P2P & CDNs
Estudio de Bittorrent

Jonathan Hernández Velasco, Oscar Barrios Torrero
<jonathan.hernandez,oscar.barrios@estudiants.urv.cat>, *Grupo 7 – P2P, Universidad Rovira i Virgili*

En este estudio hablaremos del funcionamiento básico de Bittorrent, profundizando en su arquitectura basado en la comunicación entre peers siguiendo una estrategia para compartir los contenidos propios de cada uno de ellos. Trataremos también la seguridad en este protocolo, así como distintas implementaciones de Bittorrent como el cliente oficial y otros que lo enriquecen en funcionalidad.

I. Introducción

Bittorrent es un protocolo de distribución de fichero Peer to Peer, que ofrece un rendimiento destacable respecto a sus competidores. Esto es debido a que el protocolo se centra en una optimización en la replicación y distribución de datos.

Tanto el protocolo Bittorrent como su cliente oficial están diseñados por Bram Cohen, actual Jefe Ejecutivo y cofundador de BitTorrent, Inc. y CodeCon. Antes de la creación de BitTorrent, Bram Cohen trabajó en MojoNation. MojoNation permitía a la gente subir a la red ficheros confidenciales en segmentos cifrados, de manera que se podían distribuir los ficheros entre las personas que tuvieran instalado el software de MojoNation. Este concepto sirvió como inspiración para el desarrollo de BitTorrent.

La popularidad de Bittorrent no se debe únicamente al buen rendimiento ofrecido por el protocolo, sino también a la apuesta de Bram Cohen de ofrecer su cliente Bittorrent como *código libre* escrito en Python. Esto permitió que fanáticos de este lenguaje, que en estos momentos se encuentra en auge, diseñaran otros clientes, para otras plataformas, en otros lenguajes y con nuevas funcionalidades. Permitiendo así un acercamiento de Bittorrent a millones de usuarios.

Como desventaja de este sistema hablaremos de la falta de implementación de un sistema de búsqueda de contenidos que, por su arquitectura, resulta algo compleja de centralizar dicha información. Actualmente, el *bootstrap* a un fichero compartido en la red Bittorrent se encuentra en servidores Web que almacenan este tipo de fichero de conexión para su descarga vía HTTP o FTP.

Mantener estos ficheros en páginas Web implica varios inconvenientes, nos encontramos ante un objetivo de ataque por parte de personas no interesadas en la divulgación del fichero que en el servidor Web se ofrezca y además nos encontramos ante un problema de actualización de estos ficheros, ya que no hay forma de que el servidor Web conozca el estado del contenido que ofrece (los ficheros compartidos en una red Bittorrent tienen un ciclo de vida).

Por tanto, podemos el protocolo Bittorrent aunque gana en velocidad de descarga a sus competidores, tiene dos limitaciones importantes: búsqueda de contenidos y compartir datos que no sean populares o novedosos.

II. ARQUITECTURA

1. Estrategia de participación

Los peers siguen una estrategia cuando comparten partes de un fichero con otros peers de la red, de tal manera que ofrecen un ancho de banda mayor a aquellos peers que le ofrecen un velocidad de transmisión mayor, esta estrategia es una variante del dilema del prisionero, también conocido como “Tit for Tat”. Su nombre proviene de la expresión inglesa "tip for tap", que significa "represalia equivalente". En esta estrategia, un peer responderá consecuentemente a la acción previa del peer que participa en el mismo fichero compartido. Si el oponente ha

cooperado previamente, el peer cooperará. Si el oponente deja de compartir, el peer se "vengará" de él, dejando de compartir el fichero con este peer.

Esta estrategia depende de tres condiciones:

1. El peer siempre colabora, a menos que sea provocado con una denegación para compartir
2. El peer se vengará ante cualquier denegación.
3. El peer perdona fácilmente una vez que se ha vengado, negándose durante cierto tiempo a compartir el fichero con este peer.

2. Funcionamiento del sistema

Los elementos que interactúan en este sistema son: servidores Web, ficheros .torrent, tracker y peers (leechers o seeders). Los pasos a seguir para compartir un fichero son:

1. Descarga un fichero .torrent vía HTTP o FTP, este fichero contiene la información necesaria asociada al fichero compartido.
2. Conectarse con el traker (incluido en el .torrent), sistema que nos proporciona una lista de peers que comparten el fichero que queremos descargar.
3. Crear una conexión con cada peer, los peers comparten el fichero siempre que compartas aquello que tengas disponible sobre ese fichero.

A continuación mostramos un esquema visual de la arquitectura de Bittorrent:

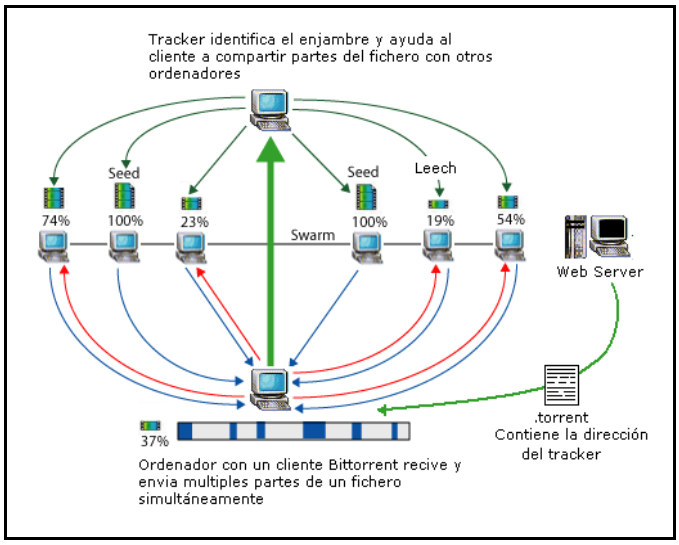

Figura 1. Esquema de protocolo Bittorrent

La comunicación entre los distintos componentes de esta arquitectura a nivel de red, tenemos conexiones HTTP petición/respuesta para los mensajes que se envían entre un peer y el tracker y conexiones TCP o UDP mediante sockets para establecer un intercambio de datos entre dos peers.

La información contenida en el fichero bittorrent así como la ofrecida por un tracker esta codificada utilizando B-encoding un método de codificación de metainformación diseñado para Bittorrent. B-encoding soporta cadena de bytes, enteros, listas y diccionarios, con estos tipos permite construir estructuras para guardar eficientemente todos estos datos:

- Cadena de bytes: <longitud del string>:<string>
  Ejemplo: 6:string
- Enteros: i<entero>e
  Ejemplo: i6e
- Listas: l<datos en formato b-encoded>e
  Ejemplo: l4:dadai8ee
- Diccionarios: d<string><elemento>
  Ejemplo: d5:clau15:valor5:clau2i12ee

En concreto un fichero torrent guarda la siguiente información: Donde se encuentra el tracker (URL), a que esta enlazado el fichero torrent, si contiene una colección de ficheros, el tamaño de cada fichero, define también el tamaño en bytes de una parte, y por último también contiene una lista de claves hash por cada parte del fichero compartido. Las claves hash son generadas utilizando SHA-1 con un resumen de 160 bits y un tamaño máximo por parte de 2^64 bits, este conjunto de claves se utiliza como mecanismo para asegurar la integridad y consistencia de una parte, una vez a sido completada la descarga de dicha parte.

3. Tipos de mensaje

En la comunicación entre peers se establece un conjunto de identificadores de mensaje, cada tipo de mensaje ofrece una característica de interacción entre peers, incluyendo un conjunto de mensajes que ofrecen soporte para implementar la estrategia "Tit for Tat" y la transmisión de segmentos de fichero entre los interlocutores. Pasamos a detallar cada uno de ellos, describiendo brevemente lo que transmiten:

keep-alive <len=0000>
Este mensaje no tiene identificador ni datos, consiste en un campo de longitud cero cuya función es mantener activa la conexión entre dos peers cuando no hay actividad.

choke <len=0001> <id=0>
Este mensaje no contiene datos. Indica que el cliente tiene el peer en estado *choked*. Al recibir este mensaje, un peer debe de decidir si le interesa mantener abierta la conexión, ya que no podrá pedir bloques de datos a este peer mientras este en ese estado. Normalmente el algoritmo cierra la conexión a no ser que este peer haya compartido una cantidad considerable de información, con lo que esperaríamos un tiempo prudencial antes de cerrar la conexión.

unchoke <len=0001> <id=1>
Este mensaje no contiene datos. Indica al cliente que el peer ha dejado de estar en estado de choked. Con lo que puede volver a pedir bloques de datos a este peer.

interested <len=0001> <id=2>
Este mensaje no contiene datos. Indica al peer receptor que el emisor esta interesado en los segmentos que comparte. Al recibir el mensaje, el peer debe comprobar que dispone de sufiente ancho de banda para realizar una transmisión. Si es así, enviará un mensaje de unchoke indicando que acepta peticiones, en caso contrario enviará un mensaje de choke.

not interested <len=0001> <id=3>
Este mensaje no contiene datos. Indica que el cliente no esta interesado en el *peer* receptor del mensage. La respuesta puede ser un mensaje *choke* para forzar al otro *peer* a enviar un mensaje *interested* en el caso de que mas adelante quiera iniciar una descarga.

have <len=0005> <id=4> <piece index>
Aquest missatge indica l'índex d'un fragment (*piece*) de l'arxiu que ha estat descarregat correctament. La seva funció principal és que els altres peers amb qui ens comuniquem mantinguin actualitzat el mapa de bits associat a aquest *stream* durant la connexió.

bitfield <len=0001+X> <id=5> <bitfield>
Este mensaje se envía solo inmediatamente tras la fase de Handshake, por parte del peer que ha recibido la petición de conexión, y es obligatorio. El campo de bits (de longitud X) representa los fragmentos que posee del flujo asociado. Lo recibirá el peer interesado en la descarga, y le servirá para saber si el otro peer dispone de fragmentos que él no tiene y le interesan, en este caso le podrá enviar un mensaje interested (y esperará la recepción del correspondiente unchoke). Por otra parte, si no sirve nuevos fragmentos, le enviará un notinterested y cerrará la conexión (no se espera que en un momento posterior este peer disponga de fragmentos que nos falten, puesto que las transmisiones se realizan ordenadas y en tiempo real).

request <len=0013> <id=6> <index> <begin> <length>
Utilizado para pedir un bloque de datos. Contiene el índice del fragmento, el offset donde se quiere iniciar la descarga indicado en bytes y la longitud del bloque a descargar. Al recibirlo, se deberá comprobar que anteriormente se le ha reservado una determinada tasa antes de enviar los datos solicitados por el canal UDP, a la tasa que le ha sido reservada.

retx_request <len=0013> <id=7> <index> <begin> <length>
Tiene exactamente la misma función y el mismo formato que un request normal. Pero este mensaje se utiliza para pedir la retransmisión de un bloque que ya se había solicitado antes y no se ha recibido, o si se han detectado errores de transmisión a partir de la comprobación del hash del fragmento correspondiente. La diferencia reside en qué esta vez, la transmisión del bloque se realizará por el canal TCP (en este caso es más importante asegurar la recepción del bloque que no la velocidad a la que sea retransmitido, puesto que estos datos no están destinadas a la reproducción en tiempo real).

piece <len=0009+X> <id=8> <index> <begin> <payload>
Utilizado por enviar un bloque de datos. Contiene el índice del fragmento al que pertenece, el punto donde empieza el bloque (un offset calculado en bytes) y los datos del bloque, de longitud X.

cancel <len=0013> <id=9> <index> <begin> <length>
Se utiliza para cancelar la petición de un bloque. Su campo de datos es idéntico al de los mensajes request y retxrequest. Al recibir este mensaje, tendremos que parar el envío del bloque asociado, si todavía no se ha hecho, o cancelarlo en caso contrario.

reserve <len=0009> <id=10> <bandwith> <time>
Su función es pedir la reserva de una determinada tasa de transferencia durante un tiempo determinado. Si el peer que recibe el mensaje no es capaz de proporcionar la tasa pedida, puede decidir entre no reservar ninguna tasa en absoluto o reservar una tasa inferior. En caso de ser posible una reserva, deberá realizarla para un tiempo igual al solicitado, independientemente de la tasa que se reserve.

reserve-ack <len=0009> <id=11> <bandwith> <time>
Es la respuesta al mensaje anterior. En este caso se indica la tasa de transferencia que se le ha reservado, que en principio, será igual o inferior a la solicitada. La duración de la reserva será igual a la del mensaje reserve que se envió. El peer que realiza la descarga deberá hacer continuas reservas para asegurarse una tasa durante toda la transferencia.

III.DHT TRACKER

Existen un añadido al protocolo de Bittorrent para mejorar notablemente uno de sus puntos débiles. Se trata del DHT Tracker, y trata de evitar el gran cuello de botella del protocolo Bittorrent: La comunicación con el tracker. El único punto centralizado del protocolo de Bittorrent es el tracker, lugar donde los nodos acuden al principio para situarse en la red y periódicamente para descubrir nuevos (y quizás mejores) vecinos para comunicarse. Gracias a esto el tracker puede ir guardando una serie de estadísticas y los nodos pueden ir buscando cuales son los vecinos que más les interese, bien porque tengan los datos que busca, porque obtiene más velocidad comunicándose con él (ping más bajo, que suele pasar cuando su ubicación geográfica es más cercana) o por las dos causas anteriores a la vez.

1. Funcionamiento

El DHT Tracker se encarga de distribuir la carga del Tracker (información de los vecinos que contiene) entre los propios nodos a partir de un DHT, teniendo como base el de Kademlia pero extendido y adaptado a las necesidades de Bittorrent.

Como concepto, se trata de una capa que trabaja por encima del protocolo de Bittorrent y que no pertenece al mismo. Para la comunicación, utiliza un puerto UDP y estable como ID del nodo el algoritmo SHA-1 aplicado a la IP + el Puerto que el nodo utiliza.

El DHT Tracker te permite, a parte de ahorrar mucho ancho de banda en la parte del Tracker, sobrevivir mucho mejor ante la caída del nodo que hace de Tracker. Antes de la implementación del DHT Tracker, la única manera de sobrevivir a esto era implementando una cache local en la parte del cliente con los nodos que conocía (IPs públicas). Esto significaba que si el Tracker caía, dependías de los nodos a los que te habías comunicado antes para poder terminar la descarga. Era aún posible terminarla pero el índice de éxito requería de varios factores (sobretodo que no se desconectaran y que entre todos contengan todas las partes del archivo que estamos descargando), y que no siempre se daban.
Después de la implementación del DHT Tracker, ante la caída del Tracker podemos sobrevivir comunicándonos con los vecinos ja que entre los propios nodos se guarda la información de la totalidad de los nodos.
Por lo tanto, estamos ante una descentralización del protocolo gracias a esta "capa". Pero, ¿podemos hablar entonces de que gracias a esto Bittorrent se podría considerar como totalmente descentralizado?. Aún es necesario bajarse el .torrent y conectarse al Tracker para empezar a conocer nodos y entrar a formar parte del DHT, así que los nuevos nodos (candidatos para entrar en la red) todavía dependen del tracker para poder entrar y por tanto existe una "centralización" todavía. Aunque existe una versión de DHT Tracker que soluciona este último problema, y que explicaremos en el siguiente punto.

2. Versiones

Actualmente existen dos versiones incompatibles muy extendidas de DHT Tracker implementadas para clientes de Bittorrent. Por un lado tenemos la implementada por los desarrolladores de Azureus, la primera que se hizo en un cliente Bittorrent. Y después esta la implementada en el cliente oficial de Bittorrent (http://www.bittorrent.com), a la que han seguido el resto de clientes como Bitcomet.

La implementación de Azureus contiene un característica extra curiosa. Se trata de los Magnet Links, creados para solucionar el problema del Bootstrap (entrada) de nodos a la red cuando el Tracker esta caído. Contienen una dirección para entrar directamente en la red de DHT y empezar a comunicarse con los nodos a partir de esta (por tanto, quitando la necesidad del Tracker). Este detalle de descentralización es el punto donde más lejos se ha llegado, actualmente en funcionamiento, en cuanto a descentralización con el protocolo de Bittorrent. Aún así, para el primer nodo que entra en la red, es necesario que entre en comunicación con el Tracker para compartir el archivo.

3. Otras utilidades

A parte de las ya comentadas, existen otras características para las que también se puede aprovechar el DHT existente, como por ejemplo el uso de "puntuación" o "comentarios" sobre el contenido que se esta descargando. Este tipo de características, implementadas por ejemplo en el cliente Azureus, da un nivel de participación y autoregulación de contenido a los propios usuarios, mientras que la información se guarda uniformemente entre los nodos conectados.

4. Algunas consecuencias

El uso del DHT y la comunicación entre nodos sin el paso por el Tracker comporta un independencia del nodo que a veces puede no interesar, como en el caso de los Trackers privados, donde se suelen aplican ciertas restricciones al usuario según sus estadísticas y donde es requisito común el estar registrado (como mínimo) para poder entrar en la red. El DHT no esta preparado para este tipo de restricciones, y por tanto la comunicación entre nodos elude todo tipo de restricción (las restricciones se hacen a nivel de Tracker).
Debido a esto, y para evitar que los administradores de los Trackers privados banearan a los usuarios de clientes con DHT activado, se ha añadido la posibilidad de establecer un nuevo parámetro en el archivo .torrent llamado "Private Flag". Cuando el cliente lee esta opción automáticamente desactiva el DHT para esa descarga. Este tipo de parámetro es compatible

con todos los clientes, así si uno no implementa el DHT lo omitirá sin más.

IV.COMPORTAMIENTO

1. Escalabilidad

Esta demostrado que el Tracker de Bittorrent consume 1/1000 parte del tráfico de toda la red. Considerando la cantidad de ancho de banda que se puede desplegar en una red de este tipo, es un valor nada despreciable. Por otro lado, el tipo de arquitectura y manera de distribuir contenido en Bittorrent hace que contra más nodos hayan conectados más rápida va la descarga (es decir, escala muy bien) siempre y cuando ese 1/1000 sea asumible por el Tracker. Por lo tanto, se puede concluir en que la escalabilidad depende del ancho de banda del Tracker, y que la descentralización del Tracker a partir del uso de un DHT Tracker contribuye muy positivamente.

2. Robustez y tolerancia a fallidas

La caída de un Tracker comporta que los nuevos nodos no puedan conectarse y que los nodos que ya estan conectados no puedan descubrir nuevos nodos a parte de los que ya estan conectados. Conceptualmente se forman pequeñas islas desorganizadas, donde en realidad todos los nodos estan conectados entre sí a través de vecinos pero no hay comunicación entre esas islas (la comunicación entre ellas las gestionaba el Tracker) y por tanto influye muy negativamente en la descarga. Una mejora fué la implementación de caché local de nodos conocidos en los clientes Bittorrent, y más tarde mejor solución fué el DHT Tracker + Magnet Links.

3. Ataques sobre Bittorrent

Bittorrent tiene un protocolo considerado de los de modelo económico. Es inocente, no esta preparado para clientes maliciosos, y por tanto el protocolo no presenta soluciones para posibles ataques a través de este tipo de clientes. Vamos a ver algunos ejemplos:

Denial of service via a Sybil attack: Cuando para atacar se utilizan múltiples identidades se denomina “Sybil attack”.
Cada cliente Bittorrent, a partir de la IP y el reloj, genera un hash que utiliza para identificarse en la red Bittorrent. Un cliente malintencionado podría generar muchas identidades y acaparar toda la “preferred peers list” de un nodo. La PPL es la lista de nodos a los que el nodo que la posee tiene más interés bien por velocidad de acceso alta o por contenido (o los dos). Si acapara toda la PPL de un seed, este sólo serviría el archivo al nodo atacante (como si no estuviera en la red).

Seed only attack: Al recibir la lista de peers, el cliente sólo se conecta a aquellos que son seed (semillas). ¿Como se pueden diferenciar? Muy fácilmente, son aquellos que tienen el bit Bittfield todo a 1 (uno), es decir, que poseen todo el contenido del fichero. De esto modo un cliente podría bajarse el archivo entero sin contribuir nada de subida. Según el protocolo Bittorrent, para bajar hay que subir (equivalencia del TIT for TAT), y esto es cierto salvo para los seeds, que sirven sin pedir nada a cambio.

Minimous upload attack: En Bittorrent se recibe el mismo incentivo si subes a 1Kb/s que a 10Kb/s, por lo tanto el objetivo del atacante es subir lo mínimo para entrar en la PPL del nodo.

Malicious upload attack: Se trata de enviar siempre al otro cliente trozos de datos corruptos. El nodo receptor no se da cuenta de ello, sinó que cree que le he enviado un paquete repetido. Descarta el paquete, pero no castiga al nodo (no lo saca de su lista de nodos preferidos (PPL).

Ataques como el último se pueden evitar poniendo soluciones a través del cliente, como por ejemplo establecer un máximo de paquetes de erróneos desde otro nodo antes de sacarlo de la PPL.

V.LIMITACIONES

1. Incentivos

La gran limitación del protocolo Bittorrent es la falta de incentivos. Por ejemplo, un seed (semilla), que es un nodo que ya ha terminado la descarga, no tiene ninguna motivación para seguir conectado. Es decir, la única razón por la que un nodo hace de sed es por altruísmo. Y, hablando de altruismo, se conoce que de un 20% a un 40% de usuarios de Napster compartían poco o nada. Un nada esperanzador 70% en caso del Gnutella.

Como resultado, en el caso de Bittorrent, la mayoría de nodos salen de la red al terminar (o al poco) la descarga, bien por desconocimiento o indiferencia. La consecuencia directa de esto se da en el rendimiento. Mientras que una descarga va muy rápida al poco de vida, llega un momento que el ancho de banda general decae muy rápidamente, justo cuando la mayoría se convierte en seed y abandona la red.

Para solucionar este tema, hay dos frentes:

- La Netickette: Algo así como la etiqueta en internet, donde se establecen una serie de reglas para que todo marche bien. Una de ellas concerniente al protocolo Bittorrent es que no abandones la red hasta que tu índice de compartición esté a 1.0 (es decir, hayas subido tanto como bajado).
- Private Trackers: Los Trackers privados establecen mecanismos restrictivos en los que si tu índice de compartición esta por debajo de X, no te dejan descargar (incluso te pueden echar indefinidamente). Estos se suelen basar en asociar un sobrenombre a una IP (cada vez que te conectas a su página web poniendo tu usuario y contraseña te actualizan la IP en caso de que sea dinámica).

2. Random chunks

Se establece en el protocolo Bittorrent que, por defecto, la descarga de partes se hace a través de un algoritmo pseudo-aleatorio, en el que tienen prioridad los trozos “más raros” (que los tiene menos nodos). También tiene un modo “Supercompartición”, donde se priorizan las partes que todavía no han sido descargadas (al principio de poner un archivo en la red Bittorrent).

Este tipo de algoritmos benefician a la red en velocidad y repartición óptima de contenido para distribuir el ancho de banda entre nodos, pero no permite, en caso de contenido multimedia como video o audio, mostrarlo antes de finalizar la descarga.

Para solucionar esto, existen en algunos clientes como Azureus la posibilidad de priorizar la primera parte del archivo.

3. Bootstrapping

Bittorrent sigue un algoritmo TIT for TAT, por lo tanto si un peer no comparte información no podrá descargar. ¿Qué pasa si un nodo no puede dar información porqué no tiene? Es decir, ¿qué pasa con los nodos que acaban de entrar en la red?

Para solucionar esto existe lo que se conoce como “Optimistic Unchocking”. Cada nodo tiene su lista de nodos preferidos o preferred peer list (PPL), donde guardan la información de los nodos más rápidos que contienen las partes de información que le interesa. A parte de eso, cada nodo guarda un % de ancho de banda de subida para, periódicamente, descubrir nuevos nodos que puedan ser más rápidos que los actuales. Este ancho de banda de subida entregado sin esperar nada a cambio es aprovechado por los nodos nuevos para adquirir información suficiente como para poder redistribuirlo y formar parte de la red como uno más.

Estabilizarse como nodo, tener un PPL estable, suele tardan entre 5 y 10 minutos de media.

VI.REFERENCIAS

[1] Robustness of Bittorrent. Karthik Tamilmani. 2003

[2] Analysis of BitTorrent and its use for the Design of a P2P based Streaming Protocol for a Hybrid CDN. Karl-Andre Skevik, Vera Goebel, Thomas Plagemann.

[3] Incentives Build Robustness in BitTorrent. Bram Cohen. 2003

[4] BitTorrent Traffic Measurements and Models. David Erman. 2005.

[5] BitTorrent Session Characteristics and Models. David Erman, Dragos Ilie, Adrian Popescu.
[6] Analyzing and Improving BitTorrent Performance. Ashwin R. Bharambe. 2005.