring one or more torrents and receiving alerts via IP or range.

Keywords: BitTorrent, toolkit, BitTorrent Protocol, DHT, Golang.

Índice general

Índice de figuras

Índice de tablas

Índice de listados de código fuente

Lista de acrónimos

BEP	BitTorrent Enhancement Proposal
CLI	Denial of Service (Command-Line Interface)
DHT	Distributed sloppy Hash Table
DNS	Domain Name System
FTP	`File Transfer Protocol`(Protocolo de transferencia de archivos)
IEEE	Institute of Electrical and Electronics Engineers
P2P	`Peer-to-Peer` (Igual a Igual)
TCP	Transmission Control Protocol
TFG	Trabajo Fin de Grado
UDP	User Datagram Protocol
URL	Uniform Resource Identifier
uTP	uTorrent Transport Protocol

Glosario

bencoding	Codificación utilizada en el protocolo BitTorrent
BitTorrent	Protocolo diseñado para el intercambio de ficheros sobre una red P2P que se usa para distribuir gran cantidad de información por Internet
bloque	Unidades de datos que forman parte de una pieza.
Champion	Autor de un BEP y el responsable de intentar crear consenso en la comunidad sobre la idea
cliente-servidor	Diseño de software en el que hay proveedores de recursos llamados servidores y consumidores llamados clientes
crawler	También conocido como spider, programa que recorre una web o cualquier otra fuente de información para recolectar la información que se busca, desde enlaces a ficheros o hashes.
enjambre	En ingles: swarm. Lista de pares conectados por un fichero .torrent
handshake	Mensaje inicial entre dos clientes a forma de saludo
issue	En GitHub comentarios de los usuarios para informar de errores y organizar tareas
magnet	Link con la información básica que sustituye a los ficheros .torrent.
malware	Cualquier tipo de software malicioso
metadatos	Información almacenada en un torrent
metainfo	torrent
Napster	Servicio de distribución de música P2P.
peer	Cliente conectado a la red BitTorrent descargando y/o compartiendo un fichero
pieza	Unidad de datos apreciable más pequeña de BitTorrent. Varia en función del fichero torrent
scrape	Obtener estadísticas(número de seeds y peers) de un tracker en un enjambre

seeder	Usuario que tiene el fichero completo en su poder
string	Cadena de caracteres
suite	Conjunto de herramientas
swarm	enjambre
torrent	Fichero .torrent
tracker	Servidor que almacena los pares compartiendo un torrent al un nuevo cliente pide pares para conectarse

Capítulo 1

Introducción

Hay tres clases de intelecto: el primero discierne por sí; el segundo entiende lo que los otros disciernen, y el tercero no discierne ni entiende lo que los otros disciernen. El primero es excelente, el segundo bueno y el tercero inútil.

Nicolás Maquiavelo, El príncipe

1.1 Presentación

La red BitTorrent sigue siendo uno de los principales medios de distribución de ficheros online entre personas de todo el mundo. A pesar de que disminuyó considerablemente su utilización tras la salida de plataformas como Netflix [7], al dividirse el contenido en diferentes plataformas ha vuelto a crecer su uso, sobre todo, en regiones donde estas plataformas ofrecen menos contenidos que en otras zonas. Según un estudio de octubre del 2018 [8], en EMEA (Europa, Oriente Medio y África) BitTorrent ocupa el 31,73 % del tráfico de subida y es el protocolo de intercambio de ficheros más utilizado, a pesar de que en esta región se están implementando medidas para impedir las descargas de contenido con copyright.

Tron, una empresa especializada en Blockchain, publicó oficialmente el 23 de julio del 2018 la compra de BitTorrent [9]. El 4 de octubre de 2018 BitTorrent publicó que habían superado el millón de usuarios activos diarios en su nueva aplicación μTorrent Web publicada tan solo un mes antes [10]. Y lo más reciente, el 28 enero del 2019 se publicó el token BTT, basado en Tron, con el fin de incentivar las altas velocidades en la red. Esta nuevo token se implementó en la actualización del 9 de julio de μTorrent [11].

1.2 Motivación y objetivos

La motivación principal de este Trabajo Fin de Grado (TFG) es realizar una investigación sobre la red BitTorrent con el objetivo de crear una suite de herramientas lo más completa posible para investigadores de esta red. Dado que las funciones principales de estos investigadores no es descargar y compartir ficheros, es necesario estudiar y comprender el protocolo para poder sacar todas las posibilidades. Y, como motivación secundaria, se pretende crear una explicación corta y clara del proyecto BitTorrent y el funcionamiento de su protocolo.

Consideramos importante crear la suite de herramientas, ya que al ser BitTorrent uno de los principales métodos de transferencia de ficheros se utiliza también para compartir tanto ficheros ilegales como malware. Y a pesar de ser de los protocolos más utilizados hay muy pocos recursos disponibles para ayudar a las investigaciones tanto de las fuerzas y cuerpos de seguridad del estado, investigadores independientes o forenses de sistemas informáticos.

Para conseguir esto, el trabajo tiene tres objetivos claros.

1. Documentar claramente el estudio teórico del protocolo BitTorrent.
2. Identificar las posibles vías en la red BitTorrent para obtener información útil para investigaciones.
3. Desarrollar una aplicación que ayude a investigar y monitorizar la red BitTorrent tanto para investigadores como para cuerpos de seguridad del estado en diferentes ámbitos, como la persecución de delitos o el análisis forense de sistemas.

1.3 Organización y formato de la memoria

Esta memoria se organiza en 3 grandes capítulos. En el primero se realizará un estudio teórico del proyecto BitTorrent y su protocolo. En el segundo capítulo se detallará el proceso de desarrollo del programa que es el objeto principal de este TFG. Y, por último, en el tercer capítulo se explicarán las conclusiones a las que se ha llegado después del estudio teórico y del desarrollo, así como las líneas futuras del aplicativo.

Al proyecto se adjuntan dos apéndices, el primero es el manual de usuario de la aplicación (Apéndice A) y el segundo es la lista de herramientas y recursos utilizados (Apéndice B).

Para este TFG se ha utilizado la plantilla de LÁTEX del Departamento de Electrónica de la UAH [12]

Capítulo 2

Estudio teórico

Cualquier tecnología suficientemente avanzada es indistinguible de la magia.

Arthur C. Clarke

2.1 Introducción

El capítulo se estructura en 6 apartados. El primer apartado trata estado del arte actual sobre esta área. El siguiente es una explicación teórica básica de las redes P2P. Los dos siguientes tratan el proyecto BitTorrent y su protocolo. El penúltimo explica brevemente su implementación. Y, por último, las conclusiones del capítulo.

2.2 Estado del Arte

Sabiendo que es uno de los protocolos más utilizados en la actualidad y que sus dueños están innovando para que siga creciendo es sorprendente que existan tan pocas investigaciones sobre ello. Por ejemplo, buscando "BitTorrent" en IEEEXplore [1] encontramos actualmente (a septiembre del 2019) 738 publicaciones de conferencias, 82 de journals, 4 en libros y 1 artículo en acceso anticipado. Pero de este total solo 2 conferencias y el artículo en acceso anticipado es del 2019. En la figura 2.1 se puede ver la evolución de publicaciones en Institute of Electrical and Electronics Engineers (IEEE) y como poco después de crearse el protocolo comenzaron a aparecer conferencias y artículos sobre el tema de manera creciente hasta 2010-2011 que alcanzó su máximo y volvía a disminuir hasta 2019 que solo hay 3 publicaciones.

A pesar de esto cabe destacar el análisis del ecosistema BitTorrent en el centro de Asia del 2018 [13] y estudios más antiguos como el análisis del ecosistema general del 2012 [14] o la explicación utilizando medidas reales del 2013 [15].

Tabla 2.1: Tabla de la evolución de publicaciones en IEEEXplore [1] en los años impares hasta 2019.

	2005	2007	2009	2011	2013	2015	2017	2019
Conferences	14	45	89	106	66	29	20	2
Journals	0	3	7	10	15	8	5	1
Magazines	2	0	0	5	4	0	1	0
Books	0	0	1	0	0	1	0	0

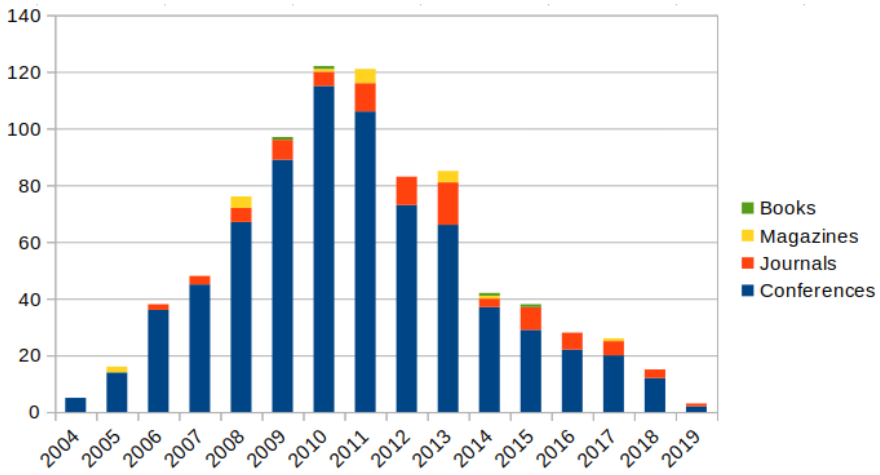

Figura 2.1: Evolución de publicaciones en IEEEXplore [1] hasta 2019.

2.3 P2P

Las redes P2P comenzaron a funcionar a finales de los años 90. La primera aplicación usada que implementaba esta arquitectura fue Napster, esta aplicación permitía compartir ficheros MP3. Sin embargo, esta red no era descentralizada, ya que utilizaba sus servidores como índice de ficheros y clientes. Por este motivo, cuando fueron denunciados por infracción de derechos de autor con solo cerrar los servidores centrales de Napster se cerró la red. Actualmente el protocolo de distribución de archivos P2P más importante es desde hace años BitTorrent, desarrollado por Bram Cohen. Protocolo en el que se centrará este trabajo.

El objetivo de las redes P2P es sustituir la distribución de ficheros centralizada que se utilizaba hasta el momento con el protocolo `File Transfer Protocol`(Protocolo de transferencia de archivos) (FTP). FTP es un protocolo de distribución de ficheros cliente-servidor, por lo que el servidor debe mandar una copia del fichero a todos los clientes que la soliciten necesitando un gran ancho de banda para satisfacer a todos los clientes a la vez. En una red P2P la carga está distribuida en la red. Como explica James F. Kurose [16]:

En la distribución de archivos P2P, cada par puede redistribuir cualquier parte del archivo que ha recibido a cualesquiera otros pares, ayudando de este modo al servidor a llevar a cabo el proceso de distribución.

La gran ventaja de las redes P2P es su auto escalabilidad. Para corroborar esto se utilizará el ejemplo del libro de Kurose [16], en el que se compara la distribución de un fichero en una red con N clientes utilizando una arquitectura cliente-servidor y una P2P. Suponiendo que todo el ancho de banda es dedicado a la descarga y subida del fichero y el límite de velocidad se encuentra únicamente en el ancho de banda de los equipos, podemos deducir la siguiente fórmula para calcular el tiempo de la descarga utilizando la arquitectura cliente-servidor:

$$D_{cs} = \text{máx}\{\frac{NF}{u_s}, \frac{F}{d_{\text{mín}}}\} \tag{2.1}$$

En la ecuación 2.1 podemos ver como el tiempo de distribución de cliente-servidor (D_{cs}) es igual al máximo tiempo de lo que tarda en transmitir N veces el fichero (NF) entre la velocidad de carga del servidor (u_s) y el tamaño total del fichero (F) entre la velocidad de descarga del cliente con menor ancho de banda ($d_{\text{mín}}$). En esta ecuación se puede ver que cuanto mayor sea N mayor será el tiempo total D_{cs}

y que este crecimiento en lineal.

Ahora calculemos la misma fórmula para una arquitectura P2P. Para hacer esto debemos suponer primero que solo el servidor tiene el fichero, debemos suponer también igual que con cliente-servidor, que no todos los clientes tienen el mismo ancho de banda y, por último, vemos que la velocidad total de subida es la suma de cada uno de los pares más la del servidor (que sería el seeder en este caso). Con esto nos queda la siguiente ecuación:

$$D_{P2P} = \text{máx}\{\frac{F}{u_s}, \frac{F}{d_{\text{mín}}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\} \tag{2.2}$$

Con estas dos ecuaciones podemos calcular un gráfico en función de N para ver la escalabilidad de la arquitectura cliente-servidor y la P2P (figura 2.2). Para que el gráfico fuese representativo se ha utilizado un F de 10000, un u_s de 100, un $d_{mín}$ de 50, un u_i de 5 y se ha pasado el tiempo a horas.

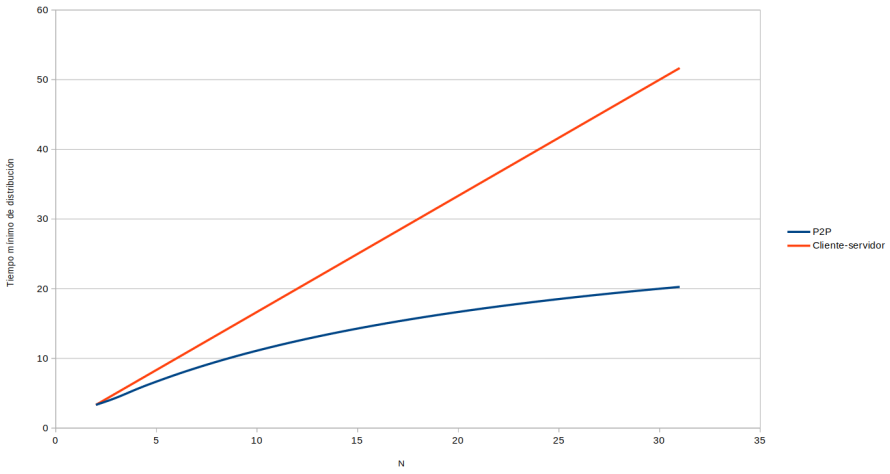

Figura 2.2: Tiempo mínimo de distribución

En una arquitectura P2P pura no hay ningún servidor centralizado, los pares se pueden desconectar y conectar a voluntad y los peers se conectan entre ellos directamente. Este no es siempre el caso de las redes P2P, por ejemplo, BitTorrent en su implementación inicial BTP/1.0 hace uso de trackers HTTP que utilizan los peers para conectarse entre ellos. Todo esto se explicará en los siguientes apartados.

2.4 BitTorrent

BitTorrent, Inc., (renombrada oficialmente a Rainberry, Inc. en 2017 [17]) es una empresa estadounidense fundada por Bram Cohen y Ashwin Navin en 2004. Es la empresa responsable del continuo desarrollo del protocolo BitTorrent y de dos de sus clientes más famosos, BitTorrent [18] y μTorrent [19].

A mediados del 2018 fue comprada por TRON Foundation [9] y el 21 de enero del 2019, Bram Cohen abandono la empresa [20] [21].

2.5 Protocolo BitTorrent

BitTorrent es un protocolo de distribución de ficheros P2P creado por Bram Cohen en 2001 [22] para la transmisión de ficheros de manera distribuida.

Para compartir ficheros utilizando este protocolo es necesario un cliente BitTorrent para conectarse a la red BitTorrent. Un cliente BitTorrent es un programa que implementa dicho protocolo. Hay un gran número de clientes conocidos, como por ejemplo BitTorrent [18] o μTorrent [19], desarrollados por BitTorrent, Inc.; o desarrollados por terceros como Vuze [23] o Transmission [24].

Este protocolo, a diferencia de otros protocolos P2P no tiene ninguna forma de buscar ficheros en la propia red, por lo que es necesario obtener el fichero torrent o el magnet de otra forma, por ejemplo, de la web. Pero esto es algo que explicaremos más adelante.

El proceso más básico de compartir ficheros en BitTorrent solo necesita de un tracker, un software que use el protocolo BitTorrent y el fichero .torrent asociado al fichero que se quiere compartir.

Lo primero que tenemos que comprender son los BEP. Una vez entendemos su funcionamiento y su uso podemos pasar al BEP más importante para el protocolo BitTorrent. En el BEP_0003 [25] se describe el fin y la implementación del protocolo BitTorrent en los que se conoce como BTP/1.0. En la imagen 2.3 se puede ver el esquema básico de una red en BitTorrent 1.0.

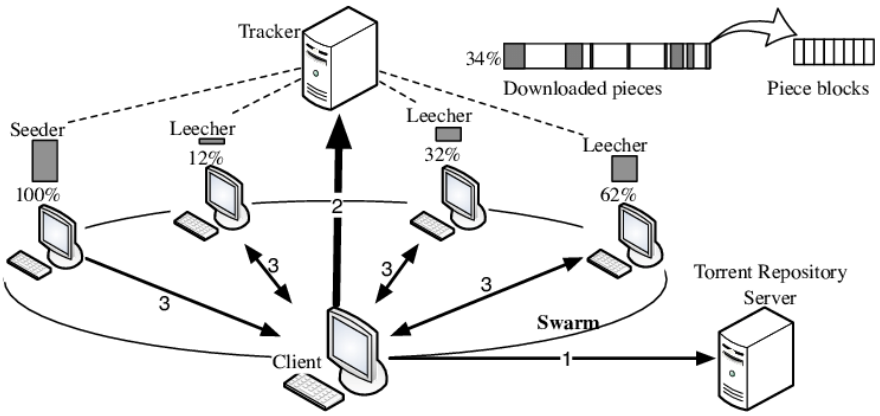

Figura 2.3: Arquitectura BitTorrent [2]

Además del BEP_0003 [25] Bram Cohen publicó un paper [26] en el que describe una serie de algoritmos para la selección de las piezas en las peticiones y para bloqueos necesarios para un rendimiento óptimo del protocolo. Entre estos algoritmos destaca el "Rarest First", en la que se comienza la descarga por las piezas que menos pares tengan. De esta manera si el peer que compartía el fichero original se va de la red y no queda nadie con todo el fichero hay muchas menos posibilidades de que alguna pieza no la tenga nadie.

2.5.1 BEP

Un BEP es un documento que proporciona información a la comunidad de BitTorrent o describe una nueva característica de los protocolos de BitTorrent. Este está definido en el BEP número 0001 y todos estos documentos se encuentran en GitHub [27] para el control de versiones. Estos documentos comenzaron en enero del 2008, 7 años después de la publicación oficial del protocolo, por lo que la mayoría de las BEP se redactaron mucho después de crearse. Este retraso produjo, por ejemplo, que la especificación de original BTP/1.0 del 2001 (BEP_0003 [25]) y la BTP/2.0 (BEP_0052 [28]) estén publicadas el mismo día.

[h] Es importante entender cómo funcionan los BEP, ya que todo el proceso de creación y posteriores modificaciones del protocolo BitTorrent están definidos en estos documentos.

Hay tres tipos de BEP:

- Standars Track BEP, que describe una extensión de los protocolos de BitTorrent o un cambio en estos.
- Informational BEP, que describe un problema de diseño de BitTorrent. Pautas generales o informa a la comunidad de BitTorrent, pero que no propone una extensión. Estos BEP no representan ningún tipo de consenso en la comunidad, por lo que no son realmente importantes ni para los usuarios ni para los desarrolladores.
- Process BEP, describe un proceso que rodea BitTorrent o propone un cambio en un proceso. Son como los Standars Tracks BEPs, pero en otros ámbitos que no son los protocolos. Estos BEP no incluyen cronogramas de lanzamiento, procedimiento, pautas, cambios en el proceso de decisiones y cambios en las herramientas o el entorno utilizado en el desarrollo de BitTorrent. Por lo que si es algo importante tanto para el usuario como para los desarrolladores.

2.5.1.1 Proceso de un BEP

El proceso de un BEP se puede simplificar en los siguientes pasos

1. Cada BEP debe tener un Champion o campeón, que será el autor del BEP y el responsable de intentar crear consenso en la comunidad sobre la idea. Si después de publicar una descripción de la idea como una issue en el proyecto de GitHub de BitTorrent, si después de discutirlo con la comunidad el Campeón considera que merece un BEP deberá crear una Pull Request al GitHub con el borrador del BEP en el formato correspondiente.
2. Si el editor de BEP lo aprueba, le asigna un número de BEP, lo etiqueta como uno de los tres tipos de BEP y le asigna el estado de borrador. En caso contrario el autor puede abrir un issue en GitHub para tratar el problema y posteriormente volver a intentarlo.
3. Los Standards Track BEPs consisten en dos partes, un documento de diseño y una implementación de referencia. Esta última parte no es necesaria inicialmente, pero si para que el BEP pase a estado final.
4. Una vez los autores den por completado el BEP deben informar al editor de BEP para que se revise.
5. Una vez aceptado se debe completar la implementación de referencia. Y después de ser aceptado este último paso el BEP se considerará Final.
6. Con el tiempo también se puede reemplazar. Por ejemplo, una API puede ser sustituida por una segunda versión siendo el mismo BEP.

Todo este proceso se puede ver en la figura 2.4 de una manera esquemática.

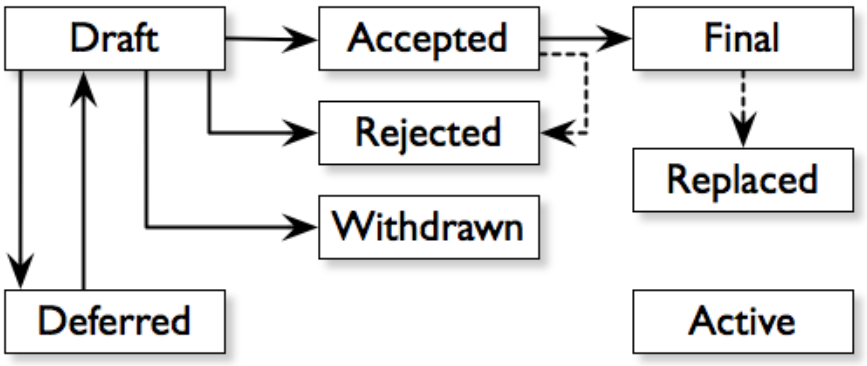

Figura 2.4: Flujo trabajo de un BEP [3]

2.5.1.2 Otros datos importantes para el BEP

Para estar seguros de que un BEP no va a ser rechazado automáticamente es necesario cumplir una seria de normas y recomendaciones que se enumeran a continuación.

1. Debe estar en el formato ".RST" detallado en el BEP_0002 [29]
2. Para el proceso de aceptación de un BEP es obligatorio que hayan existido discusiones en GitHub.
3. Debe ser una descripción clara y completa de la mejora propuesta
4. Debe ser funcional y haber sido probada en un enjambre de BitTorrent.
5. Es recomendable mostrar los análisis realizados de las simulaciones o experimentos detallados.
6. Debe ser de dominio público. Aunque esto no significa que el autor tenga que publicar la implementación software de alguna idea.

2.5.2 Bencoding

Los ficheros torrent se codifican utilizando un método especial llamado `Bencoding`. Esta codificación soporta 4 tipo[h]e datos, dos escalares (enteros y cadenas de caracteres) y compuestos (listas y diccionarios), definidos en el BEP_0003 [25]

- Los **enteros** se codifican con una "i" seguida del número en base 10 y la letra "e". Por ejemplo, el número 9 se codifica como i9e. Los números deben de estar sin ningún cero a la izquierda, por lo que el número i09e no es válido. A excepción del número 0 que sería i0e. Acepta también números negativos como i-9e.
- Las **cadenas de caracteres** se codifican con la longitud de la cadena, dos puntos y la cadena. Por ejemplo, "6:string". La cadena vacía se representa como "0:".
- Las **listas** se representan con la letra "l" seguida de sus elementos también codificados y al final la letra "e". Por ejemplo, la lista ["uno","dos","tres"] se codifica como l3:uno3:dos4:trese.
- Los **diccionarios** se codifican con una "d" seguido de la lista de clave-valor y acabado en "e". Por ejemplo, el diccionario "uno":1, "dos":2, "tres":3 se codificaría como d3:unoi1e3:dosi2e4:tresi3ee. La clave debe ser una cadena de caracteres y aparecer en orden.

2.5.3 Ficheros Metainfo

Los ficheros metainfo, también conocidos como ficheros .torrent o simplemente como torrents son diccionarios codificados con bencoding (2.5.2) que contienen las siguientes claves:

- **announce**: Tipo string que contiene la Uniform Resource Identifier (URL) del tracker.
- **announce-list** (Opcional): Es una lista de strings que contiene cada uno una URL a trackers de reserva (añadido en el BEP_0012 [30]).
- **comment** (Opcional): Es un string que puede contener un comentario del creador del torrent.
- **created by** (Opcional): Es un string que puede contener el nombre y la versión del programa utilizado para crear el fichero.
- **creation date** (Opcional): Es un string con la fecha de creación del torrent.
- **info**: Es una clave con un diccionario que contiene información de los ficheros a descargar. Este diccionario está formado por:
  - **name**: Cadena de texto en UTF-8 que contiene el nombre del fichero.
  - **piece length**: Entero que indica el número de bytes en cada pieza.
  - **pieces**: Es el string de la concatenación de los hashes SHA1 (20 bytes) de cada pieza.
  - **md5sum** (Opcional): Cadena de texto opcional que contiene el MD5 del fichero (no se utiliza en BTP/1.0).
  - **length o files**: Se utiliza **length** cuando el torrent solo contiene un fichero y su valor es un entero. Se utiliza **files** cuando el torrent contiene más de un fichero y su valor es una lista de diccionarios con el campo **length** para representar el tamaño del fichero, **path** para su ruta y opcionalmente también puede incluir md5sum.

2.5.4 Trackers

Un tracker o rastreador es una servidor que se utiliza en la red BitTorrent para comunicar diferentes pares. Cuando un peer quiere descargar un fichero se conecta al tracker que aparece en el torrent que quiere descargar para buscar peers que tengan el fichero o también lo estén descargando. Actualmente hay dos tipos:

- Los trackers HTTP, que inicialmente eran los únicos definidos.
- Los trackers User Datagram Protocol (UDP), que se implementaron en el BEP_0015 [31]

2.5.4.1 Trackers HTTP

Un tracker es un servidor HTTP definido inicialmente en el BEP_0003 [25]. Cada peer utiliza un tracker para conectarse al enjambre o swarm. Existen dos tipos de peticiones al tracker: announce y scrape. Announce es la petición normal como parte de la descarga de un fichero en la que un peer quiere obtener la lista de peers a los que conectarse, definida en el BEP_0003 [25]. Y scrape es una petición especial añadida en el BEP_0048 [32] para obtener estadísticas de uno o más ficheros concretos. Para ello, el peer debe realizar una petición HTTP GET al servidor con una serie de parámetros:

2.5.4.1.1 Petición

- **info_hash**: El hash SHA1 del fichero a compartir (tanto subir como descargar) que se encuentra en el fichero torrent. Codificado con URLencode.
- **peer_id**: Identificador de 20 bytes de largo que utiliza el cliente. Debe ser generado aleatoriamente para evitar colisiones. Codificado con URLencode. En este parámetro los diferentes desarrolladores introducen en algunos casos su identificador y el número de la versión del software como se comentará más adelante.
- **ip**: Campo opcional para incluir la IP pública del peer (IPv4 o IPv6) o su nombre Domain Name System (DNS).
- **port**: Puerto en el que el peer escucha, en BTP/1.0 no se especifica un rango de puertos, aunque el rango habitual es 6881-6889
- **uploaded**: Número total de bytes (en base 10) que el peer ha subido al enjambre desde que envió el evento "START" al tracker.
- **downloaded**: Número total de bytes (en base 10) que el peer ha descargado del enjambre desde que envió el evento "START" al tracker.
- **left**: Número total de bytes (en base 10) que el peer necesita para terminar la descarga del torrent.
- **numwant**: Campo opcional para indicar el número de peers que quiere recibir del tracker. Si no aparece, el tracker usara su valor por defecto.
- **event**: Campo opcional con tres opciones, si no se especifica el tracker considera que es una petición periódica sin evento.
  - **started**: Evento de inicio para comunicarse con un tracker.
  - **stopped**: Evento para indicar la finalización correcta de la comunicación con un tracker.
  - **completed**: Evento enviado al tracker cuando estando conectado completa la descarga del fichero.

2.5.4.1.2 Respuesta

Las respuestas de un tracker son diccionarios codificados con bencoding y la respuesta debe tener el tipo MIME "text/plain".

- **failure reason**: Clave opcional en texto plano indicando el error. Si esta clave aparece el diccionario no debe tener ninguna clave más.
- **interval**: Indica el tiempo que el peer debe esperar antes de realizar una consulta al tracker para el mismo torrent.
- **complete**: Clave opcional con el número de seeders.
- **incomplete**: Clave opcional con el número de peers descargando el fichero.
- **peers**: Diccionario codificado con la lista de peers a los que conectarse para descargar el fichero. En el BEP_0003 [25] las siguientes tres claves son obligatorias.
  - **peer id**: ID auto asignado del peer.
  - **ip**: IP pública del peer.

– **port**: Puerto a la escucha del peer.

Actualmente en la respuesta de los peers se utiliza el BEP_0023 [33] para devolver la lista de peers de una manera más compacta. Esto se consigue devolviendo un string en vez de un diccionario. En este modo cada peer usa 6 bytes, donde los 4 primeros son la IP y los 2 últimos el puerto usado. Para ello utiliza la variable `compact=1`.

```
GET /announce?peer_id=aaaaaaaaaaaaaaaaaaaa&info_hash=aaaaaaaaaaaaaaaaaaaa
&port=6881&left=0&downloaded=100&uploaded=0&compact=1
```

Actualmente la mayoría de trackers utilizan la forma compacta por defecto, por lo que si queremos la respuesta definida en BTP/1.0 se deberá utilizar la variable `compact=0`. en estos servidores si la repuesta buscada es compacta no es necesario especificarlo.

2.5.4.2 Trackers UDP

En la definición original de BitTorrent solo se especifican los tracker sobre HTTP, pero para reducir el número de paquetes que envía un tracker se implementaron los trackers UDP. Esta nueva definición aparece en el BEP_0015 [31], en este BEP el autor asegura que con UDP se reduce el tráfico un 50 %, tráfico que para un peer no es importante, pero si para el tracker.

Esta nueva implementación tiene 4 tipos de mensajes.

- **Connect**: Primer mensaje que debe enviar el cliente. En esta parte el tracker devuelve un connectionID al peer, que debe usar en las posteriores peticiones. Si el tracker recibe una petición sin connectionID o si es erróneo lo ignorará. El cliente puede reutilizar el identificador, pero un máximo de tiempo de un minuto desde que lo recibe. Por otro lado, el tracker debe aceptar el connectionID hasta 2 minutos después de enviarlo. Si expira el identificador, el peer deberá realizar el proceso de nuevo.
- **Announce**: Mensaje básico que se utilizaba en los tracker HTTP
- **Scrape**: Función para obtener estadísticas de trackers HTTP. Devuelve el número de seeders, leachers y completados
- **Errors**: Esta función la puede utilizar el tracker para avisar al peer que se ha producido un error junto a una explicación de este error.

2.5.4.3 Scrape

En el BEP_0048 [32] se añadió la petición `scrape` a los trackers HTTP. En esta petición un peer no quiere publicar que tiene o quiere el fichero, sino simplemente obtener las estadísticas de uno o varios ficheros, concatenando diferentes variables info_hashes.

```
http://tracker/scrape?info_hash=xxxxxxxxxxxxxxxxxxxx&info_hash=yyyyyyyyyyyyyyyyyyyy
```

La respuesta en JSON sería algo como lo siguiente:

```
{
  'files': {
    'xxxxxxxxxxxxxxxxxxxx': {'complete': 11, 'downloaded': 13772, 'incomplete': 19},
```

```
    'yyyyyyyyyyyyyyyyyyyy': {'complete': 21, 'downloaded': 206, 'incomplete': 20}
  }
}
```

2.5.4.4 Multitracker

En BTP/1.0 cada torrent tiene una clave `announce` en la que se incluye el tracker al que buscar. Esto es un problema si el tracker deja de estar disponible, ya que, aunque haya peers no se pueden encontrar entre ellos para iniciar las conexiones. Para solucionar esto y seguir utilizando tracker se implementó el BEP_0012 [30]. En este documento se especifica una nueva clave en torrent llamada `announce-list`. Esta clave contiene una lista de URL de diferentes trackers en los que buscar peers. Si esta clave está presente el cliente debe omitir la clave `announce`.

Estos tracker se organizan en niveles. Cada nivel se comprueba secuencialmente y se deben comprobar todas las URL antes de pasar al siguiente nivel. Además, cada tracker dentro de un nivel se procesa en orden aleatorio la primera vez. Después de eso si una conexión a un tracker es correcta se colocará el primero en la lista.

De este modo se puede añadir una lista como la siguiente:

```
d['announce-list'] = [ [ tracker1, tracker2 ], [backup1] ]
```

En esta lista el primer nivel contiene tracker1 y tracker2, por lo que en la primera comprobación se colocan en orden aleatorio y se comprobarán en cada `announce` antes de probar con backup1, ya que está en el segundo nivel.

2.5.5 Comunicación P2P o Peer Protocol

Cuando un cliente ya conoce una lista de peers y puertos que tienen el fichero que intenta descargar se debe conectar a ellos utilizando este protocolo. Para esto BitTorrent puede utilizar Transmission Control Protocol (TCP) o uTorrent Transport Protocol (uTP) (definido en el BEP_0029 [34]). Aquí hablaremos solo de TCP que aparece en la definición de BTP/1.0.

Las conexiones entre peers son simétricas, es decir, en estas conexiones los mensajes enviados en ambas direcciones tienen el mismo aspecto y los datos pueden fluir en ambas direcciones. Esto es así porque ambos peers pueden cumplir las mismas funciones.

Este protocolo tiene las siguientes características:

2.5.5.1 Handshake

Cualquier conexión con un peer debe abrir una conexión TCP al puerto que ha especificado el tracker y realizar el proceso de handshake. Este proceso se debe realizar antes de enviar cualquier otro tipo de dato al peer remoto.

Un handshake es una cadena de bytes con la siguiente estructura:

1. **Name Length**: Indica la longitud de la cadena de caracteres del nombre del protocolo. En BTP/1.0 es 19.
2. **Protocol Name**: Cadena “BitTorrent protocol” indicando el nombre del protocolo usado.

3. **Reserved**: 8 bytes reservados para futuras extensiones del protocolo.
4. **Info Hash**: Identificador hash (SHA1) del torrent.
5. **Peer ID**: ID del cliente.

Cuando un peer recibe este mensaje, contesta con el mismo mensaje cambiando el **peer ID** por el suyo. Una vez hecho esto los peers pueden comenzar a compartir sus piezas.

2.5.5.2 Estados

Las conexiones tienen dos bits de estado en cada extremo:

- `Choked`: Significa aquí bloqueado, indica que no se va a enviar ningún dato hasta que se elimine el bloqueo. Los motivos y las técnicas del bloqueo se explican más adelante.
- `Interested`: Cuando un peer marca a otro con este bit indica que este tiene partes del torrent que él no tiene.

La transferencia de datos tiene lugar cuando un lado está interesado y el otro no está bloqueado. El estado `interested` debe estar siempre actualizado e indicar si se deja de estar interesado, aunque este bloqueado. Las conexiones por defecto comienzan con el peer remoto con el bit choked activado y el interested deshabilitado.

2.5.5.3 Mensajes

Una vez hecho el handshake comienzan los mensajes entre los dos peers para mandarse información. Los mensajes con una longitud de 0 son Keepalives para mantener la conexión abierta. Normalmente estos mensajes se mandan cada 2 minutos. Los mensajes tienen el siguiente formato:

```
Longitud del mensaje | Tipo de mensaje | Datos
```

Todos los mensajes que no son Keepalives comienzan con un byte que indica el tipo del mensaje. Que tiene 9 opciones:

- `choke`: Se envía un mensaje de longitud 1 con ID 0, indicando que le ha bloqueado.
- `unchoke`: El peer envía un mensaje de longitud 1 con ID 1, indicando que le ha desbloqueado
- `interested`: El peer indica al otro que está interesado en alguna pieza suya, indicándolo con ID 2.
- `not interested`: Con el ID 3 el peer indica que no está interesado.
- `have`: El peer envía un mensaje de longitud 5, indicando el id 4 y el índice de la pieza que busca.
- `bitfield`: El peer envía un mensaje con el ID 5 en el que cada bit indica un bloque poniendo a 0 las que no tiene y a 1 las que tiene. Este mensaje tiene longitud variable dependiendo del fichero y es el primero que se envía en la conexión P2P, salvo que el peer no tenga ninguna pieza.
- `request`: envía un mensaje a un peer para pedirle un bloque. Este mensaje tiene una longitud de 13 bytes, en 4 bloque. El ID 6, el índice de la pieza, el comienzo y la longitud.

- `piece`: Mensaje con ID 7, que envía el índice de la pieza, el comienzo y a continuación el bloque que ha solicitado el peer.

- `cancel`: Mensaje con ID 8 que envía el índice de la pieza, el comienzo y la longitud (al igual que el `request`) pero con el objetivo de cancelar la petición, principalmente para eliminar peticiones duplicadas.

2.5.5.4 Peer_id

Como ya se ha visto el `peer_id` tiene 20 bytes de longitud que puede ser totalmente aleatorio, pero actualmente hay tres formas principales de crear el ID. Esta información aparece en el BEP_0020 [35] y en la wiki [36].

2.5.5.4.1 Estilo Mainline BitTorrent El cliente de BitTorrent Inc. originalmente utilizaba el ID: el carácter 'M', seguido de 3 números para la versión separados por guiones y finalizado con bytes aleatorios. Por ejemplo, para la versión 4.20.8 sería:

```
M4-20-8-
```

2.5.5.4.2 Estilo Azureus El estilo de Azureus forma los ID comenzando por un guion, seguido de dos caracteres para el software, cuatro dígitos para la versión, otro guion y acaba en números aleatorio.

```
-AZ2060-123456789012
```

Algunos clientes que utilizan este estilo son:

- `'AG' - Ares`
- `'A~' - Ares`
- `'AZ' - Azureus:`
- `'BP' - BitTorrent Pro (Azureus + spyware)`
- `'BT' - mainline BitTorrent (versions >= 7.9)`
- `'Bt' - Bt`
- `'LT' - libtorrent`
- `'lt' - libTorrent`
- `'qB' - qBittorrent`
- `'TR' - Transmission`
- `'UT' - µTorrent`

2.5.5.4.3 Estilo Shadow El estilo Shadow utiliza el siguiente formato: Una letra para el cliente, 5 caracteres para la versión y seguido de '—'. Si la versión no llega los 5 caracteres se rellenan hasta llegar a 5 con '-'. Por ejemplo, la versión 5.8.11 de Shadow seria.

```
S58B---
```

Algunos clientes conocidos que utilizan este estilo son:

- `'A' - ABC`
- `'Q' - BTQueue`
- `'S' - Shadow's client`
- `'T' - BitTornado`

2.5.5.5 Seguridad

A continuación, se listan las consideraciones de seguridad del protocolo BTP/1.0 [37].

- Los trackers utilizan el protocolo HTTP, enviando los datos en texto plano.
- Si se realiza un DoS a un tracker este dejará de dar servicio y los peers no se podrán encontrar entre ellos.
- No existe ningún tipo de autenticación con los trackers. El tracker utiliza el ID del cliente y su IP para identificarlo, por lo que podría realizarse una suplantación de identidad desde la misma IP.
- Los clientes utilizan DNS para hacer sus consultas, por lo que son vulnerables a ataques de DNS Spoofing.
- En los ficheros torrent aparecen los nombres de los ficheros, así como sus carpetas y rutas. Un atacante podría utilizar. Estas rutas para sobrescribir ficheros del equipo. Por lo que los clientes deben realizar las comprobaciones necesarias, como comprobar que no escapa de la capeta de descarga con '..' o intenta pisar ficheros utilizando la ruta absoluta del fichero.
- Los identificadores de los peer suelen mostrar, como ya se ha visto, un formato en el que aparece el software y su versión. Por lo que un atacante podría utilizar esa información para encontrar peers con software vulnerable.

2.5.6 DHT

Como se ha visto anteriormente, y sobre todo en el apartado 2.5.4, la red BitTorrent en su versión 1.0 no es completamente descentralizada ya que utiliza trackers para poner en contacto a los pares. El mayor inconveniente de esto es que si esos trackers dejan de estar disponibles la red entera deja de funcionar, da igual el número de usuarios activos que haya en ella, ningún usuario podrá descubrir nuevos pares para conectarse.

BitTorrent utiliza lo que llaman "distributed sloppy hash table" (DHT) para el almacenamiento de la información de los pares sin trackers (trackerless). De este modo a efectos prácticos cada par se convierte en un tracker de la red. Este nuevo protocolo, especificada en el BEP_0005 [38], está basado en Kademlia [39] e implementado utilizando UDP.

2.5.6.1 Funcionamiento DHT

La terminología aquí es importante. Un peer es un cliente o un servidor Escuchando en un puerto TCP que utiliza el protocolo BitTorrent. Un nodo es un cliente o servidor que escucha en un puerto UDP y utiliza en el protocolo UDP que implementa la DHT, La DHT está compuesta por nodos que almacenan la información de peers.

Cada nodo tiene un identificador (que debe ser único) en la red. La DHT utiliza lo que se llama "distance metric" para comparar dos ID de dos nodos para comprobar su proximidad. Cada nodo debe mantener una tabla de ruta con la información de contacto de un pequeño número de nodos. Esta tabla se vuelve más detallada según los IDs se "acercan" a los del ID del nodo. Por lo que un nodo conoce de muchos otros nodos con un ID "cercano" al suyo, pero almacena pocos nodos con IDs "lejanos".

En Kademlia la distancia entre los nodos (distance metric) es un XOR y el resultado se interpreta como un entero sin signo (valor absoluto), en el que cuanto menor sea el número más cercano. De este modo, la distancia entre a y b se calcula con la ecuación 2.3

$$|A \oplus B| \tag{2.3}$$

Cuando un nodo quiere encontrar los peers de un torrent, usa la distancia métrica para comparar el infohash del torrent con los IDs en su tabla de rutas. Después, contacta con los nodos más cercanos al hash que busca de esta tabla y les pide los peers que lo tienen o están descargando. Si el nodo al que le pregunta conoce peers de ese torrent le envía esa información en su respuesta. Mientras que si no tiene esta información le devuelve una lista de peers que tiene almacenados que sean "cercanos" a ese infohash. El nodo original repetirá este proceso buscando nodos cercanos hasta que no pueda encontrar más. Después de realizar la búsqueda, el cliente almacenará la información de contacto de los nodos con el ID más cercano al infohash del torrent.

La respuesta de las consultas de peers incluyen un valor "opaco" conocido como token. Este token se utiliza cuando un nodo quiere a anunciar al mismo peer que un peer que controla está descargando el fichero. Cuando se intenta anunciar un torrent, el nodo al que le envían la petición comprueba el token recibido con las IPs de los nodos que le han consultado. Esto se hace para prevenir ataques maliciosos anunciando otros peers al torrent. Como este token solo se devuelve al nodo que te lo ha enviado, su implementación no está definida. Los tokens deben aceptarse por un tiempo razonable desde que se han distribuido. La implementación de BitTorrent utiliza un hash SHA1 de la dirección IP concatenada con un secreto que cambia cada 5 minutos y los tokens se aceptan hasta 10 minutos después de ser enviados.

Dentro de la Tabla de rutas no todos los nodos son iguales. Algunos son los que se consideran "buenos" y otros no. La mayoría de los nodos de la DHT son capaces de consultar datos y recibir respuesta, pero no todos son apaches de responder a consultas de otros nodos. Es importante que cada tabla de ruta de cada nodo solo contenga nodos "buenos". Un nodo bueno es aquel que:

- ha respondido a una consulta nuestra en los últimos 15 minutos
- nos ha respondido a una consulta alguna vez y no ha hecho una en los últimos 15 minutos.

Después de 15 minutos de inactividad un nodo "bueno" se vuelve "cuestionable". Un nodo se vuelve "malo" cuando no responde múltiples consultas seguidas. Los nodos que conocemos como "buenos" tienen prioridad frente al resto.

La tabla de enrutamiento cubre todas los posibles ID (desde 0 a 2^{160}). Esta tabla está dividida en "buckets" que contienen una parte del rango. Estos "buckets" solo pueden contener 8 nodos en su interior

antes de llenarse. Cuando está lleno de nodos "buenos" conocidos no se puede agregar ningún nodo más, excepto en el caso de que nuestro ID se encuentre dentro del rango, en cuyo caso se remplaza por dos nuevos "buckets", cada uno con la mitad de rango del anterior y se dividen los nodos entre los dos. Para una nueva tabla con un solo "bucket", este se dividirá en dos, uno desde 0 hasta 2^{159} y el otro desde 2^{159} hasta 2^{160}.

Cuando un "bucket" está lleno de nodos "buenos", los nuevos nodos se descartan. Salvo que uno de los "buenos" se haya vuelto "malo". Para poder controlar los tiempos que lleva un nodo se añade un campo con la última modificación.

Con respecto a la implementación del protocolo BitTorrent, un torrent sin trakers no tiene la clave "annonunce", pero si tiene una clave "nodes". En está clave de deben añadir los nodos más cercanos al torrent o un nodo "bueno" como por ejemplo un nodo controlado por el creador del torrent.

2.5.6.2 Protocolo KRPC

El protocolo KRPC es un mecanismo RPC básico para enviar diccionarios codificados en bencoding sobre UDP. Estas comunicaciones tienen un solo paquete de consulta y uno de respuesta.

Un mensaje KRPC es un diccionario con 3 claves comunes para todos los mensajes y claves adicionales dependiendo del tipo de mensaje. Todos los mensajes tienen las siguientes claves:

- "t": clave con una cadena de caracteres que representan el ID de la transacción. Este número lo genera el nodo que hace la consulta y es devuelto en la respuesta.
- "y": clave con un solo carácter codificado que indica el tipo de mensaje, "q" para consulta, "r" para respuesta y "e" para error.
- "v": clave con la versión del cliente. Para este identificador se utilizan los 2 caracteres especificados en el BEP_0020 [35] Para el cliente seguido de dos caracteres para la versión.

En este protocolo hay 3 tipos de mensajes: consultas, respuestas y errores:

- `Consultas`: contiene dos claves adicionales: "q" que contiene una cadena de caracteres que contiene el nombre del método de la consulta, y "a" que contiene un diccionario que contiene los nombres de lo que consulta.
- `Respuestas`: contiene una clave adición: "r" que es un diccionario que contiene los valores que consulta el nodo.
- `Errores`: contiene una clave adicional: "e", que es una lista en la que el primer elemento es el código de error Y la segunda es una cadena de caracteres del error. Los códigos de error básicos van desde el 201 hasta el 204, en el que significan: error genérico, error del servidor, error en el protocolo o error en el método, respectivamente.

Dentro de las consultas existen cuatro tipos:

- `Ping`: es la consulta más básica, en este tipo la clave "q" = "ping". Un Ping es una consulta con un solo argumento, "id" con el valor del ID del emisor.

- `Find_node`: se utiliza para encontrar la información de contacto de un nodo dado su ID. Esta consulta tiene dos argumentos, "id" que contiene el ID del nodo al que se le pregunta, y "target" que contiene el ID del nodo por el que se pregunta. Cuando un nodo recibe esta consulta debe responder con una clave "nodes" y un string que contiene la información en modo compacto del nodo o de X nodos "cercanos".

- `Get_peers`: es la consulta para obtener peers de un infohash. Esta consulta tiene también dos argumentos, "id" con el identificador del nodo que consulta e "info_hash" del que queremos peers. Si el nodo tiene información sobre peers para ese torrent devuelve una clave "values" con una lista de strings en formato compacto con la información de los peers. Si no tiene información sobre peers devolverá una clave "nodes" con un número de nodos "cercanos" al infohash consultado. En ambos casos se devuelve también una clave "token" para el "announce_peer" que pueda hacer posteriormente.

- `Annonunce_peer`: esta consulta se utiliza para comunicar a un nodo que se está descargando el torrent. Esta consulta tiene 4 argumentos. "id" que igual que en el resto contiene el valor del ID del emisor. "info_hash" del fichero descargando, "port" con el puerto a la escucha y "token" con el valor que nos ha devuelto el nodo en la consulta de get_peers. Esta consulta tiene, además, un argumento adicional "implied_port" a modo de booleano (0 o 1). Si está a 1 el servidor debe omitir el puerto del parámetro "port" y utilizar el puerto origen del paquete UDP. Esto es útil para peers detrás de un NAT que no conoces su puerto externo.

La información de contacto de los peers (IP, puerto) se codifica como un string de 6 bytes. Este método se conoce como información IP/puerto compacto [33]. Los primeros 4 bytes para la IP y los dos últimos para el puerto.

La información de contacto de otros nodos se codifica como un string de 26 bytes de longitud. Conocido como información del nodo compacta. Que está formada por el ID del nodo concatenado de 4 bytes para la IP y 2 para el puerto.

2.5.7 Otros BEPs

Aparte de todos los anteriores, hay un gran número de BEPs más con muchas modificaciones del protocolo. De entre todas estas extensiones es importante destacar el BEP_0051 [40] en el que se define una consulta para indexar infohashes en la DHT, el BEP_0052 [28] en el que se especifica la versión 2 del protocolo BitTorrent y el BEP_0042 [41] en que se especifica la extensión de seguridad para DHT.

2.5.7.1 DHT Infohash Indexing

Esta extensión permite a los nodos de la DHT obtener una muestra de infohashes que otros nodos tienen almacenados [40]. Está nueva consulta se hizo para poder recorrer toda la DHT obteniendo los hashes que tienen los nodos almacenados. Esto se puede realizar también de forma pasiva conectado a la DHT y monitorizando las consultas de get_peers pero es menos eficiente.

El nodo envía la consulta "sample_infohashes" junto a los parámetros básicos que hemos visto en "find_node"("id" y "target", sección 2.5.6.2), a la que el servidor responde con una clave "interval" que indica al nodo el tiempo que debe esperar antes de preguntar de nuevo, los nodos cercanos al "target" de la consulta, un número indicando el número de infohashes enviados y un string formado por múltiples infohashes seguidos de los que tiene pares almacenados.

2.5.7.2 The BitTorrent Protocol Specification v2

En el 2008 Bram Cohen publicó la segunda versión del protocolo en el BEP_0052 [28]. Aquí no vamos a explicar las modificaciones completas, ya que es una mejora frente a la versión original y ya se han ido comentando las modificaciones más importantes. Aunque se publicó en 2008 sí que es importante para comprender como funcionan los BEPs que sigue con el estado de Borrador. Con esto podemos ver cómo, aunque un BEP lleve mucho tiempo y esté implementado en la mayoría de clientes, no tiene por qué estar completamente aceptado.

2.5.7.3 DHT Security extension

En enero del 2014 se creó un BEP [41] para intentar mitigar ciertos ataques a la DHT. Principalmente el ataque que pretende mitigar este BEP, es el de lanzar 8 o más nodos con IDs cercanos a un infohash específico para convertirse en los nodos principales de ese torrent. Un atacante con este método podría monitorizar todo el tráfico hacia ese fichero e incluso bloquear a los pares.

Este BEP no es útil en este proyecto, pero sí que es muy interesante para comprender este ataque a la DHT y como mitigarlo.

2.6 Implementaciones BitTorrent

El protocolo BitTorrent tiene un gran número de clientes y bibliotecas implementados, a continuación, se enumerarán algunas de ellas.

2.6.1 Clientes

- `BitTorrent` [18]
- μ`Torrent` [19]
- `Vuze` [23]
- `Transmission` [24]

aparte de estos clientes, es muy recomendable el TFG de David Gracia Celemendi sobre el desarrollo de una aplicación BitTorrent [42].

2.6.2 Bibliotecas

- `libtorrent` [43]
- `Ttorrent` [44]
- `torrent (Anacrolix)` [45]
- `MonoTorrent` [46]

2.7 Conclusiones

Una vez conocemos el protocolo internamente y entendemos su funcionamiento para compartir ficheros, podemos profundizar un poco más y analizar la estructura para ver que se puede conseguir desde el punto de vista de un investigador. Es decir, debemos saber que información podemos obtener teniendo un torrent o un fichero utilizando los métodos que nos permite la definición del protocolo.

Está claro que lo más básico es la descarga de ficheros junto a la creación de nuevos torrents para compartir. Aparte de esto, analizando el protocolo podemos ver que se puede monitorizar un torrent y guardar quien está descargando un fichero en concreto. Podemos también recorrer la DHT con el método sample_infohashes para crear una base de datos de hashes de ficheros en la red y posteriormente consultar esos hashes a la DHT para obtener sus metadatos. Por último, podemos juntar la creación de ficheros con la búsqueda en la DHT para, dado un fichero local, encontrar sus diferentes ficheros torrent en la red. Todo esto se desarrollará en el siguiente apartado.

Capítulo 3

Desarrollo

El que no coloca los cimientos con anticipación podría colocarlos luego si tiene talento, aún con riesgo de disgustar al arquitecto y de hacer peligrar el edificio.

Nicolás Maquiavelo, El príncipe

3.1 Introducción

En este capítulo se van a explicar los objetivos del desarrollo, y el proceso de implementación, así como las partes más importantes del código.

3.2 Objetivo del desarrollo

Para la realización de este proyecto se ha decidido utilizar Golang como lenguaje de programación junto a PostgreSQL como base de datos. La idea original es tener la base de datos en un servidor específico y poder tener tantos clientes como hagan falta ejecutando la aplicación en distintos dispositivos pudiendo tener una base de datos conjunta. Así un cliente podría estar ejecutando el crawler para buscar nuevos ficheros, otro monitorizando alertas y otro realizando diferentes consultas y búsquedas. Para ello, la herramienta contará con las siguientes funcionalidades en una interfaz de tipo CLI:

1. Creación de un fichero BitTorrent a partir de cualquier fichero o carpeta
2. Descarga de ficheros de la red BitTorrent desde un fichero, su magnet o su hash.
3. Inserción de la información de ficheros BitTorrent en la base de datos directamente desde un fichero o a mano.
4. Modo automático en el que recorrerá la red en busca de ficheros para almacenarlos en su base de datos (crawler).
5. Buscaqueda de un fichero en la red BitTorrent de la manera inversa a la natural de la red.
6. Un modo demonio para la monitorización de los clientes que descargan una serie de ficheros que le digamos y nos pueda alertar cuando aparezca una IP o una red especifica.

7. Mostrar información de la base de datos con consultas personalizables.

De todas estas funcionalidades, se quiere destacar sobre todo la posibilidad de encontrar un fichero local en la red, ya que esta funcionalidad implica dar la vuelta al proceso original de BitTorrent. Utilizando esta funcionalidad se podrá encontrar todos los hashes que utiliza la red internamente de un fichero para poder descargarlo de nuevo o monitorizarlo.

3.3 Tecnología utilizada

1. Golang [47]
2. PostgreSQL [48]
3. Docker [49]

3.4 Plan de trabajo

Las etapas que se van a seguir están basadas en los tres objetivos originales. A continuación, se enumeran sus fases internas y el tiempo esperado en cada objetivo.

1. Identificar (1 mes)
   (a) Investigación del funcionamiento de la red BitTorrent
   (b) Investigación de los protocolos de comunicación utilizados para replicarlos.
   (c) Identificar mejores bibliotecas de código para ayudar en el desarrollo de la aplicación.
2. Desarrollar (2 meses)
   (a) Implementar las funcionalidades más sencillas (descarga y creación de un fichero)
   (b) Diseño de la Base de datos
   (c) Creación del crawler
   (d) Búsqueda de ficheros de una manera inversa
   (e) Monitorización de ficheros
   (f) Creación de alertas
   (g) Demonio para las alertas
3. Facilitar (2 semanas)
   (a) Crear la documentación de uso.

3.5 Desarrollo del programa

3.5.1 Bibliotecas de Go utilizadas

1. Cobra [50]
2. torrent (Anacrolix) [51]

3. dht (Anacrolix) [52]
4. tagflag (Anacrolix) [53]
5. bencode-go [54]
6. pq [55]
7. TOML [56]
8. Goscrape [57]

3.5.2 Base de datos

Para el diseño de la base de datos se ha utilizado Toad Data Modeler. La imagen del diagrama de este programa puede verse en la captura 3.1 junto a parte del código SQL 3.1 y la función de alerta en el código **??**

3.5.2.1 Diagrama de la base de datos

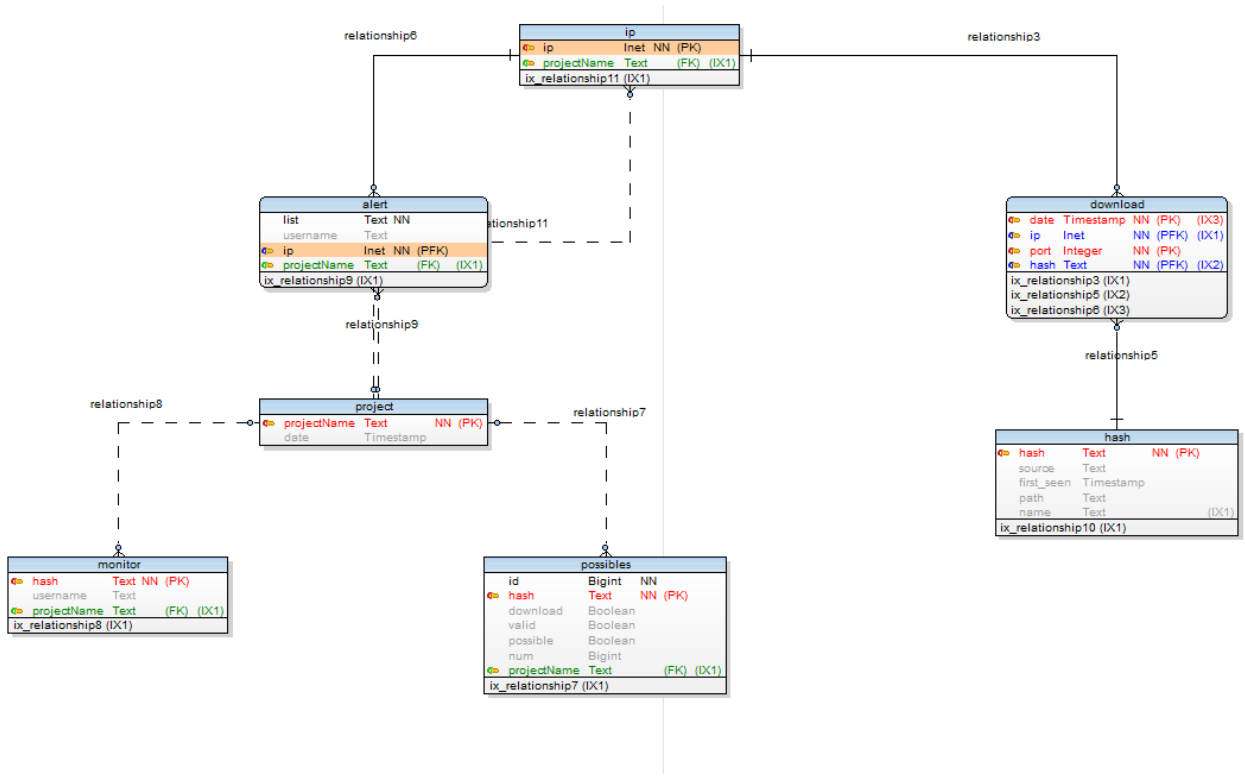

Figura 3.1: Diagrama de la base de datos con Toad Data Modeler [4]

3.5.2.2 Código SQL de la base de datos

Listado 3.1: Ejemplo del código SQL de la base de datos. Fichero **sql.sql**

```
-- Create tables section -------------------------------------------------

-- Table hash

CREATE TABLE hash(
 hash Text NOT NULL,
 source Text,
 first_seen Timestamp,
 path Text,
 name Text
)
;

-- Create indexes for table hash

CREATE INDEX IX_Relationship10 ON hash (name)
;

-- Add keys for table hash

ALTER TABLE hash ADD CONSTRAINT hashKey PRIMARY KEY (hash)
;

-- Table possibles

CREATE TABLE possibles(
 id Bigint NOT NULL,
 hash Text NOT NULL,
 download Boolean,
 valid Boolean,
 possible Boolean,
 num Bigint,
 "projectName" Text
)
;

-- Create indexes for table possibles

CREATE INDEX IX_Relationship7 ON possibles ("projectName")
;

-- Add keys for table possibles

ALTER TABLE possibles ADD CONSTRAINT possibleKey PRIMARY KEY (hash)
;
```

3.5.3 Explicación código

A continuación se explican las principales funciones del programa desarrollado, tanto para el Denial of Service (Command-Line Interface) (CLI) como para las diferentes funcionalidades del programa.

3.5.3.1 CLI

Para la ayuda de la implementación del CLI se ha utilizado la biblioteca Cobra [50]. Esta biblioteca permite una interfaz de tipo CLI que permite utilizar órdenes para cada funcionalidad y "FLAG"s globales o para cada orden para modificar el comportamiento del comando a ejecutar. Para utilizar esta forma de interfaz con esta biblioteca se debe crear una carpeta llamada "cmd" con un fichero por cada orden del programa (recomendable, pero no obligatorio) junto a un fichero "root.go" con la configuración general

y la funcionalidad del programa sin parámetros. Aquí se definen las "FLAGs" globales de la aplicación. Es este caso:

1. –**verbose**: Modo verbose para mostrar más información por pantalla.
2. –**debug**: Modo debug para mostrar aún más información que el modo verbose.
3. –**cfgFile file**: Parámetro con un fichero .toml con la configuración para el acceso a la base de datos. Por defecto viene el fichero **configFile.toml**.

A continuación se muestran las declaraciones de las "FLAGs" globales en el fichero **cmd/root.go** (código fuente 3.2).

Listado 3.2: Declaración de "FLAGs" del fichero **root.go**

```
  rootCmd.PersistentFlags().BoolVarP(&verbose, "verbose", "v", false, "verbose output")
  rootCmd.PersistentFlags().BoolVarP(&debug, "debug", "d", false, "debug output")
  rootCmd.PersistentFlags().StringVarP(&cfgFile, "cfgFile", "c", "configFile.toml", "Config
      file to DB (host, port, user, password)")
```

El resto de órdenes deben añadirse a este root. Para ello utilizan la función "AddCommand". En el código 3.3 se puede ver como en la función init se ejecuta esa función pasándole por parámetro la nueva orden que se define más arriba y a continuación se añaden las "FLAGs" para esa orden.

Listado 3.3: Declaración de la orden y definición de "FLAGs" del fichero **cmd/find.go**

```
var findCmd = &cobra.Command{
  Use:   "find <path to file>",
  Short: "Find the file in Bittorrent network",

[...]

func init() {
  rootCmd.AddCommand(findCmd)

  findCmd.PersistentFlags().StringVarP(&tracker, "tracker", "", "", "<not implemented> tracker
      ")
  findCmd.PersistentFlags().BoolVarP(&noadd, "no-add", "n", false, "<not implemented> no add
      to database")
  findCmd.PersistentFlags().IntVarP(&timeout, "timeout", "t", 5, "timeout download in minutes"
      )
  findCmd.PersistentFlags().IntVarP(&typeF, "mode", "m", 2, `Opciones
  0) Trackers
  1) Trackers + DHT, Bool valid
  2) Trackers + DHT + Download valid, Bool valid and download
  3) Trackers + DHT + Download possible, Bool valid and download
  `)
}
```

3.5.3.2 Manipulación fichero torrent

Las funcionalidades más básicas del programa son la de manipulación de ficheros torrent. Entre estas funcionalidades se encuentra la creación del fichero torrent, la descarga de un torrent dado (tanto de un hash como de un torrent) y la obtención del torrent asociado a un hash. Para ver todas las opciones de cada función se recomienda ir al Apéndice A

3.5.3.2.1 Crear y descargar un torrent La función **create** llama a Createtorrent (que veremos más adelante que se utiliza también en la función **find**) esta función utiliza las bibliotecas de Anacrolix, torrent [51] y tagflag [53]. Esta función está basada en el ejemplo de crear un torrent de Anacrolix [58]. En el código 3.4 podemos ver sus "FLAGs" y en el caso del "piecesize" podemos ver su valor por defecto.

Listado 3.4: Declaración de la orden y definición de "FLAGs" del fichero **cmd/create.go**

```
rootCmd.AddCommand(createCmd)

createCmd.PersistentFlags().StringVarP(&outfile, "outfile", "o", "", "Save the generated .
    torrent to this filename")
createCmd.PersistentFlags().StringVarP(&file, "file", "f", "", "File to create")
createCmd.PersistentFlags().Int64VarP(&piecesize, "piecesize", "p", 2*1024, "Save the
    generated .torrent to this filename")
createCmd.PersistentFlags().StringVarP(&tracker, "tracker", "t", "", "Save the generated .
    torrent to this filename")
createCmd.PersistentFlags().StringVarP(&comment, "comment", "m", "", "Save the generated .
    torrent to this filename")
```

Al igual que en la creación, para la descarga se ha utilizado el ejemplo de Anacrolix [58]. Este ejemplo se ha modificado para hacerlo más simple, ya que la versión original permite la descarga de múltiples ficheros, pero para esta funcionalidad lo imprescindible es descargar un fichero. En mejoras futuras se implementará la gestión de diferentes ficheros simultáneos.

3.5.3.2.2 Obtener el torrent dado un hash En esta función el programa debe descargar el fichero .torrent con los metadatos de un hash. Esta función es básicamente la primera fase de una descarga de un hash. Ya que lo primero que debe hacer antes de comenzar a descargar es obtener los metadatos de un par con el fichero. Por lo que la función está tambien basada en [58] pero una vez obtenida los metadatos se cancela la descarga y se guardan en un fichero .torrent. A continuación (código 3.5) se muestra la función **getTorrent**:

Listado 3.5: Función para obtener el .torrent de un hash dado. Función en el fichero **cmd/getTorrent.go**

```
func getTorrent(magnet string) error {
  cfg := torrent.NewDefaultClientConfig()
  cl, err := torrent.NewClient(cfg)
  if err != nil {
    log.Fatalf("error creating client: %s", err)
  }
  http.HandleFunc("/torrent", func(w http.ResponseWriter, r *http.Request) {
    cl.WriteStatus(w)
  })
  http.HandleFunc("/dht", func(w http.ResponseWriter, r *http.Request) {
    for _, ds := range cl.DhtServers() {
      ds.WriteStatus(w)
    }
  })

  t, err := cl.AddMagnet(magnet)
  if err != nil {
    log.Fatalf("error adding magnet to client: %s", err)
  }
  <-t.GotInfo()
  mi := t.Metainfo()
  t.Drop()
  fmt.Println("Name: " + t.Info().Name + "\n")
  files := t.Info().Files
  fmt.Println("FILES: ")
  for _, element := range files {
    fmt.Printf("\tName: %v , size: %d\n", element.Path, element.Length)
  }

  if path != "" {
    f, err := os.Create(t.Info().Name + ".torrent")
    if err != nil {
      return err
    }
    defer f.Close()
    err = bencode.NewEncoder(f).Encode(mi)
    if err != nil {
      return err
    }
  }
  return nil
}
```

3.5.3.3 Alertas y monitorización

Las alertas tienen dos órdenes asociadas en el mismo fichero, una para añadir y otra para borrar. El funcionamiento de esta orden es básicamente almacenar la IP o el rango para la alerta en la base de datos. Tiene, además, una "FLAG" para especificar el nombre del proyecto asociado a esa alerta. Esta funcionalidad se muestra al ver la alerta y se utiliza a la hora de monitorizar ("daemon") para seleccionar un único proyecto.

Cuando se llama a esta orden el programa se conecta a la base de datos y ejecuta la siguiente función (código 3.6). Del mismo modo se realiza la función de eliminar alertas ejecutando el borrado de la base de datos.

La monitorización se encuentra en el fichero **cmd/monitor.go** y utiliza la misma aproximación que las alertas. Al añadir un hash a monitorizar se inserta en la base de datos y al eliminar se elimina. Podemos ver la función de insertar en la base de datos en el código 3.7

Listado 3.6: Función para la inserción de una IP en la base de datos del fichero **utils/database.go**

```
//InsertIP inserts a IP or range in the database db with the proyectName
func InsertIP(db *sql.DB, debug bool, verbose bool, ip string, projectName string) error {

  sqlStatement := `
    INSERT INTO ip(ip, projectName)
    VALUES($1,$2)
    RETURNING ip`
  id := ""

  if debug {
    log.Println("Insert ip  ", ip, projectName)
  }

  err := db.QueryRow(sqlStatement, ip, projectName).Scan(&id)
  if err != nil {
    return err
  }
  if debug {
    log.Println("New record ID is:", id)
  }
  return nil
}
```

3.5.3.4 Demonio

La función **"daemon"** tiene 2 partes que realiza simultáneamente. La primera es ponerse a la escucha de alertas de la base de datos y mostrarlas por pantalla cuando llega una notificación de una IP o rango añadido como alerta se ha detectado. Para ello la base de datos tiene una función para notificar cuando añade una IP o rango que está en la tabla de alertas. Está función se puede ver en el código 3.8.

Listado 3.7: Función para la inserción de un hash en la base de datos para la monitorización. Código del fichero **utils/database.go**

```
import (
  "database/sql"
  "log"
)

//InsertMonitor inserts a hash in the database for monitor in daemon mode.
func InsertMonitor(db *sql.DB, debug bool, verbose bool, hash string, username string,
    projectName string) error {

  sqlStatement := `
    INSERT INTO monitor(hash, username, projectName)
    VALUES($1,$2,$3)
    RETURNING hash`
  id := ""

  if debug {
    log.Println("Insert monitor ", hash, username, projectName)
  }

  err := db.QueryRow(sqlStatement, hash, username, projectName).Scan(&id)
  if err != nil {
    return err
  }
  if debug {
    log.Println("New record ID is:", id)
  }
  return nil
}
```

La segunda parte de la función obtiene los pares de los hashes de la tabla monitor y los añade a la base de datos. Cada vez que se inserta una nueva IP se ejecuta la función de **??** para notificar si esa IP está en la lista de alertas o está en un rango en esa lista.

3.5.3.5 Crawl

El crawler o spider es una de las funciones más importantes de la aplicación. Esta función se utiliza para buscar en la DHT todos los hashes posibles utilizando el método sample_infohashes. Esa función tiene una "FLAG" para establecer el número de hilos con los que se quiere ejecutar esta funcionalidad. Es importante destacar que un gran número de hilos puede ocupar todo el ancho de banda o incluso bloquear el router, ya que hace consultas a un gran número de IPs diferentes y puede llenarse la memoria de éste. Al igual que desde linux se puede superar el número de ficheros abiertos y no poder crear más conexiones y fallar. Para solucionar esto es necesario ejecutar cambiar el límite de ficheros abiertos simultáneamente (ulimit).

Esta función crea dos canales, uno para la cola y otro para los hashes que encuentra, "data" y "hashes". A continuación, busca nodos para iniciar la búsqueda y crea X número de hilos para buscar en la DHT y X/250 hilos para insertar los hashes que encuentran en la base de datos (modelo productor-consumidor). En Go estos canales son auto gestionados para los multi hilos, por lo que no hay que hacer nada más con ellos salvo insertar y obtener los datos. la función principal se puede ver en el código 3.9.

Los hilos que recorren la DHT utilizan la función "sampleInfohashes" para obtener tanto los nodos que devuelve los servidores como la lista de hashes (si soportan la llamada sample_infohashes). Los nuevos nodos que devuelve se almacenan en el canal "data" y los hashes en "hashes". El código de la consulta de infohashes se encuentra en el código 3.10

Listado 3.8: Función para las notificaciones de alertas en el demonio. Parte del código del fichero **utils/AlertaMonitor.go**

```
import (
  "database/sql"
  "fmt"
  "time"
)

//MonitorAlert create a listener for the PostgreSQL database for alerts and print them.
func MonitorAlert(configfile string, debug bool, verbose bool, projectName string) {

  config, err := GetConfig(configfile, debug, verbose)
  if err != nil {
    panic(err)
  }

  conninfo := "dbname=" + string(config.Dbname) + " user=" + string(config.User) + " password=
      " + string(config.Password+" sslmode=disable")

  _, err = sql.Open("postgres", conninfo)
  if err != nil {
    panic(err)
  }

  reportProblem := func(ev pq.ListenerEventType, err error) {
    if err != nil {
      fmt.Println(err.Error())
    }
  }

  listener := pq.NewListener(conninfo, 10*time.Second, time.Minute, reportProblem)
  err = listener.Listen("events")
  if err != nil {
    panic(err)
  }

  fmt.Println("Start monitoring PostgreSQL...")
  for {
    waitForNotification(listener, verbose, projectName)
  }
}
```

3.5.3.6 Buscar ficheros en la red (find)

La función **find** es la funcionalidad más importante de todo el desarrollo de la aplicación. Necesita un argumento que es el "path" a un fichero y el programa busca sus posibles hashes en la red BitTorrent. Actualmente hay 4 modos de búsqueda.

1. El **modo 0** es el más simple y rápido, ya que solo busca los posibles ficheros en la base de datos local y una lista de trackers por defecto. Para hacer esto se genera una lista de posibles hashes del fichero utilizando diferentes tamaños para las piezas. Por defecto se calculan 20 hashes con tamaños desde 2^{10} a 2^{30} bytes (desde 1 k hasta 1024 megas).

2. El **modo 1** utiliza lo mismo que el modo 0 peor además utilizar la DHT para buscar el fichero.

3. El **modo 2** es el modo por defecto y utiliza todo lo anterior, pero añade la descarga de los fichero que ha encontrado en los trackers o en la DHT.

4. El **modo 3** es el modo más agresivo, ya que intenta descargar todos los ficheros, aunque no los haya encontrado en la DHT o en los trackers.

Para todos estos modos el primer paso es el mismo, ya que hay que calcular todos los posibles hashes con diferentes tamaños de piezas. Para ello se utiliza un método muy parecido al de crear un torrent, pero se le añade una paralelización para tardar menos. Para ello se utilizan los Channels de Go. Esta función inicial que crea los hilos y crean los hilos se puede ver en el código 3.11.

Una vez hecho esto en el **modo 1** y en el **modo 2** realizan las consultas a una lista de trackers utilizando la biblioteca [57] y buscan en la DHT con la función SearchDHT que crea un hilo por cada hash y llama a los workers (hilos) para hacer las consultas paralelamente. En este proceso se utiliza la funcion findNodes (código 3.12) para conectarse a la red DHT.

También utiliza la función getPeers (código 3.13) para buscar cada uno de los hashes. esta función utiliza las llamadas básicas de la DHT para obtener hashes que se han explicado antes, pero cuando un nodo devuelve que conoce un fichero y nos devuelve pares comprobamos que no nos está dando siempre datos sin importar el hash por el que preguntamos. Para ello se le pregunta por 3 hashes más inventados para comprobar si en esos nos devuelve 0 pares como debería o siempre contesta y no nos sirve.

3.5.3.7 Mostrar información de la base de datos

Además de esto, al ser una aplicación completamente CLI, se ha añadido una pequeña función llamada **show** para realizar consultas simples a la base de datos. Esta orden tiene una serie de subórdenes para cada una de las tablas de la base de datos y una extra llamada **count** que devuelve el número de hashes almacenados en la base de datos:

1. **alert**. Muestra la tabla de alertas. Se puede buscar una IP especifica con la "FLAG" –**ip**.
2. **hash**. Muestra la tabla de hashes. Se puede filtrar por un hash específico para comprobar si está con –**hash** o por una fuente con la "FLAG" –**source**.
3. **ip**. Muestra la tabla de ips. Se puede buscar una IP especifica con la "FLAG" –**ip**.
4. **monitor**. Muestra la tabla de ficheros monitorizados. Tiene un "FLAG" –**user** para buscar por usuario, por defecto se llama **default**.
5. **possibles**. Muestra la tabla de possibles. Se puede filtrar por un hash específico para comprobar si está con –**hash**.
6. **project**. lista los proyectos de la base de datos. Se puede filtrar por nombre del proyecto **projectName**.

Todas ellas tienen una "FLAG" –**where** para especificar el where de la consulta a la base de datos.

3.5.4 Problemas encontrados

Durante el desarrollo ha habido una serie de problemas que se han podido solventar o evitar. Sobretodo las complicaciones a la hora de hacer pruebas controladas del protocolo. Esto es debido a que únicamente se puede probar en la red y no se puede saber cuando va a continuar una IP que estaba compartiendo un fichero porque se puede ir en cualquier momento.

A pesar de las pruebas sigue habiendo algun problema con alguna ejecucion del programa pero que ocurre en situaciones poco comunes y que no se han podido replicar a la hora de buscar el problema. Se espera que con tiempo y más pruebas se solucionen todos los pequeños problemas que hay ahora mismo.

3.6 Conclusiones

Una vez desarrollado todas las funcionalidades explicadas en este apartado, es importante destacar que no está todo el trabajo terminado, como se verá en el siguiente apartado con las líneas futuras. Es posible que no se haya llegado al máximo de funcionalidades de la aplicación, pero sí que se han conseguido todos los objetivos que se habían planteado en la investigación del protocolo. Como objetivo añadido está que al realizar la implementación en un lenguaje multiplataforma como Golang [47] se puede ejcutar con unas pequeñas modificaciones en las rutas tanto en Linux [59] como en Windows [60]. La imagen de la ejecución en Windows se puede ver en A.3.17

Listado 3.9: Función del crawler de la red DHT para obtener hashes. Función principal parte del código del fichero **crawler.go**

```
//Crawler DHT
func Crawler(db *sql.DB, debug bool, verbose bool, threads int) {
  if debug {
    log.Println("Starting the application...")
  }

  max = threads * 1000
  data = make(chan *utils.Node, max)
  hashes = make(chan string, max)

  for len(data) == 0 {
    nodes, err := findNodes(debug, verbose)
    if err != nil {
      if debug {
        log.Println(err)
      }
    }
    if debug {
      log.Println("NODES")
      log.Println(len(nodes))
      log.Println(len(hashes))
    }
    for _, v := range nodes {
      data <- v
    }
    fmt.Println("Can't find any node")
  }

  if verbose {
    fmt.Println("Start scrapers ")
  }
  for i := 0; i < threads; i++ {
    waitGroup.Add(1)
    go workerCrawler(debug, verbose)
  }

  //start inserts
  if verbose {
    fmt.Println("Start inserts ")
  }
  for i := 0; i < threads/250; i++ { //pq: deadlock detected
    waitGroup.Add(1)
    go workerInsert(db, debug, verbose)
  }

  waitGroup.Wait()
  close(data)
  close(hashes)

}
```

Listado 3.10: Función para realizar la consulta sample_infohashes a un nodo de la red DHT. Parte del código del fichero **dht/sampleinfohashes.go**

```
//http://www.bittorrent.org/beps/bep_0051.html
func sampleInfohashes(debug bool, verbose bool, n *utils.Node) ([]*utils.Node, []string, error
    ) {
  //random id and target
  var hashList []string
  addr := n.Ip + ":" + strconv.Itoa(int(n.Port)) //juntamos ip:port
  if debug {
    log.Printf("sampleInfohashes %s", addr)
  }
  //id := nodeRQ{"q", "aa", "sample_infohashes", auxNRQ{id: "kjhgrtyhgfr45tyhgft6", target: "
      agr45yu763efrgthyji8"}}
  id := nodeRQ{"q", "aa", "sample_infohashes", auxNRQ{id: string(utils.RandomID()), target:
      string(utils.RandomID())}}

  var buf bytes.Buffer
  err := bencode.Marshal(&buf, id)
  if err != nil && debug {
    log.Printf("could not marshal: %v\n", err)
  }

  [...]

  conn, err := net.DialTimeout("udp", addr, timeoutDuration)

  [...]

  err = bencode.Unmarshal(r, &i)
  if err != nil && debug {
    log.Printf("Error Sample infohashes, could not unmarshal: %v\n", err)
  }

  encodedStr := hex.EncodeToString([]byte(i.R.Samples))

  aux := "" //use var buffer bytes.Buffer
  contador := 0
  for _, r := range encodedStr {
    c := string(r)
    aux += c
    contador++
    if contador == 40 {
      hashList = append(hashList, aux)
      aux = ""
      contador = 0
    }
  }

  nodes, err := utils.DecodeNodes(i.R.Nodes)

  [...]

  return nodes, hashList, nil
}
```

Listado 3.11: Función para la creación de las goroutines para generar todos los posibles hashes de un fichero. Parte del código del fichero **dht/createTorrent.go**

```
var hashesChan chan string
var inicio = 10

//WorkersTorrents create a hashlist for the file
func WorkersTorrents(cfgFile string, debug bool, verbose bool, file string, projectName string
    ) {
  fmt.Println("file", file)

  max := 1024
  hashesChan = make(chan string, max)

  db, err := utils.ConnectDb(cfgFile, debug, verbose)
  if err != nil {
    log.Fatal(err)
  }
  defer db.Close()
  utils.DeletePossibles(db, debug, verbose)
  for i := inicio; i < 30; i++ {
    waitGroup.Add(1)
    go worker(db, debug, verbose, i, file, file, false, projectName)
  }
  waitGroup.Wait()

}
```

Listado 3.12: Función para conectarse a la red DHT. Parte del código del fichero **dht/findTorrent.go**

```
//First function to search in DHT
func findNodes(debug bool, verbose bool) ([]*utils.Node, error) {
  nodesList := [7]string{"router.bittorrent.com:6881",
    "router.utorrent.com:6881",
    "dht.transmissionbt.com:6881",
    "dht.aelitis.com:6881",
    "router.silotis.us:6881",
    "dht.libtorrent.org:6881",
    "dht.libtorrent.org:25401"}

  var nodes []*utils.Node
  var err error
  for _, node := range nodesList {
    nodesAux, err := findNode(debug, verbose, node)
    if err != nil && debug {
      log.Printf("Error: %v\n", err)
    }
    nodes = append(nodes, nodesAux...)

  }

  return nodes, err
}
```

Listado 3.13: Función para buscar los peers de un fichero en la red DHT. Parte del código del fichero **dht/findTorrent.go**

```
func getPeers(db *sql.DB, debug bool, verbose bool, n *utils.Node, hash string) ([]*utils.Node
    , bool, error) {
  //get_peers Query = {"t":"aa", "y":"q", "q":"get_peers", "a": {"id":"abcdefghij0123456789",
      "info_hash":"mnopqrstuvwxyz123456"}}
  //bencoded = d1:ad2:id20:abcdefghij01234567899:info_hash20:mnopqrstuvwxyz123456e1:q9:
      get_peers1:t2:aa1:y1:qe

  //Response with peers = {"t":"aa", "y":"r", "r": {"id":"abcdefghij0123456789", "token":"
      aoeusnth", "values": ["axje.u", "idhtnm"]}}
  //bencoded = d1:rd2:id20:abcdefghij01234567895:token8:aoeusnth6:valuesl6:axje.u6:idhtnmee1:
      t2:aa1:y1:re

  //Response with closest nodes = {"t":"aa", "y":"r", "r": {"id":"abcdefghij0123456789", "
      token":"aoeusnth", "nodes": "def456..."}}
  //bencoded = d1:rd2:id20:abcdefghij01234567895:nodes9:def456...5:token8:aoeusnthe1:t2:aa1:y1
      :re

  addr := n.Ip + ":" + strconv.Itoa(int(n.Port)) //juntamos ip:port
  if debug {
    log.Printf("Find node %s", addr) //"\xe8B\x13\xa7\x94\xf3\xcc\xd8\x908*T\xa6L\xa6\x8b~\
        x92T3"
  }
  h, err := hex.DecodeString(hash)
  id := getPeerQ{"aa", "q", "get_peers", auxGetPeerQ{id: "abcdefghij0123456789", info_hash: h
      }}
  var buf bytes.Buffer
  if err := bencode.Marshal(&buf, id); err != nil {
    return nil, false, fmt.Errorf("could not marshal:", err)
  }

  conn, err := net.DialTimeout("udp", addr, time.Second*2)
  if err != nil {
    return nil, false, fmt.Errorf("could not dial %s: ", addr, err)
  }

  text := buf.String()
  fmt.Fprintf(conn, text)
  p := make([]byte, 2048)

  timeoutDuration := time.Second * 2
  conn.SetReadDeadline(time.Now().Add(timeoutDuration))

  _, err = bufio.NewReader(conn).Read(p)
  if err != nil {
    return nil, false, fmt.Errorf("could not read:", err)
  }
  defer conn.Close()

  var i getPeerR
  r := strings.NewReader(string(p))
  if err := bencode.Unmarshal(r, &i); err != nil {
    if debug {
      log.Printf("could not unmarshal:", err)
    }
  }

  nodes, err := utils.DecodeNodes(i.R.Nodes)
  if err != nil {
    return nil, false, fmt.Errorf("could not decode nodes:", err)
  }

...
```

Capítulo 4

Conclusiones y líneas futuras

En este apartado se resumen las conclusiones obtenidas y se proponen futuras líneas de investigación que se deriven del trabajo.

4.1 Conclusiones

Una vez hecho el estudio teórico completo y tras realizar el desarrollo de la aplicación se da por terminado el TFG. El software ha acabado con un gran número de funcionalidades que esperamos que sean útiles para investigadores y profesionales. A continuación, se enumeran todas las funcionales incluidas:

1. Añadir y eliminar hashes para monitorizar después.
2. Añadir y eliminar IPs para las alertas durante la monitorización.
3. Recorrer la red BitTorrent en busca de hashes para almacenar en la base de datos.
4. Crear un fichero .torrent dado un fichero o carpeta con diferentes parámetros para configurarlo.
5. Iniciar un proceso para monitorizar los hashes de la base de datos y alertas según las IPs almacenadas.
6. Descargar un fichero de la red BitTorrent.
7. Encontrar los hashes de la red BitTorrent dado un fichero.
8. Obtener el fichero .torrent desde un hash o un magnet.
9. Capacidad desde el programa de insertar hashes en la base de datos una a una o desde un fichero.
10. Mostrar desde la terminal los datos de la base de datos.

La orden “find” se considera la principal del programa junto a la monitorización, ya que permite dado un fichero sospechoso buscar en la red BitTorrent y monitorizar las IPs que lo están compartiendo, sin tener inicialmente el fichero .torrent. Tambien es importante la funcion crawl, ya que una base de datos suficientemente llena de hashes puede realizar el proceso de find mucho mas rápido.

4.2 Líneas futuras

A continuación, se enumera una lista de futuros avances a este TFG.

1. Crear una interfaz gráfica más amigable.
2. Añadir a un Docker [49] para facilitar su uso.
3. Gestionar por usuarios la monitorización y las alertar.
4. Añadir a la base de datos lo metadatos (principalmente nombre y/o .torrent) de cada hash almacenado. La base de datos está preparada para ello pero falta la implementación, tanto en el crawler como en las inserciones a mano.
5. Añadir la forma de obtener hashes pasiva siendo un nodo DHT y esperar consultas de hashes.
6. Modificar el crawler con memoria para recorrer todos los nodos en el menor tiempo posible (mucha RAM).
7. Descargar múltiples ficheros simultáneos.
8. Realizar esta investigación para el protocolo IPFS [61].
9. Optimizar de la aplicación

Bibliografía

[1] "Ieee xplore digital library," https://ieeexplore.ieee.org [Último acceso 18/Septiembre/2019].

[2] P. Evangelista, M. Amaral, C. Miers, W. Goya, M. Simplicio, T. Carvalho, and V. Souza, "Ebitsim: An enhanced bittorrent simulation using omnet++ 4 - bittorrent architecture," 07 2011, pp. 437–440.

[3] D. Harrison, "The possible paths of the status of beps," Enero 2008, https://www.bittorrent.org/beps/bep_0001_1.png.[imagen] [Último acceso 18/Septiembre/2019].

[4] "Página oficial de toad data modeler," http://www.toadworld.com/products/toad-data-modeler [Último acceso 18/Septiembre/2019].

[5] "Cómo instalar y usar docker en ubuntu 18.04," https://www.digitalocean.com/community/tutorials/como-instalar-y-usar-docker-en-ubuntu-18-04-1-es [Último acceso 18/Septiembre/2019].

[6] "How to install go on ubuntu 18.04," https://www.digitalocean.com/community/tutorials/how-to-install-go-on-ubuntu-18-04 [Último acceso 18/Septiembre/2019].

[7] "Página oficial netflix," https://www.netflix.com/browse [Último acceso 18/Septiembre/2019].

[8] Sandvine, "Sandvine 2018 internet phenomena report," Octubre 2018, https://www.sandvine.com/hubfs/downloads/phenomena/2018-phenomena-report.pdf [Último acceso 1/septiembre/2019].

[9] BitTorrent, "It's official: Bittorrent is now part of tron!" Julio 2018, https://www.depeca.uah.es/images/plantillas/book-latex.zip [Último acceso 21/Julio/2019].

[10] smcdonald, "μtorrent torrent web passes 1 million daily active users," Octubre 2018, http://blog.utorrent.com/2018/10/04/%C2%B5torrent-web-passes-1-million-daily-active-users/ [Último acceso 18/Septiembre/2019].

[11] "utorrent now includes tokenized bittorrent speed," Julio 2019, https://torrentfreak.com/utorrent-now-includes-tokenized-bittorrent-speed-190709/ [Último acceso 18/Septiembre/2019].

[12] D. de Electrónica UAH, "Plantilla latex tfc, tfg, tesis," https://www.depeca.uah.es/images/plantillas/book-latex.zip [Último acceso 18/Septiembre/2019].

[13] D. Azhigulov, S. Kairatova, I. A. Ukaegbu, and M. Myrzalieva, "Analyzing the bittorrent ecosystem of central asia," in *2018 International Conference on Computing and Network Communications (CoCoNet)*, Aug 2018, pp. 159–163.

[14] C. G. López, "Measuring bittorrent ecosystems," in *2012 International Conference on High Performance Computing Simulation (HPCS)*, July 2012, pp. 307–307.

[15] W. Mazurczyk and P. Kopiczko, "Understanding bittorrent through real measurements," *China Communications*, vol. 10, no. 11, pp. 107–118, Nov 2013.

[16] J. Kurose and K. Ross, "Redes de computadoras 5 ed." Pearson Educación, 2010, p. 139.

[17] "Bittorrent inc. changed its name to rainberry," Mayo 2018, https://torrentfreak.com/bittorrent-inc-changed-its-name-to-rainberry-180512/ [Último acceso 18/Septiembre/2019].

[18] "Página oficial bittorrent," https://www.bittorrent.com [Último acceso 18/Septiembre/2019].

[19] "Página oficial μtorrent," https://www.utorrent.com [Último acceso 18/Septiembre/2019].

[20] B. Cohen, "I'm no longer in any way affiliated with bittorrent and have never been affiliated with tron or justin sun," Enero 2019, https://twitter.com/bramcohen/status/1087577311715577856?s=20 [Último acceso 18/Septiembre/2019].

[21] "Bittorrent inventor bram cohen leaves bittorrent behind," Agosto 2018, https://torrentfreak.com/bittorrent-inventor-bram-cohen-leaves-bittorrent-behind-180819/ [Último acceso 18/Septiembre/2019].

[22] B. Cohen, "Bittorrent - a new p2p app," Julio 2001, https://web.archive.org/web/20080129085545/http://finance.groups.yahoo.com/group/decentralization/message/3160 [Último acceso 18/Septiembre/2019].

[23] "Página oficial vuze," https://www.vuze.com [Último acceso 18/Septiembre/2019].

[24] "Página oficial transmission," https://transmissionbt.com [Último acceso 18/Septiembre/2019].

[25] B. Cohen, "The bittorrent protocol specification," Enero 2008, https://www.bittorrent.org/beps/bep_0003.html [Último acceso 18/Septiembre/2019].

[26] ——, "Incentives build robustness in bittorrent," Mayo 2003, http://bittorrent.org/bittorrentecon.pdf [Último acceso 1/septiembre/2019].

[27] "Github oficial bittorrent," https://github.com/bittorrent/bittorrent.org [Último acceso 18/Septiembre/2019].

[28] B. Cohen, "The bittorrent protocol specification v2," Enero 2008, https://www.bittorrent.org/beps/bep_0052.html [Último acceso 18/Septiembre/2019].

[29] D. Harrison, "Sample restructured text bep template," Febrero 2008, https://www.bittorrent.org/beps/bep_0002.html [Último acceso 18/Septiembre/2019].

[30] J. Hoffman, "Multitracker metadata extension," Febrero 2008, https://www.bittorrent.org/beps/bep_0012.html [Último acceso 18/Septiembre/2019].

[31] O. van der Spek, "Udp tracker protocol for bittorrent," Febrero 2008, https://www.bittorrent.org/beps/bep_0015.html [Último acceso 18/Septiembre/2019].

[32] J. Zelinskie, "Tracker protocol extension: Scrape," Septiembre 2016, https://www.bittorrent.org/beps/bep_0048.html [Último acceso 18/Septiembre/2019].

[33] D. Harrison, "Tracker returns compact peer lists," Mayo 2008, https://www.bittorrent.org/beps/bep_0023.html [Último acceso 18/Septiembre/2019].

[34] A. Norberg, "utorrent transport protocol," Junio 2009, https://www.bittorrent.org/beps/bep_0029.html [Último acceso 18/Septiembre/2019].

[35] D. Harrison, "Peer id conventions," Febrero 2008, https://www.bittorrent.org/beps/bep_0020.html [Último acceso 18/Septiembre/2019].

[36] "Bittorrentspecification: peer_id," https://wiki.theory.org/index.php/BitTorrentSpecification#peer_id [Último acceso 18/Septiembre/2019].

[37] J. Fonseca, B. Reza, and L. Fjeldsted, "Bittorrent protocol – btp/1.0," April 2005.

[38] A. Norberg and A. Loewenstern, "Dht protocol," Enero 2008, http://www.bittorrent.org/beps/bep_0005.html [Último acceso 18/Septiembre/2019].

[39] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the xor metric," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, ser. IPTPS '01. London, UK, UK: Springer-Verlag, 2002, pp. 53–65. [Online]. Available: http://dl.acm.org/citation.cfm?id=646334.687801

[40] T. 8472, "Dht infohash indexing," Diciembre 2016, https://www.bittorrent.org/beps/bep_0051.html [Último acceso 18/Septiembre/2019].

[41] A. Norberg, "Dht security extension," Enero 2014, https://www.bittorrent.org/beps/bep_0042.html [Último acceso 18/Septiembre/2019].

[42] D. G. Celemendi, "Desarrollo de una aplicación bittorrent(en pharo-smalltalk)," Ph.D. dissertation, Universitat Politècnica de Catalunya, 2015.

[43] "Página oficial libtorrent," https://www.libtorrent.org [Último acceso 18/Septiembre/2019].

[44] "Página oficial ttorrent," https://mpetazzoni.github.io/ttorrent/ [Último acceso 18/Septiembre/2019].

[45] "Github oficial torrent (anacrolix)," https://github.com/anacrolix/torrent [Último acceso 18/Septiembre/2019].

[46] "Github oficial monotorrent," https://github.com/mono/monotorrent [Último acceso 18/Septiembre/2019].

[47] "Página oficial golang," https://golang.org/ [Último acceso 18/Septiembre/2019].

[48] "Página oficial postgresql," https://www.postgresql.org/ [Último acceso 18/Septiembre/2019].

[49] "Página oficial docker," https://www.docker.com/ [Último acceso 18/Septiembre/2019].

[50] "Github oficial cobra," https://github.com/spf13/cobra [Último acceso 18/Septiembre/2019].

[51] "Github oficial torrent (anacrolix)," https://github.com/anacrolix/torrent/ [Último acceso 18/Septiembre/2019].

[52] "Github oficial dht (anacrolix)," https://github.com/anacrolix/dht [Último acceso 18/Septiembre/2019].

[53] "Github oficial tagflag (anacrolix)," https://github.com/anacrolix/tagflag [Último acceso 18/Septiembre/2019].

[54] "Github oficial bencode-go," https://github.com/jackpal/bencode-go [Último acceso 18/Septiembre/2019].

[55] "Github oficial pq," https://github.com/lib/pq [Último acceso 18/Septiembre/2019].

[56] "Github oficial toml," https://github.com/BurntSushi/toml [Último acceso 18/Septiembre/2019].

[57] "Github oficial goscrape," https://github.com/etix/goscrape [Último acceso 18/Septiembre/2019].

[58] "Ejemplo de descarga de un torrent en el github oficial torrent (anacrolix)," https://github.com/anacrolix/torrent/blob/master/cmd/torrent/main.go [Último acceso 18/Septiembre/2019].

[59] "Información sobre gnu/linux en wikipedia," http://es.wikipedia.org/wiki/GNU/Linux [Último acceso 1/noviembre/2013].

[60] "Información sobre windows," https://www.microsoft.com/es-es/windows [Último acceso 1/noviembre/2013].

[61] "Página oficial de ipfs," https://ipfs.io [Último acceso 18/Septiembre/2019].

[62] "Página de la aplicación gcc," http://savannah.gnu.org/projects/gcc/ [Último acceso 1/noviembre/2013].

[63] "Página oficial de visual studio code," https://code.visualstudio.com/ [Último acceso 18/Septiembre/2019].

[64] "Página oficial de latex," https://www.latex-project.org/ [Último acceso 18/Septiembre/2019].

Apéndice A

Manual de usuario

A.1 Introducción

A continuación, se explicará la instalación de la aplicación y su uso. En la instalación se explicará únicamente la instalación de la base de datos utilizando Docker y la aplicación con los ficheros adjuntos (carpeta instalación). La aplicación utiliza CLI, por lo que se explicarán tanto sus funciones como sus "FLAGs". En las órdenes de este manual se utilizarán las palabras entre llaves " " para indicar que es un campo obligatorio y entre corchetes "[ ]" para indicar que es opcional.

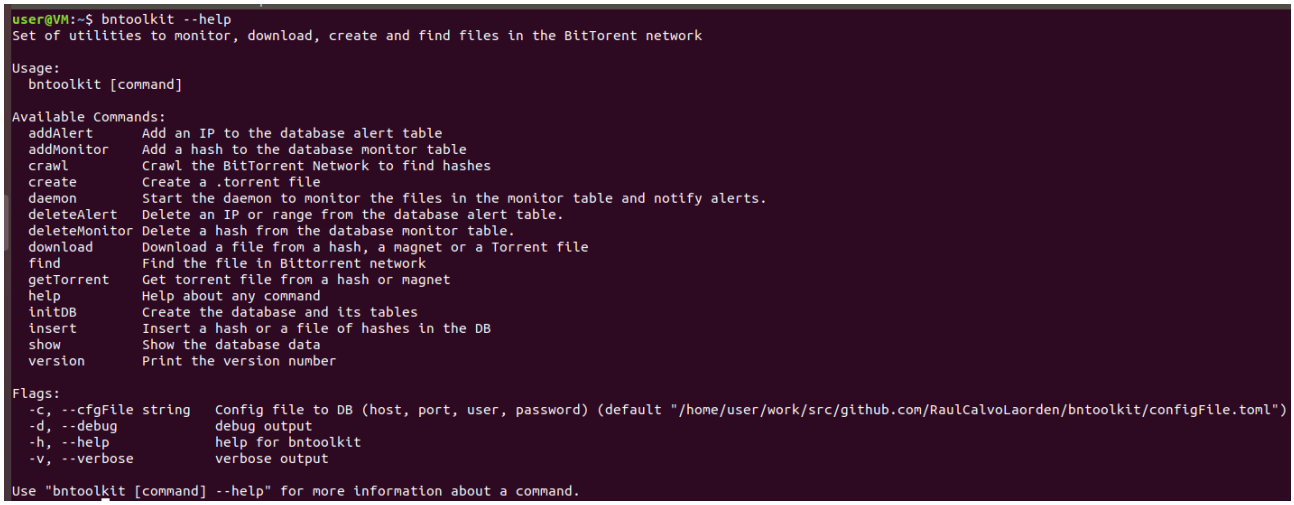

Figura A.1: BNTtoolkit –help

A.2 Instalación

Lo primero que hay que hacer es tener instalados tanto Docker como Golang. a continuación, se instalará BNToolkit y se inicializará la base de datos. Una vez hecho esto la aplicación estará lista para su uso.

A.2.1 Docker

Para instalar de Docker en Ubuntu 18 se pueden ejecutar las órdenes del código A.1.

Listado A.1: Ordenes para la instalación de Docker en Ubuntu 18 [5]

```
sudo apt install apt-transport-https ca-certificates curl software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic
    stable"
sudo apt update
sudo apt install docker-ce
```

A.2.2 Golang

Para instalar de Golang en Ubuntu 18 se pueden ejecutar las órdenes del código A.1.

Listado A.2: Ordenes para la instalación de Golang en Ubuntu 18 [6]

```
sudo apt-get update
sudo apt-get -y upgrade
wget https://dl.google.com/go/go1.12.9.linux-amd64.tar.gz #Check latest in https://golang.org/
    dl/
sudo tar -xvf go1.12.9.linux-amd64.tar.gz
sudo mv go /usr/local
mkdir ~/work
echo 'export GOROOT=/usr/local/go
export GOPATH=$HOME/work
export PATH=$GOPATH/bin:$GOROOT/bin:$PATH' >> ~/.profile
source ~/.profile
```

A.2.3 BNToolkit

Si fuese público el repositorio de GitHub únicamente sería necesario ejecutar **go install github.com/RaulCalvoLaorden/bntoolkit**. Pero para la instalación de la aplicación según viene con el TFG es recomendable utilizar los ficheros en la carpeta instalación. Una vez está esa carpeta en el sistema donde se quiere instalar solo hay que ejecutar **bash installLocal.sh**. El código de "installLocal.sh" se puede ver en el código A.3. Si fuese público el repositorio de GitHub únicamente sería necesario ejecutar **go install github.com/RaulCalvoLaorden/bntoolkit**.

Listado A.3: Código fuente de **installLocal.sh**

```
#!/bin/bash
mkdir -p $GOPATH/src/github.com
cp -r * $GOPATH/src/github.com/
cd $GOPATH/src/github.com/RaulCalvoLaorden/bntoolkit/
ls
go get .
go install
bntoolkit
```

A.3 Uso

Una vez instalados todos los prerrequisitos podemos iniciar la base de datos, y empezar a usar la aplicación.

A.3.1 Inicio PostgreSQL

Para levantar la base de datos es necesario ejecutar el Docker de PostgreSQL. Para ello se puede utilizar las órdenes del código A.4. Este código se muestra también en la ayuda de la orden "help". Como se puede ver en la imagen A.4.

Listado A.4: Código para levantar un Docker con PostgreSQL

```
mkdir ~/postgres #or any folder to store data
sudo docker stop hashpostgres ; sudo docker rm hashpostgres #delete if it exists
sudo docker run -d -p 5432:5432 --mount type=bind,source=$HOME/postgres/,target=/var/lib/
    postgresql/data --name hashpostgres -e POSTGRES_PASSWORD=postgres99 postgres
```

A.3.2 Fichero de configuración configFile.toml

El fichero de configuración **configFile.toml** contiene la información del host de la base de datos, su puerto, el usuario, la contraseña y el nombre de la base de datos donde se almacenan los datos. Se recomienda no cambiar el nombre de la base de datos para funciones como initDB. Este fichero se puede modificar directamente o hacer una copia y especificarlo a la hora de ejecutar la aplicación con la "FLAG" –**cfgFile** y la ruta al fichero. El fichero por defecto se puede ver en el código A.5

Listado A.5: Código del fichero de configuración **configFile.toml**

```
host="localhost"
port=5432
user="postgres"
password="postgres99"
dbname="hash"
```

Junto a –**cfgFile** hay dos "FLAGs" globales más. –**verbose** para mostrar más mensajes por pantalla y –**debug** para aumentar aún más el número de mensajes por pantalla.

A.3.3 help

La orden help muestra ayuda de cualquier orden de la aplicación. Para ejecutarla solo hay que poner **help** [**orden**].

Esta orden tiene la siguiente "FLAGs":

- –**help**. Esta "FLAG" muestra ayuda para la orden.

```
user@VM:~$ bntoolkit help --help
Help provides help for any command in the application.
Simply type bntoolkit help [path to command] for full details.

Usage:
  bntoolkit help [command] [flags]

Flags:
  -h, --help   help for help

Global Flags:
  -c, --cfgFile string   Config file to DB (host, port, user, password) (default "/home/user/work/src/github.com/RaulCalvoLaorden/bntoolkit/configFile.toml")
  -d, --debug            debug output
  -v, --verbose          verbose output
user@VM:~$
```

Figura A.2: BNTtoolkit help –help

A.3.4 version

La orden version muestra la versión la aplicación. Para ejecutarla solo hay que poner **version**.

Esta orden tiene la siguiente "FLAGs":

- –**help**. Esta "FLAG" muestra ayuda para la orden.

```
user@VM:~$ bntoolkit version --help
Print the version number.
For example:
        bntoolkit version

Usage:
  bntoolkit version [flags]

Flags:
  -h, --help   help for version

Global Flags:
  -c, --cfgFile string   Config file to DB (host, port, user, password) (default "/home/user/work/src/github.com/RaulCalvoLaorden/bntoolkit/configFile.toml")
  -d, --debug            debug output
  -v, --verbose          verbose output
user@VM:~$ bntoolkit version
bntoolkit (BitTorrent Network toolkit) v0.9
user@VM:~$
```

Figura A.3: BNTtoolkit version –help y ejecución

A.3.5 initDB

La orden initDB inicializa la base de datos. Se debe ejecutar antes de cualquier otra orden si la base de datos está vacía. Para ejecutarla solo hay que poner **initDB**.

Esta orden tiene la siguiente "FLAG":

- –**help**. Esta "FLAG" muestra ayuda para la orden.

```
user@VM:~$ bntoolkit initDB --help
Create the database and its tables. This command is required the first time the database is connected.
Recommended DB:
        mkdir ~/postgres #or any folder to store data
        sudo docker stop hashpostgres ; sudo docker rm hashpostgres #delete if it exists
        sudo docker run -d -p 5432:5432 --mount type=bind,source=$HOME/postgres/,target=/var/lib/postgresql/data --name hashpostgres -e POSTGRES_PASSWORD=postgres99 postgres
For example:
        bntoolkit initDB

Usage:
  bntoolkit initDB [flags]

Flags:
  -h, --help   help for initDB

Global Flags:
  -c, --cfgFile string   Config file to DB (host, port, user, password) (default "/home/user/work/src/github.com/RaulCalvoLaorden/bntoolkit/configFile.toml")
  -d, --debug            debug output
  -v, --verbose          verbose output
```

Figura A.4: BNTtoolkit initDB –help

A.3.6 create

La orden create sirve para crear un fichero .torrent. A este fichero se le puede especificar el nombre de salida, el tamaño de las piezas, el tracker principal y un comentario. Para ejecutarla solo hay que poner **create <fichero/carpeta>[flags]**, como se puede ver en la imagen A.5.

Esta orden tiene las siguientes "FLAGs":

- **–help**. Esta "FLAG" muestra ayuda para la orden.
- **–comment**. Para especificar el comentario a añadir al .torrent.
- **–outfile**. Guarda el .torrent con ese nombre.
- **–piecesize**. Tamaño de las piezas.
- **–tracker**. especifica un tracker para el fichero.

```
user@VM:~$ bntoolkit create -h
Create a .torrent file. You can specify the output file, the pieze size, the tracker and a comment
For example:
        bntoolkit create ubuntu.iso -o ubuntuTorrent -m test

Usage:
  bntoolkit create <file/folder> [flags]

Flags:
  -m, --comment string   Comment for the .torrent
  -h, --help             help for create
  -o, --outfile string   Save the generated .torrent to this filename (default "output")
  -p, --piecesize int    Pieze size to the .torrent (bytes) (default 2048)
  -t, --tracker string   Default tracker to the .torrent

Global Flags:
  -c, --cfgFile string   Config file to DB (host, port, user, password) (default "/home/user/work/src/github.com/RaulCalvoLaorden/bntoolkit/configFile.toml")
  -d, --debug            debug output
  -v, --verbose          verbose output
user@VM:~$ bntoolkit create go1.12.9.linux-amd64.tar.gz -o filegolang
create called
go1.12.9.linux-amd64.tar.gz
filegolang.torrent
```

Figura A.5: BNTtoolkit create –help y ejecución

A.3.7 download

La orden download descarga el .torrent, hash o magnet que se le indique. Para ejecutarla solo hay que poner **download <fichero/hash/magnet>[flags]**, como se puede ver en la imagen A.6.

Esta orden tiene las siguientes "FLAGs":

- –**help**. Esta "FLAG" muestra ayuda para la orden.
- –**path**. Ruta donde se guardará los ficheros del torrent (por defecto la ruta actual).

```
user@VM:~$ bntoolkit download --help
Download a file from a hash, a magnet or a Torrent file.
For example:
        bntoolkit download e84213a794f3ccd890382a54a64ca68b7e925433
        bntoolkit download ubuntu.torrent

Usage:
  bntoolkit download <file/hash/magnet> [flags]

Flags:
  -h, --help          help for download
  -p, --path string   path to download

Global Flags:
  -c, --cfgFile string   Config file to DB (host, port, user, password) (default "/home/user/work/src/github.com/RaulCalvoLaorden/bntoolkit/configFile.toml")
  -d, --debug            debug output
  -v, --verbose          verbose output
user@VM:~$ bntoolkit download e84213a794f3ccd890382a54a64ca68b7e925433
download called 1
e84213a794f3ccd890382a54a64ca68b7e925433
2019-09-11 17:45:12 client.go:324: dht server on [::]:44765: dht bootstrap: dht.TraversalStats{NumAddrsTried:9, NumResponses:0}
2019-09-11 17:45:14 portfwd.go:31: discovered 0 upnp devices
Downloading ubuntu-18.04.1-desktop-amd64.iso
go-libutp: 2019/09/11 17:52:27 callbacks.go:67: error sending packet: write udp6 [::]:44765->[2a02:8108:8240:31ce:211:32ff:fe0f:efa6]:16881: sendto: network is unreachable
go-libutp: 2019/09/11 17:52:30 callbacks.go:67: error sending packet: write udp6 [::]:44765->[2a02:8108:8240:31ce:211:32ff:fe0f:efa6]:16881: sendto: network is unreachable
go-libutp: 2019/09/11 17:52:30 callbacks.go:67: error sending packet: write udp6 [::]:44765->[2a02:8108:8240:31ce:211:32ff:fe0f:efa6]:16881: sendto: network is unreachable
go-libutp: 2019/09/11 17:52:40 callbacks.go:67: error sending packet: write udp6 [::]:44765->[2a02:8108:8240:31ce:211:32ff:fe0f:efa6]:16881: sendto: network is unreachable
go-libutp: 2019/09/11 17:52:44 callbacks.go:67: error sending packet: write udp6 [::]:44765->[2a02:8108:8240:31ce:211:32ff:fe0f:efa6]:16881: sendto: network is unreachable
go-libutp: 2019/09/11 17:52:44 callbacks.go:67: error sending packet: write udp6 [::]:44765->[2a02:8108:8240:31ce:211:32ff:fe0f:efa6]:16881: sendto: network is unreachable
go-libutp: 2019/09/11 17:53:52 callbacks.go:67: error sending packet: write udp6 [::]:44765->[2a02:20a8:3::a:a000]:51413: sendto: network is unreachable
2019/09/11 17:53:53 Torrent downloaded
user@VM:~$
```

Figura A.6: BNTtoolkit download –help y ejecución

A.3.8 addAlert

La orden addAlert sirve para añadir una IP o un rango a la tabla de alertas para cuando se comience a monitorizar los hashes de la base de datos aparezca por pantalla las IPs añadidas o las IPs en el rango que estén compartiendo ese fichero. Para ejecutarla solo hay que poner **addAlert <IP/rango>[flags]**, como se puede ver en la imagen A.7.

Esta orden tiene las siguientes "FLAGs":

- –**help**. Esta "FLAG" muestra ayuda para la orden.
- –**projectName**. Nombre del proyecto asociado a esa Alerta.

A.3.9 deleteAlert

La orden deleteAlert sirve para eliminar una IP o un rango de la tabla de alertas. Para ejecutarla solo hay que poner **deleteAlert <IP/rango>[flags]**, como se puede ver en la imagen A.7.

Esta orden tiene la siguiente "FLAG":

- –**help**. Esta "FLAG" muestra ayuda para la orden.

```
user@VM:~$ bntoolkit addAlert -h
Add an IP or range to the database alert table. When the daemon is executed an alert appears if that IP appears in the alerts table (IP or range).
For example:
               bntoolkit addAlert 1.1.0.0/16

Usage:
  bntoolkit addAlert <IP/Range> [flags]

Flags:
  -h, --help                help for addAlert
  -p, --projectName string   Monitoring projectName (default "default")

Global Flags:
  -c, --cfgFile string   Config file to DB (host, port, user, password) (default "/home/user/work/src/github.com/RaulCalvoLaorden/bntoolkit/configF
  -d, --debug            debug output
  -v, --verbose          verbose output
user@VM:~$ bntoolkit addAlert 1.1.0.0/16
addAlert called
user@VM:~$ bntoolkit show alert
results called
ip              |        list   |        user
1.1.0.0/16      |        test   |        username

user@VM:~$ bntoolkit deleteAlert -h
Delete an IP or range from the database alert table.
For example:
        bntoolkit deleteAlert 1.1.0.0/16

Usage:
  bntoolkit deleteAlert <IP/Range> [flags]

Flags:
  -h, --help   help for deleteAlert

Global Flags:
  -c, --cfgFile string   Config file to DB (host, port, user, password) (default "/home/user/work/src/github.com/RaulCalvoLaorden/bntoolkit/configF
  -d, --debug            debug output
  -v, --verbose          verbose output
user@VM:~$ bntoolkit deleteAlert 1.1.0.0/16
deleteAlert called
user@VM:~$ bntoolkit show alert
results called
ip              |        list   |        user

user@VM:~$
```

Figura A.7: addAlert y deleteAlert

A.3.10 addmonitor

La orden addmonitor sirve para añadir un hash a la tabla de hashes para monitorizar. Para ejecutarla solo hay que poner **addmonitor <hash>[flags]**, como se puede ver en la imagen A.8.

Esta orden tiene las siguientes "FLAGs":

- –**help**. Esta "FLAG" muestra ayuda para la orden.
- –**projectName**. Nombre del proyecto asociado a esa monitorización.
- –**userName**. Nombre del usuario que crea la monitorización.

A.3.11 deleteMonitor

La orden deleteMonitor sirve para eliminar un hash de la tabla de hashes a monitorizar. Para ejecutarla solo hay que poner **deleteMonitor <hash>[flags]**, como se puede ver en la imagen A.8.

Esta orden tiene la siguiente "FLAG":

- –**help**. Esta "FLAG" muestra ayuda para la orden.

```
user@VM:~$ bntoolkit addMonitor -h
Add a hash to the database monitor table.
For example:
        bntookit addMonitor e84213a794f3ccd890382a54a64ca68b7e925433

Usage:
  bntoolkit addMonitor <HASH> [flags]

Flags:
  -h, --help                help for addMonitor
  -p, --projectName string  Monitoring projectName (default "default")
  -u, --userName string     Monitoring username (default "default")

Global Flags:
  -c, --cfgFile string   Config file to DB (host, port, user, password) (default "/home/user/work
  -d, --debug            debug output
  -v, --verbose          verbose output
user@VM:~$ bntoolkit addMonitor e84213a794f3ccd890382a54a64ca68b7e925433
addMonitor called
user@VM:~$ bntoolkit show monitor
results called
hash                                        |       username       |       projectName
e84213a794f3ccd890382a54a64ca68b7e925433    |       default        |       default

user@VM:~$ bntoolkit deleteMonitor -h
Delete a hash from the database monitor table.
For example:
        bntoolkit deleteMonitor e84213a794f3ccd890382a54a64ca68b7e925433

Usage:
  bntoolkit deleteMonitor <HASH> [flags]

Flags:
  -h, --help   help for deleteMonitor

Global Flags:
  -c, --cfgFile string   Config file to DB (host, port, user, password) (default "/home/user/work
  -d, --debug            debug output
  -v, --verbose          verbose output
user@VM:~$ bntoolkit deleteMonitor e84213a794f3ccd890382a54a64ca68b7e925433
deleteMonitor called
user@VM:~$ bntoolkit show monitor
results called
hash                                        |       username       |       projectName
```

Figura A.8: addMonitor y deleteMonitor

A.3.12 crawl

Esta orden recorre la DHT para encontrar hashes que estén activos y los almacena en la base de datos. Para ejecutarla solo hay que poner **crawl** [**flags**], como se puede ver en la imagen A.9.

Esta orden tiene las siguientes “FLAGs”:

- –**help**. Esta “FLAG” muestra ayuda para la orden.
- –**threads**. Entero que indica el número de hilos para el Crawl, por defecto 500.

```
user@VM:~$ bntoolkit show monitor
results called
hash                                        |        username       |        projectName

user@VM:~$ bntoolkit crawl -h
Crawl the BitTorrent Network to find hashes and storage it in the DB.
For example:
        bntoolkit crawl -t 500

Usage:
  bntoolkit crawl [flags]

Flags:
  -h, --help          help for crawl
  -t, --threads int   threads, over 500 you need to change the max files to connections (ulimit -n XXXX) (default 500)

Global Flags:
  -c, --cfgFile string   Config file to DB (host, port, user, password) (default "/home/user/work/src/github.com/RaulCalv
  -d, --debug            debug output
  -v, --verbose          verbose output
user@VM:~$ bntoolkit crawl
crawl called
Can't find any node
^C
user@VM:~$
```

Figura A.9: bntoolkit crawl –help y ejecución

A.3.13 daemon

Esta orden comienza la monitorización de todos los hashes que haya en la base de datos (monitor) o de un determinado proyecto y muestra por pantalla las alertas de IPs o rangos añadidos también a la base de datos. Para ejecutarla solo hay que poner **daemon** [**flags**], como se puede ver en la imagen A.10.

Esta orden tiene las siguientes “FLAGs”:

- –**help**. Esta “FLAG” muestra ayuda para la orden.
- –**projectName**. Nombre del proyecto asociado a esa monitorización.
- –**crawl**. Ejecuta también la orden crawl.

```
user@VM:~/full$ bntoolkit  show alert
results called
ip              |        list   |        user
0.0.0.0/0       |        test   |        username

user@VM:~/full$ bntoolkit  show monitor
results called
hash                                            |        username       |        projectName
e84213a794f3ccd890382a54a64ca68b7e925433        |        default        |        default

user@VM:~/full$ bntoolkit daemon --help
Start the daemon to monitor the files in the monitor table, notify alerts and optionally crape DHT
For example:
        bntoolkit daemon -s

Usage:
  bntoolkit daemon [flags]

Flags:
  -s, --crawl               Crawl DHT
  -h, --help                help for daemon
  -p, --projectName string  Monitoring project (default "default")

Global Flags:
  -c, --cfgFile string   Config file to DB (host, port, user, password) (default "/home/user/work/src/github.com/RaulC
  -d, --debug            debug output
  -v, --verbose          verbose output
user@VM:~/full$ bntoolkit daemon
daemon called
Start monitoring PostgreSQL...
e84213a794f3ccd890382a54a64ca68b7e925433
starting get peers lib
[[232 66 19 167 148 243 204 216 144 56 42 84 166 76 166 139 126 146 84 51]]
DATE: 2019-09-11 17:43:21.415355, IP: 211.72.224.176/32, HASH: e84213a794f3ccd890382a54a64ca68b7e925433
DATE: 2019-09-11 17:43:21.813677, IP: 217.76.31.6/32, HASH: e84213a794f3ccd890382a54a64ca68b7e925433
DATE: 2019-09-11 17:43:21.987883, IP: 84.53.198.229/32, HASH: e84213a794f3ccd890382a54a64ca68b7e925433
DATE: 2019-09-11 17:43:22.051358, IP: 114.73.37.208/32, HASH: e84213a794f3ccd890382a54a64ca68b7e925433
DATE: 2019-09-11 17:43:22.169809, IP: 90.143.162.146/32, HASH: e84213a794f3ccd890382a54a64ca68b7e925433
DATE: 2019-09-11 17:43:22.290878, IP: 178.206.89.117/32, HASH: e84213a794f3ccd890382a54a64ca68b7e925433
DATE: 2019-09-11 17:43:22.410351, IP: 86.106.32.9/32, HASH: e84213a794f3ccd890382a54a64ca68b7e925433
DATE: 2019-09-11 17:43:22.529888, IP: 220.137.29.72/32, HASH: e84213a794f3ccd890382a54a64ca68b7e925433
DATE: 2019-09-11 17:43:22.649184, IP: 88.180.140.51/32, HASH: e84213a794f3ccd890382a54a64ca68b7e925433
DATE: 2019-09-11 17:43:22.711319, IP: 188.242.232.74/32, HASH: e84213a794f3ccd890382a54a64ca68b7e925433
DATE: 2019-09-11 17:43:22.773473, IP: 185.235.115.167/32, HASH: e84213a794f3ccd890382a54a64ca68b7e925433
^C
user@VM:~/full$
```

Figura A.10: bntoolkit daemon –help y ejecución

A.3.14 find

Esta orden es la más importante de todo el programa. Sirve para buscar en la red BitTorrent (tanto DHT como tracker) un fichero o carpeta que esté en local. De este modo podemos encontrar todos los hashes de ficheros torrent que se están compartiendo. Para ejcutar esta orden es necesario poner **find <fichero/carpeta>[flags]**

Esta orden tiene las siguientes “FLAGs”:

- –**help**. Esta “FLAG” muestra ayuda para la orden.
- –**timeout**. Tiempo de espera antes de dejar de buscar en la DHT y tracker y tiempo antes de cancelar los intentos de descargas.
- –**mode**. Find tiene cuatro modos diferentes. El modo 0 busca solo en los trackers más conocidos y acaba. El modo 1 busca en los trackers y en la DHT. El modo 3 añade tambien la descarga de los hashes que ha encontrado en los Trackers y en la DHT. Y, por último, el modo 4 intenta descargar todos los posibles hashes generados de el fichero o carpeta. El tiempo de ejecución es cada vez mayor dependiendo del modo elegido.
- –**no-add**. “Flag” para evitar almacenar los hashes validos en la base de datos (no implementado aún).

- –**projectName**. Nombre del proyecto asociado a esa búsqueda.
- –**tracker**. Tracker específico para buscar el fichero (no implementado aún).

Lo más recomendable es utilizar el modo por defecto, ya que la descarga permite eliminar falsos positivos, y modificar los valores de la "FLAG" –**timeout** según el tiempo del que dispongamos, ya que mayor tiempo para buscar permite minimizar los falsos negativos.

Figura A.11: bntoolkit find –help

A.3.15 insert

Esta orden inserta uno o más hashes en la base de datos, admite un hash directamente o un fichero con un hash por línea. Para ejecutarla solo hay que poner **insert <hash/fichero>[flags]**, como se puede ver en la imagen A.12.

Esta orden tiene la siguiente "FLAG":

- –**help**. Esta "FLAG" muestra ayuda para la orden.

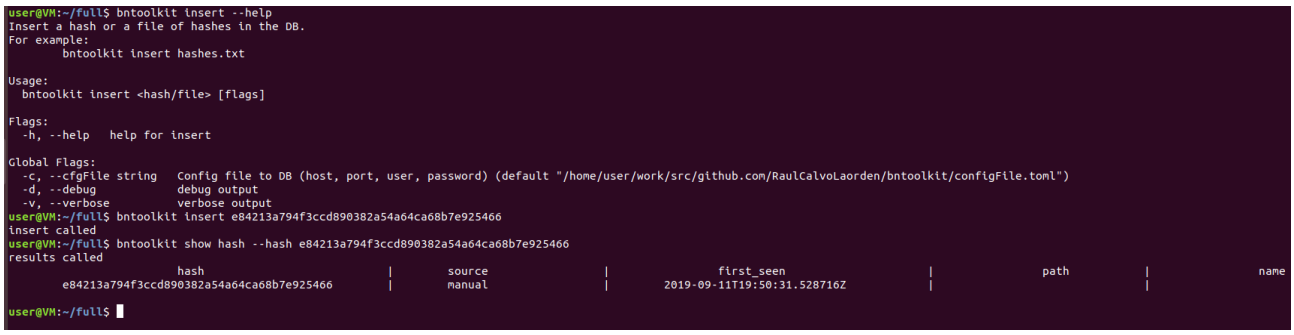

Figura A.12: bntoolkit insert –help y ejecución

A.3.16 show

Esta orden muestra los datos de la base de datos. Esta orden necesita otra después que especifique que dato quiere mostrar, en la imagen A.13 se puede ver el listado completo.

```
user@VM:~/full$ bntoolkit show --help
Show the database data

Usage:
  bntoolkit show [flags]
  bntoolkit show [command]

Available Commands:
  alert       Show the database data of the table alert
  count       Show the database count of hash
  hash        Show the database data of the table hash
  ip          Show the database data of the table ip
  monitor     Show the database data of the table monitor
  possibles   Show the database data of the table possibles
  project     Show the database data of the table project

Flags:
  -h, --help           help for show
  -w, --where string   Where in SQL lenguaje

Global Flags:
  -c, --cfgFile string   Config file to DB (host, port, user, password) (default "/home
  -d, --debug            debug output
  -v, --verbose          verbose output

Use "bntoolkit show [command] --help" for more information about a command.
user@VM:~/full$
```

Figura A.13: bntoolkit show –help

A.3.16.1 hash

Muestra la tabla de hashes.

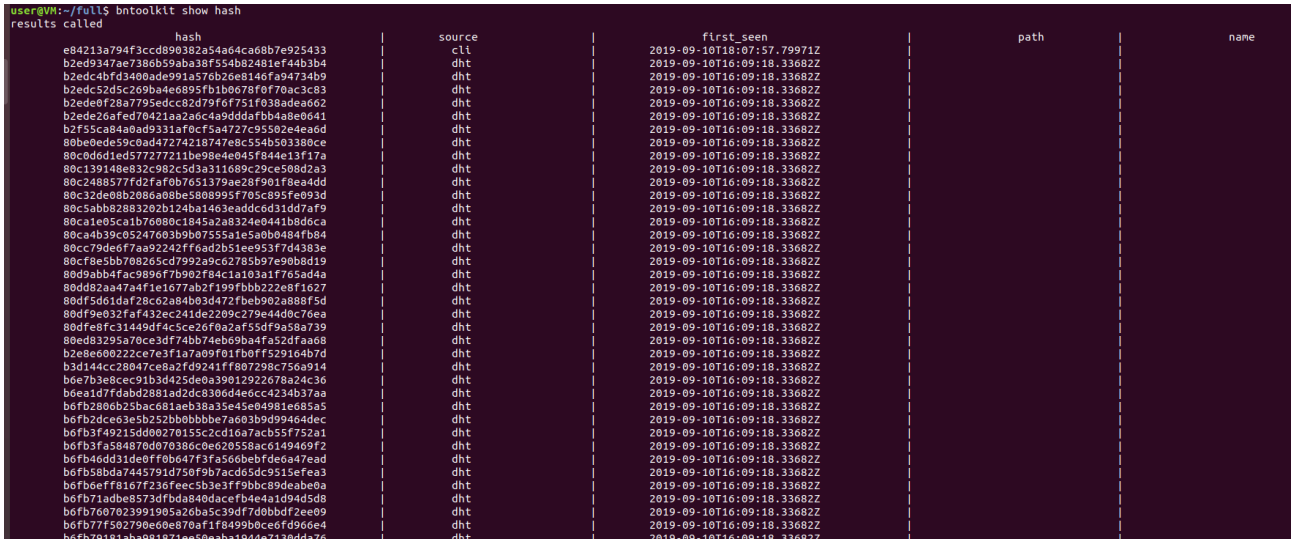

Figura A.14: bntoolkit show hash

A.3.16.2 alert

Muestra las alertas almacenadas en la base de datos.

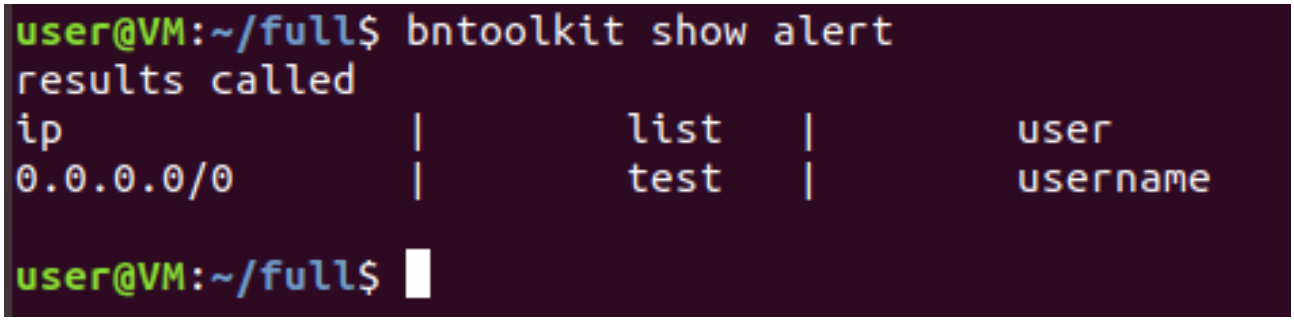

Figura A.15: bntoolkit show alert

A.3.16.3 count

Cuenta el número de hashes almacenados en la base de datos.

```
user@VM:~/full$ bntoolkit show count
results called
count
3629
```

Figura A.16: bntoolkit show count

A.3.16.4 ip

Muestra todas las IPs almacenadas en la base de datos.

```
user@VM:~/full$ bntoolkit show ip
results called
ip
1.1.0.0/16
180.0.0.0/5
0.0.0.0/0
92.249.240.52
190.191.248.180
79.112.81.250
84.39.117.57
83.55.205.239
183.87.222.69
146.199.135.170
188.254.32.227
185.245.87.43
37.60.18.161
185.149.90.66
77.47.116.54
80.191.221.50
```

Figura A.17: bntoolkit show ip

A.3.16.5 monitor

Muestra todos los hashes almacenados en la base de datos para monitorizar.

```
user@VM:~/full$ bntoolkit show monitor
results called
hash                                    |        username       |        projectName
e84213a794f3ccd890382a54a64ca68b7e925433        |        default        |        default

user@VM:~/full$
```

Figura A.18: bntoolkit show monitor

A.3.16.6 possibles

Muestra la tabla de posibles de la última ejecución de find.

```
user@VM:~/full$ bntoolkit show possibles
results called
id      |                       hash                       |    download   |        valid  |        Possible
32      |        fc1c93e5b0e6447ce1539985314f41a40fdf873a        |        false  |        false  |        true
26      |        6ee823771df0234b85fde7c41cade1259172aaac        |        false  |        false  |        true
42      |        92a6963b6976c9d0ea7d73cab881cdc3d6d62f3b        |        false  |        false  |        true
44      |        c4b00e295baa7036953e6c9826372267eda7c8af        |        false  |        false  |        true
48      |        3b03498a64b2f788e66878b4388367fffddbdf34        |        false  |        false  |        true
49      |        44f999fe0f7e550d092bb30f73376f08a807fba4        |        false  |        false  |        true
31      |        568651d93721750427bc707e550519243d225e31        |        false  |        false  |        true
45      |        d5b89b18d7ddeb7d21fd160b4c34ffebd8b094f8        |        false  |        false  |        true
35      |        b0eb1dd2ab6d99fe3e2290084960824c8ad42129        |        false  |        false  |        true
22      |        19362ff5666e0a734e13209224ad66ebcc4cd7d2        |        false  |        false  |        true
29      |        1aa0528dd1b93fd8f07b04deedd026739a74f146        |        false  |        false  |        true
47      |        49126275ab8708cc334cde0015b4659fc899c673        |        false  |        false  |        true
40      |        6321cdb0e58708a0056506327c2e7ecd2b7c7e72        |        false  |        false  |        true
33      |        6d77277d238dcf51519617d5115b058ce8949257        |        false  |        false  |        true
28      |        3c7670fb5d66fbd507f234ae0d13a103d4270de9        |        false  |        false  |        true
27      |        cac06a5fa4157f6e780a2299f2163c8a2dafa227        |        false  |        false  |        true
36      |        74c3bf2a9114953c9a9c0c7a2b20db2765284ae8        |        false  |        false  |        true
25      |        5a32cae3c37f11145286c912d10f254fc029f77f        |        false  |        false  |        true
38      |        bb00353cbe800ca2f57838b42d2857c01e4eda88        |        false  |        false  |        true
34      |        35e3293a4f1e942f8d5c6c48002aa5d8aad22ce3        |        false  |        false  |        true
30      |        0100f788b184b828ceda343f742b79ea3ee52465        |        false  |        false  |        true
18      |        b638714a18b4ee63892712f27aee70c798e3efbc        |        false  |        false  |        true
16      |        1533f2cde863bcc9935b870eb959dcd91de753d2        |        false  |        false  |        true
46      |        41dc25cbe087ba1b834fda6ac9fb549bf1d115ac        |        false  |        false  |        true
21      |        a6e7ec0db9d5fed97efb70ec2a4ab50e4d106cfd        |        false  |        false  |        true
20      |        44a4106316f1b58aa3c0a8f1ac607e5b1445b19c        |        false  |        false  |        true
39      |        2a20018b91cfe962f6da3f8cfe50ffbcf4eee5f1        |        false  |        false  |        true
41      |        11afed5b0717d0cfbb61ff2e3eb65ec58e4c912a        |        false  |        false  |        true
24      |        3db3ffd894306b201e59172b3660010c95796e34        |        false  |        false  |        true
17      |        4410e4a57ece50f38f2d758a78a0f73a3f06af6e        |        false  |        false  |        true
11      |        5846f7ca3670cef53424d97f5f38fe342d54eda7        |        true   |        true   |        true
37      |        63b1ac46f22548afd81e7dbc61a6f3b571f6d2b6        |        false  |        false  |        true
```

Figura A.19: bntoolkit show possibles

A.3.16.7 project

Muestra los nombres de todos los proyectos almacenados en la base de datos.

```
user@VM:~/full$ bntoolkit show project
projects called
projectName     |        date
default         |        2019-09-10T16:07:27.093244Z
```

Figura A.20: bntoolkit show project

A.3.17 Ejecución en Windows

En la siguiente imagen se puede ver la ejecución del programa en Windows utilizando igualmente Docker. Se han realizado unicamente cambios en la ruta para que el fichero de configuración esté en la ruta de ejecución.

```
PS D:\shared\bntoolkit> docker run -p 5432:5432 --name yourContainerName2 -e POSTGRES_PASSWORD=postgres99 -d postgres
b2be2bfea9d425387abb6c79270030ebd3feb4779018060b5e3fc2a905d172d5
PS D:\shared\bntoolkit> .\bntoolkit.exe initDB
initDB called
PS D:\shared\bntoolkit> .\bntoolkit.exe insert e84213a794f3ccd890382a54a64ca68b7e925433
insert called
PS D:\shared\bntoolkit> .\bntoolkit.exe show hash
results called
                    hash                          |       source          |              first_seen                |              path          |                name
       e84213a794f3ccd890382a54a64ca68b7e925433       |          manual               |       2019-09-16T20:55:33.874745Z          |                            |

PS D:\shared\bntoolkit> .\bntoolkit.exe -h
Set of utilities to monitor, download, create and find files in the BitTorent network

Usage:
  bntoolkit [command]

Available Commands:
  addAlert      Add an IP to the database alert table
  addMonitor    Add a hash to the database monitor table
  crawl         Crawl the BitTorrent Network to find hashes
  create        Create a .torrent file
  daemon        Start the daemon to monitor the files in the monitor table and notify alerts.
  deleteAlert   Delete an IP or range from the database alert table.
  deleteMonitor Delete a hash from the database monitor table.
  download      Download a file from a hash, a magnet or a Torrent file
  find          Find the file in Bittorrent network
  getTorrent    Get torrent file from a hash or magnet
  help          Help about any command
  initDB        Create the database and its tables
  insert        Insert a hash or a file of hashes in the DB
  show          Show the database data
  version       Print the version number

Flags:
  -c, --cfgFile string   Config file to DB (host, port, user, password) (default "configFile.toml")
  -d, --debug            debug output
  -h, --help             help for bntoolkit
  -v, --verbose          verbose output

Use "bntoolkit [command] --help" for more information about a command.
PS D:\shared\bntoolkit>
```

Figura A.21: bntoolkit en Windows

Apéndice B

Herramientas y recursos

Las herramientas necesarias para la elaboración del proyecto han sido:

- Sistema operativo GNU/Linux [59]
- Sistema operativo Windows [60]
- Compilador C/C++ gcc [62]
- Golang [47]
- Transmission [24]
- Docker [49]
- PostgreSQL [48]
- Toad Data Modeler [4]
- Visual Studio Code [63]
- LaTeX[64]

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá