Proyecto Fin de Carrera

Ingeniería en Informática

CENSURA EN BITTORRENT

Ismael Saad García

Director: Björn Knutsson
Ponente: Sergio Ilarri Artigas

Departamento de Informática e Ingeniería de Sistemas
Centro Politécnico Superior
Universidad de Zaragoza

Telecommunication Systems Laboratory
Information and Communication Technology
Kungliga Tekniska Högskolan – Royal Institute of Technology
Estocolmo (Suecia)

Noviembre 2010 – Curso 2010/2011

Resumen

BitTorrent es, hoy en día, una de las redes P2P (Peer-to-Peer) de compartición de objetos más populares. Tiene millones de usuarios. BitTorrent proporciona un mecanismo eficiente para compartir objetos entre un gran número de clientes, incentivando a aquellos que descargan un objeto a compartirlo con el resto.

Para obtener peers con los que intercambiar un objeto, las versiones más recientes de BitTorrent empiezan a incorporar el uso de un DHT (Distributed Hash Table). El DHT es un mecanismo para distribuir el almacenamiento de las listas de peers participantes en la distribución de un objeto entre todos los nodos participantes en la red P2P. BitTorrent tiene dos DHTs: Mainline DHT y Azureus DHT. Este proyecto se centra en el estudio de Mainline DHT.

Concretamente, este proyecto se centra en el estudio de la generación, distribución y obtención de valores en Mainline DHT. En primer lugar, se presenta un análisis teórico de esta parte concreta del DHT y, posteriormente, se contrasta con el comportamiento real. Se identifican situaciones inesperadas y casos en los que el rendimiento del DHT se podría mejorar.

Además, de acuerdo con el análisis que se presenta, hay situaciones en las que el DHT es vulnerable, haciendo posible: censura mediante la denegación a nodos del acceso al intercambio de un objeto, encaminamiento de tráfico a modo de ataque DDoS (Distributed Denial of Service) y un problema de escalabilidad. Se han comprobado estos problemas experimentalmente y se incluye una documentación de los mismos. El análisis ha ayudado a diseñar algunos experimentos que muestran la robustez del DHT contra la censura y, por otro lado, un serio problema de escalabilidad.

Para llevar a cabo los experimentos, se ha desarrollado una colección de herramientas que sirve para monitorizar aspectos concretos del DHT. Estas herramientas son Open Source de modo que se puedan utilizar y ampliar para llevar a cabo más experimentos.

Agradecimientos

Quiero dar las gracias a mi supervisor, Raúl Jiménez, y a mi director, Björn Knutsson, por toda la ayuda que me prestaron en Estocolmo durante el desarrollo de este proyecto. Doy también las gracias a mi compañera de departamento, Sara Dar, por su apoyo.

Gracias a mi padre, Mahmoud Saad, a mi madre, Teresa García, y a mi hermano, Carlos Saad, que siempre me apoyaron desde la distancia.

Gracias a todos los amigos que estuvieron junto a mí durante este tiempo tan maravilloso: David, Javier, Daniel C, Miguel, Peio, Silvana, Luigi, Elena, Daniel W, Jens, Sebastian, Alexandra, Paolo, Víctor, Adrian, Marta, Marcos, Tony, Xandra, Luis, Pablo, Beatriz y José.

Finalmente, gracias a todos aquellos que me apoyaron durante la recta final de este proyecto.

Prólogo

El proyecto "Censura en BitTorrent" ha sido desarrollado enteramente en el Real Instituto Tecnológico de Estocolmo (Kungliga Tekniska Högskolan - Royal Institute of Technology). Al final de la memoria se adjunta una copia original escrita en inglés.

El presente texto es una adaptación al español de la memoria original. Dada la inexistencia o imprecisión de una traducción directa para algunos de los términos utilizados, se ha decidido mantener los nombres originales. Además, los anexos están escritos en inglés. Únicamente, han sido traducidos al español sus títulos, pero no su contenido (texto, secciones y pies de figuras y tablas).

Este proyecto se presentó en el Real Instituto Tecnológico de Estocolmo el día 5 de noviembre de 2010. Fue evaluado por el Doctor y Profesor Don Björn Knutsson (Departamento "Telecommunication Systems Laboratory") con una calificación de A. Las calificaciones siguen una escala de seis valores (A, B, C, D, E, F) donde A es la nota más alta y F es suspenso.

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Las redes P2P (Peer-to-Peer) son un tipo de red donde todos los participantes tienen las mismas responsabilidades. BitTorrent [1] es, hoy en día, una de las redes P2P más populares. Es utilizada por millones de usuarios.

Este proyecto explora la generación, distribución y obtención de valores en un DHT (Distributed Hash Table) de BitTorrent. En primer lugar, se presenta un análisis teórico de algunos aspectos del DHT que es contrastado posteriormente con resultados experimentales.

El análisis muestra además algunas posibles vulnerabilidades del DHT, las cuales son: censura de contenidos, uso del DHT para llevar a cabo ataques DDoS (Distributed Denial of Service) y un problema de escalabilidad. Estas vulnerabilidades han sido comprobadas experimentalmente.

En las redes P2P, el servicio debería estar distribuido equitativamente entre todos los participantes, pero las vulnerabilidades estudiadas implican un desequilibrio en esta equidad. Por tanto, estudiar vulnerabilidades en una red P2P como BitTorrent puede mejorar significativamente el rendimiento y seguridad del software en beneficio de millones de usuarios.

1.1 Introducción al contexto tecnológico

Las redes P2P (Peer-to-Peer) son un tipo de red en las que la responsabilidad del servicio se distribuye entre todos los nodos participantes en la misma. BitTorrent es una de las redes P2P más populares hoy en día. Su propósito es proporcionar un mecanismo distribuido, escalable y robusto para compartir objetos.

En BitTorrent, los objetos se obtienen siguiendo tres pasos: descubrir el objeto, obtener nodos con los que intercambiar el objeto (llamados peers) y, finalmente, comenzar la descarga del objeto así como compartir con los peers las partes del mismo que se tienen. La lista de peers se obtiene pidiéndosela a un servidor centralizado llamado tracker.

La existencia de trackers como entidades centralizadas puede suponer problemas de escalabilidad y robustez. Para prevenir estos problemas, las versiones más

recientes de BitTorrent tienden a prescindir de los trackers delegando su tarea de forma progresiva a un DHT (Distributed Hash Table). BitTorrent tiene dos DHTs, ambos basados en Kademlia [2]: Mainline DHT [22] y Azureus DHT [23]. Este proyecto estudia Mainline DHT.

Mainline DHT es un mecanismo distribuido que gestiona las listas de los peers que están descargando o que contienen los objetos. Cada objeto tiene un identificador llamado info hash. Cuando un participante (llamado nodo) quiere descargar un objeto, envía un mensaje (llamado announcement) al conjunto de los nodos más cercanos al info hash para ser incluido en la lista de peers. Por lo tanto, los nodos que almacenan una lista de peers son elegidos de forma determinista.

Cuando un nodo quiere obtener la lista de peers de un objeto, pregunta iterativamente a los nodos conocidos más cercanos al info hash de dicho objeto sobre otros nodos más cercanos, hasta que encuentra aquellos que contienen la lista de peers.

1.2 Alcance del proyecto

La primera parte de este proyecto perfila una parte del DHT. Cuando un nodo comienza la descarga de un objeto, envía annoucenments periódicamente al conjunto de los nodos más cercanos al info hash del objeto para ser incluido en la lista de peers. Centrándose en este aspecto, este proyecto estudia: cuántos nodos contienen la lista de peers, cuántos nodos se eligen para enviarles announcements y cómo son elegidos, cuál es el período de los announcements, cómo las listas de peers están distribuidas entre los nodos y la distancia al info hash de los nodos que contienen la lista de peers.

En primer lugar, se han analizado los resultados esperados en base al modelo teórico de Mainline DHT. La necesidad de hacer una comprobación empírica radica en que, aunque el análisis puede proporcionar una aproximación razonable, puede no concordar con el comportamiento real del DHT por dos motivos:

- La coexistencia de diferentes clientes en el DHT: Mainline DHT es una red P2P abierta, de modo que cualquier implementación de un cliente del DHT puede unirse. Hoy en día coexisten varios clientes distintos. Por tanto, si tienen comportamientos diferentes, pueden influir en el estado del DHT.

- La popularidad de algunos objetos: objetos con un gran número de peers pueden implicar un desequilibrio en el DHT (un objeto con 10 peers puede influir en el DHT de manera muy diferente a otro con un millón de peers). Este desequilibrio puede alterar el comportamiento del DHT.

La segunda parte de este proyecto consiste en el estudio de algunas posibles vulnerabilidades del DHT deducidas a partir del análisis. De nuevo, el análisis indica que son reales, pero es necesario comprobarlas empíricamente ya que, como se ha explicado, el comportamiento real del DHT puede diferir del teórico.

Todas las vulnerabilidades son un punto de vista diferente del mismo escenario. Como se ha explicado antes, las listas de peers de los objetos se almacenan en un conjunto de nodos elegidos de forma determinista (los más cercanos al info hash). Los nodos son capaces de elegir su posición en el DHT (como se explica detalladamente en el Capítulo 3) y, por tanto, pueden posicionarse en el DHT cerca de un info hash para así contener la lista de peers de cualquier objeto. Esto puede dar lugar a las siguientes vulnerabilidades:

- Censura: si un conjunto de nodos acuerda posicionarse cerca de un info hash, puede llegar a ser la única entidad responsable del almacenamiento de la lista de peers de un objeto. Controlando la lista de peers de un objeto, estos nodos podrían denegar a otros nodos el acceso a la descarga del objeto.
- Ataques DDoS: una lista de peers es un conjunto de pares (direcciones IP, puerto UDP). Al obtener una lista de peers, los nodos se comunican con las máquinas cuyas direcciones están contenidas en la lista. Por tanto, si un nodo contiene la lista de peers de un objeto popular, proporcionando una lista de peers con la dirección IP de una víctima, puede encaminar hacia la víctima parte de su tráfico entrante en forma de ataque DDoS.
- Escalabilidad: si un nodo contiene la lista de peers de un objeto muy popular, la tasa de tráfico que se generará en él puede llegar a ser crítica. En el DHT hay millones de nodos. Por tanto, si un objeto llega a ser muy popular, su lista de peers será enorme y, en consecuencia, el número de peticiones de peers recibidas en los nodos que contienen su lista de peers será muy elevado.

Este proyecto presenta además una colección de herramientas desarrolladas para estudiar el comportamiento real y las vulnerabilidades del DHT de empíricamente.

1.3 Objetivos

Los objetivos que este proyecto pretende alcanzar son los siguientes:

- Comparar el análisis teórico con el comportamiento real de Mainline DHT.
- Proporcionar una perspectiva experimental de la generación, distribución y obtención de listas de peers en Mainline DHT.
- Proporcionar una colección de herramientas para monitorizar Mainline DHT.
- Explorar el problema de censura en Mainline DHT.
- Comprobar y cuantificar los problemas de escalabilidad y ataques DDoS en Mainline DHT.

Estos objetivos han sido alcanzados y se puede ver su cumplimiento en el Capítulo 6. Además, el Apéndice U muestra la planificación y el tiempo real que llevó todo el trabajo.

Capítulo 2

Contexto tecnológico

Este capítulo presenta el contexto tecnológico del proyecto, profundizando poco a poco partiendo desde un contexto general. Se complementa con el Apéndice S donde se presentan otros trabajos con contextos cercanos al ámbito de este proyecto.

2.1 Sistemas distribuidos

El modelo clásico de servicios en red se ha basado tradicionalmente en la existencia de un servidor y varios clientes. En aquellos casos en los que un gran número de clientes solicita el mismo servicio, este modelo puede dar lugar a problemas de robustez y escalabilidad. Si un servidor deja de funcionar o demasiados clientes lo sobrecargan, el servicio que ofrece puede perder su disponibilidad.

Una forma de prevenir los problemas de robustez y escalabilidad es replicando el servidor, creando así una redundancia en el servicio. Esta solución mejora la robustez y escalabilidad del servicio. Sin embargo, los mismos problemas pueden persistir. La existencia de una entidad centralizada implica, en general, un límite en la robustez y escalabilidad de un servicio, ya que siempre puede fallar o sufrir sobrecargas (aunque la redundancia reduzca estos problemas).

Uno de los objetivos principales de las redes P2P (Peer-to-Peer) es tratar los problemas de robustez y escalabilidad proponiendo una arquitectura completamente distinta. Las redes P2P evitan o minimizan la existencia de entidades centralizadas mediante la distribución de las tareas de servicio entre todos los participantes.

2.2 Redes P2P y DHTs

Tal como se define en [4], "*Una red P2P (Peer-to-Peer) es un tipo de red en la que los participantes comparten una parte de sus recursos hardware. Estos recursos compartidos son necesarios para proveer el servicio y contenido ofrecido por la red. Con esta distribución, las redes P2P no requieren la intermediación o soporte de un servidor o autoridad global y centralizada*".

Un DHT (Distributed Hash Table) es un mecanismo para almacenar y recuperar valores indexados mediante claves. Esta tarea se distribuye entre todas las entidades participantes en él. Cada participante en el DHT (llamado nodo) es responsable del almacenamiento de un conjunto de valores cuya clave está dentro de un rango dado. El DHT proporciona el mecanismo para encontrar al nodo o conjunto de nodos responsables del almacenamiento del valor mediante su clave asociada. La metodología utilizada para gestionar el almacenamiento de los pares (clave, valor) es lo que define el diseño de un DHT. Entre los posibles diseños de DHTs destacan: Chord [5], Pastry [6] y Kademlia [2]. Kademlia es el estudiado en este proyecto dado que es el diseño del DHT en el que está basado este trabajo.

2.3 Kademlia

Kademlia es el diseño de un DHT. Cada valor tiene como clave asociada un identificador de 160 bits. Además, cada nodo elige un identificador de 160 bits cuando se une al DHT. Un valor se almacena en el conjunto de los nodos más cercanos a su clave; esta propiedad es uno de los aspectos de diseño más importantes para comprender el trabajo de este proyecto, ya que es la base de gran parte del análisis presentado en el Capítulo 3.

2.3.1 La distancia XOR

La distancia entre un identificador de nodo y una clave se mide como la función XOR bit a bit de ambas secuencias de 160 bits. La métrica XOR es simétrica, es decir, $distancia(x, y) = distancia(y, x)$. La Tabla 2.1 muestra algunos ejemplos de distancia XOR suponiendo identificadores de 8 bits. Además, esta métrica es unidireccional ya que, para cualquier nodo x y distancia $z > 0$, hay sólo un punto y tal que $distancia(x, y) = z$ (tal como se explica en [2]).

En los experimentos se ha utilizado la métrica de distancia logarítmica, una métrica más fácil de manejar y que es el logaritmo en base 2 de la distancia XOR. Esta métrica se explica más en profundidad en la Sección T.5.

X	00000001 (1)	00001010 (10)	00000001 (1)	11111010 (250)
Y	00000010 (2)	00010100 (20)	11001000 (200)	11111111 (255)
Distancia XOR	00000011 (3)	00011110 (30)	11001001 (201)	00000101 (5)

Tabla 2.1. Ejemplo de distancias XOR usando identificadores de 8 bits.

2.3.2 Mensajes

Todos los mensajes en Kademlia se envían utilizando el protocolo UDP. Kademlia tiene cuatro tipos de mensaje:

- PING: comprueba que un nodo está vivo.
- FIND_NODE: dada una clave, devuelve la dirección IP, puerto UDP e identificador de nodo de los **k** nodos más cercanos a la clave.
- STORE: manda a un nodo que almacene un par (clave, valor).
- FIND_VALUE: se comporta como FIND_NODE con una excepción, si se encuentra un nodo que había recibido previamente un mensaje STORE para la clave, devuelve el valor almacenado.

Para almacenar un par (clave, valor), un nodo debe encontrar los **k** nodos más cercanos a la clave y enviarles un mensaje STORE. Además, cada nodo debe publicar periódicamente los pares (clave, valor) que posee ya que estos lo borrarán pasado este tiempo.

2.3.3 La tabla de rutas

Cada nodo del DHT gestiona una tabla de rutas en la que almacena y actualiza información acerca de un conjunto de nodos conocidos. La tabla de rutas almacena, para cada $0 \leq i < 160$, la dirección IP, puerto UDP e identificador de **k** nodos a distancia $[2^i, 2^{i+1})$ de él mismo en lo que se llama k-bucket.

2.3.4 Descubrimiento de nodos

Cuando un nodo quiere encontrar aquellos nodos donde un valor debería estar almacenado (ya sea para recuperarlo o para enviarlo para su almacenamiento), debe saber la clave asociada al valor. Usando la clave, el nodo pregunta iterativamente a los nodos conocidos más cercanos a la clave sobre otros nodos más cercanos, hasta que encuentra aquellos que contienen el valor. Dado que la distancia XOR es unidireccional, todas las búsquedas convergerán en el mismo camino, independientemente del nodo que la lleve a cabo.

2.4 BitTorrent

BitTorrent [1] es una red P2P de compartición de objetos. Su objetivo es proveer un mecanismo eficiente de distribución de objetos a un gran número de clientes, incentivando a aquellos que descargan un objeto a compartirlo con el resto.

Un cliente participando en el intercambio de un objeto se llama peer del objeto. Cuando un cliente quiere descargar un objeto, necesita un fichero torrent (el cual contiene meta datos del objeto). Los ficheros torrent se pueden descargar, por

ejemplo, desde un servidor web (algunos de los distribuidores de torrents más importantes son The Pirate Bay [8] y Mininova [9]). Una vez obtenido el fichero torrent, el cliente puede empezar la descarga del objeto.

En BitTorrent, la obtención de un objeto se divide en tres fases:

1. Descubrir el contenido: obtener el fichero torrent del objeto.

2. Obtener peers: conseguir peers de los que descargar el objeto.

3. Descargar el objeto: descargar el objeto de los peers.

Los ficheros torrent contienen, opcionalmente, la URL de un tracker (que es un servidor centralizado que gestiona listas de peers). El cliente le pide al tracker peers del objeto y, comunicándose con ellos, comienza la descarga del este, así como el envío a otros clientes de las partes que posee. El conjunto de los peers que comparten un objeto se llama swarm. El intercambio del objeto se lleva a cabo utilizando el protocolo BitTorrent (explicado en [1]. Es importante aclarar que este protocolo tiene el mismo nombre que el software) el cual funciona sobre el protocolo TCP.

El uso de trackers como entidades centralizadas puede suponer problemas de escalabilidad y robustez. Las versiones más recientes de BitTorrent van delegando progresivamente la obtención de peers a un DHT. La mayoría de los clientes BitTorrent existentes, implementan tanto el protocolo BitTorrent como el protocolo del DHT. Aunque, en general, aún se utilizan los trackers para la obtención de peers, la mayoría de los clientes BitTorrent tienen compatibilidad con el DHT y utilizan ambos mecanismos. Algunos de los clientes de BitTorrent más populares son: UTorrent [26], BitSpirit [27] y KTorrent [28].

2.5 Mainline DHT

Existen dos DHTs en BitTorrent: Mainline DHT y Azureus DHT. Ambos son implementaciones de Kademlia. Este proyecto estudia Mainline DHT.

2.5.1 Una implementación de Kademlia

Como implementación de Kademlia, la mayoría de los aspectos del diseño de Kademlia están presentes en Mainline DHT. Sin embargo, algunos detalles (especificados en [20]) se añaden en la implementación:

- Las claves son un hash SHA-1 [25] de 160 bits llamado info hash calculado a partir del contenido del objeto. El uso de la función hash SHA-1 garantiza que es casi imposible que dos info hashes tengan el mismo valor. Además, por definición, las funciones hash aseguran una distribución uniforme en sus valores. El info hash está contenido en el fichero torrent del objeto.

- Elige k = 8 (como se ha citado anteriormente, k en Kademlia es el número de nodos almacenados en un k-bucket así como el número de nodos elegidos para enviar un mensaje STORE).

2.5.2 Listas de peers como valores

En Mainline DHT, los valores son listas de peers. Por tanto, lo que devuelve una búsqueda en Mainline DHT es una lista de peers (dirección IP y puerto TCP de cada peer). Con esta lista, es posible unirse al intercambio del objeto iniciando comunicación con los peers mediante el protocolo BitTorrent. El mensaje de recuperación de una lista de peers se llama get_peers.

Esta decisión implica que un nodo le envía un mensaje a otro para ser incluido en la lista de peers cuando se une al intercambio del objeto. En este caso, se dice que un nodo se anuncia y el mensaje se llama announcement o announce_peer. Los peers de la lista de peers también expiran. Por tanto, aunque las listas de peers son siendo valores, son dinámicas, dado que crecen y decrecen con los announcements y sus expiraciones. El Apéndice P ilustra qué implicaciones supone esta decisión frente al diseño de Kademlia.

2.5.3 Mensajes

Al igual que Kademlia, Mainline DHT tiene cuatro tipos de mensaje (el formato del paquete para estos mensajes se explica en el Apéndice C y se define en [20]):

- PING: comprueba que un nodo está vivo.
- FIND_NODE: dado un info hash, devuelve la dirección IP, puerto UDP e identificador de nodo de los k (valor fijado a 8 en las especificaciones) nodos más cercanos al info hash.
- ANNOUNCE_PEER: manda a un nodo que le añada en la lista de peers de un info hash.
- GET_PEERS: se igual que FIND_NODE con una excepción, si se encuentra un nodo que había recibido previamente un mensaje ANNOUNCE_PEER para el info hash, devuelve la lista de peers.

Capítulo 3

Análisis

Este capítulo presenta el análisis teórico que comprueban los experimentos.

3.1 Período de los announcements

El tiempo de expiración de un announcement debe ser el mismo valor que su período para que cuando este expire se envíe de nuevo. En las especificaciones de Mainline DHT este valor no está explícitamente definido, pero [7] indica que es de 30 minutos (aunque el diseño de Kademlia [2] lo fija en una hora, también dice que se puede modificar para optimizaciones).

Dado que Crosby y Wallach [7] afirman explícitamente que este valor es de 30 minutos, siendo su documento el más reciente, y que las especificaciones de Mainline DHT [20] no determinan ningún valor para este parámetro, se asume que **el tiempo de expiración es de 30 minutos**.

3.2 Número de nodos que contienen lista de peers

Según Crosby y Wallach [7], un peer se anuncia a los tres nodos más cercanos al info hash del objeto. No obstante, el diseño de Kademlia [2] afirma que un nodo almacena un valor en los k nodos más cercanos a la clave. Las especificaciones de Mainline DHT [20] fijan el valor de k a 8 y no indican nada más acerca de este parámetro. Ambas definiciones son inconsistentes pero coinciden en que los nodos elegidos para enviar un announcement deben ser los más cercanos al info hash.

Al ser el documento más reciente, se ha considerado la **elección de los tres nodos más cercanos al info hash para los announcements** que afirman Crosby y Wallach [7]. Esto indica además que el número de nodos que contiene la lista de peers será aproximadamente de tres (dado que son los elegidos para almacenarla).

Se podrían encontrar casos en los que la lista de peers estuviera contenida en más o menos de tres nodos dada la adición y exclusión de nodos en el DHT y la expiración y periodicidad de los announcements. En cualquier caso, la media en el número de nodos conteniendo lista de peers de un info hash debería ser cercana a

tres y la desviación típica un valor suficientemente pequeño como para considerar la media significativa.

3.3 Posicionar un nodo en Mainline DHT para que contenga una lista de peers dada

Las especificaciones de Mainline DHT [20] explican que los nodos deben elegir un identificador aleatorio de 160 bits cuando se unen al DHT. El identificador de un nodo posiciona a este en una parte dada del DHT. Esta posición es relativa ya que, dependiendo de su identificador, un nodo estará más cerca o más lejos del resto de nodos e info hashes. El diseño de Kademlia [2] explica que, dado que los identificadores de nodo son aleatorios, es bastante improbable que su distribución sea no uniforme.

Los info hashes también deberían seguir una distribución uniforme. Estos se calculan como la función hash SHA-1 del contenido del objeto. Por definición, una función hash debe asegurar una distribución uniforme en sus valores.

Dado que Mainline DHT es una red P2P abierta, cualquier cliente se puede unir. Así pues, se puede crear un cliente en el que la elección del identificador de nodo no sea aleatoria. Por otro lado, los info hashes de los objetos son conocidos, ya que están contenidos en los ficheros torrent y son inmutables. Por lo tanto, un nodo es capaz de elegir su identificador para situarse cerca de un info hash dado. Además, al ser aleatorios los identificadores de nodo, no es posible comprobar si un nodo ha elegido su identificador de forma aleatoria o no.

A través de la búsqueda de un info hash se puede conocer el conjunto de nodos que contienen la lista de peers de un objeto. Estos nodos deberían ser los más cercanos al info hash dado que son los que eligen los anunciantes (tal y como se define en [2]). En la respuesta a la búsqueda, los nodos que contienen la lista de peers no sólo proporcionan dicha lista, sino también información acerca de ellos mismos, incluyendo su identificador (para ver el formato de todos los mensajes y respuestas en Mainline DHT, consultar el Apéndice C).

Sabiendo la distancia entre un info hash y sus nodos más cercanos, y siendo capaz un nodo de elegir su identificador, este se puede posicionar en el DHT más cerca del info hash que aquellos que contienen la lista de peers. El nodo empezará a recibir announcements y a contener la lista de peers del objeto. A su vez, el nodo más cercano al info hash de los que contenían la lista de peers dejará de recibir announcements y, por tanto, su lista de peers empezará a decrecer a medida que los announcements vayan expirando (hasta que quede vacía). Así pues, debería ser posible añadir un nodo al DHT y hacer que contenga la lista de peers de cualquier objeto.

3.4 Control total de una lista de peers

De acuerdo con las referencias presentadas en este capítulo, el número de nodos conteniendo la lista de peers de un info hash debería ser alrededor de tres y, tanto el período como el tiempo de expiración de los announcements, debería ser de 30 minutos. Además, según se ha analizado, un nodo es capaz de elegir su identificador para posicionarse en cualquier punto del DHT.

Teniendo en cuenta el análisis presentado en las anteriores secciones de este capítulo, debería ser posible controlar la totalidad de una lista de peers llevando a cabo las siguientes acciones:

1. Obtener el info hash del fichero torrent del objeto.
2. Realizar una búsqueda y encontrar los nodos responsables de la lista de peers (deberían ser alrededor de tres).
3. Observar su distancia al info hash y posicionar tres nuevos nodos en el DHT más cercanos al info hash que el más cercano de los que contienen la lista de peers (la situación quedaría como refleja la Figura 3.1, en la que las formas cuadradas representan los nuevos nodos y las formas circulares los nodos antiguos.).
4. Esperar a que pase el tiempo de expiración (que debería ser 30 minutos).

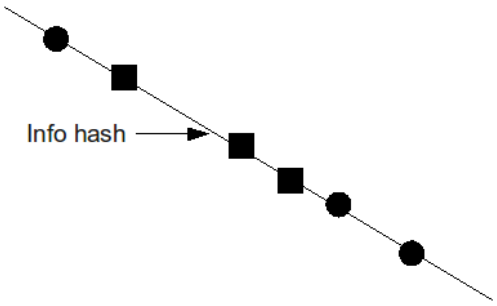

Figura 3.1. Nodos que contienen la lista de peers tras la adición de los nuevos nodos.

Tras seguir estos pasos, los peers del objeto no se anunciarán más en los nodos en los que lo estaban haciendo sino en los nuevos nodos (dado que en ese momento serían los tres nodos más cercanos al info hash). Así pues, **tres nodos posicionados como los más cercanos a un info hash serán la única entidad responsable del almacenamiento de la lista de peers pasado el tiempo de expiración**. El espacio entre un info hash y sus nodos más cercanos debería ser

suficientemente grande como para añadir en él millones de nodos (como se muestra en el Apéndice R).

3.4.1 Consecuencias

De acuerdo con el análisis presentado hasta ahora en este capítulo, un conjunto de nodos se puede coordinar para tomar el control de una lista de peers. Esto da la posibilidad a esos nodos de hacer un uso inadecuado de la lista de peers, lo que puede tener consecuencias perjudiciales para el DHT.

3.4.1.1 Responsabilidades

Un conjunto de nodos controlando la totalidad de la lista de peers de un objeto sería la única entidad responsable de permitir a otros nodos el acceso al intercambio del objeto. Cuando un nodo quiera obtener peers para intercambiar un objeto, realizará una búsqueda en el DHT. La búsqueda siempre desembocará en los nodos que contienen la lista de peers dado que todas las búsquedas sobre la misma clave convergen a lo largo del mismo camino (como se demuestra en [2]). El nodo que lleva a cabo la búsqueda enviará una petición de peers a los nodos que contienen la lista de peers.

3.4.1.2 Censura de contenidos

Una vez que los nodos que controlan una lista de peers empiecen a recibir peticiones de peers, pueden acordar proporcionar listas de peers vacías o falsas a los nodos que ellos quieran. En ese caso, se les estaría denegando el acceso al objeto haciéndoles imposible unirse al intercambio del objeto. Esta acción se consideraría censura de contenidos dada la denegación de acceso al objeto.

Es importante aclarar que siempre puede existir un tracker que contenga peers del objeto, en cuyo caso, los nodos que quieran descargarlo podrán obtener peers a través del tracker.

3.4.1.3 Ataques DDoS

Existe otra posible vulnerabilidad cuyo escenario es el mismo que en el problema de la censura aunque, a diferencia de este, en este caso no hace falta poseer el control total de una lista de peers, basta con uno o más nodos conteniendo parte de la misma. Este uso inapropiado consiste, de nuevo, en devolver una lista falsa de peers a los nodos que la pidan. Si esta lista de peers contiene la dirección IP de una máquina a la que atacar y el objeto asociado a la lista de peers es popular (es decir, tiene un gran número de peers), se estaría llevando a cabo un ataque DDoS silencioso (ya que el culpable no tomaría parte en el ataque, dado que él no enviaría ningún mensaje de forma directa a la víctima).

Los nodos que solicitan peers no son capaces de saber a priori si la dirección IP y el puerto proporcionados en la lista de peers corresponden a un cliente BitTorrent

hasta que intentan comunicarse con él. Si el info hash asociado a la lista de peers es muy popular, la víctima podría llegar a recibir cientos de miles de mensajes.

3.5 Escalabilidad en los nodos que contienen la lista de peers

Dada la aleatoriedad de los identificadores de nodo y la distribución uniforme de los info hashes, un nodo no puede saber a priori qué listas de peers va a contener. Si un nodo es el encargado de almacenar la lista de peers de un objeto muy popular, las tasas de tráfico que se generen en él pueden ser muy elevadas.

3.5.1 Análisis

El objetivo en este escenario es estudiar el crecimiento del tráfico entrante y saliente generado en el nodo que contiene la lista de peers de un objeto en función del número de peers. En el DHT hay millones de nodos. Por tanto, nada imposibilita la existencia de un objeto con cientos de miles de peers.

En los nodos que contienen la lista de peers de un objeto popular, el número de announcements y peticiones de peers por segundo puede ser excesivo, haciendo imposible que los nodos atiendan todos los mensajes. El tráfico entrante para estos nodos puede llegar a ser enorme, pero el tráfico saliente sería incluso mayor (porque las respuestas a los mensajes find_nodes y get_peers son mayores que las peticiones en sí ya que tienen que adjuntar una lista de nodos o una lista de peers tal como se muestra en [20]).

3.5.2 Consecuencias

El desequilibrio generado por este problema tendría tres implicaciones importantes:

3.5.2.1 Equidad

Aunque los nodos fueran capaces de soportar la tasa de tráfico generada por contener la lista de peers de un objeto popular, esta situación rompería con la propiedad de equidad en las redes P2P. Los nodos cercanos a un info hash de un objeto popular sufrirían una tasa de tráfico debida al DHT mucho mayor que el resto. El DHT es un esfuerzo para distribuir la indexación de las listas de peers a través de la colaboración equitativa de todos los nodos. Si algunos nodos sufren tasas de tráfico mucho mayores que el resto, el esfuerzo no estará repartido de forma equitativa entre todos los nodos.

3.5.2.2 Impacto

Este problema puede ser serio en conexiones de Internet domésticas, pero hay algunos casos donde puede ser más crítico. En la actualidad, existen algunos estudios

de clientes DHT en dispositivos móviles. Estos estudios tratan de minimizar el consumo de energía a través de una tasa de tráfico reducida (como el trabajo presentado en [31]). Estos dispositivos podrían no ser capaces de afrontar este problema. Por otro lado, el coste de algunas conexiones a Internet está basado en el tráfico generado. Los usuarios podrían no darse cuenta de que su cliente de BitTorrent está generando una tasa de tráfico enorme debida al DHT ya que no es algo de lo que un usuario final deba percatarse. Esto podría generar un coste elevado en la factura de Internet de los usuarios.

3.5.2.3 Futuro

Los estudios llevados a cabo en este proyecto muestran que el problema de escalabilidad no es demasiado crítico en los info hashes encontrados (de hasta 33000 peers). Sin embargo, en el DHT hay millones de nodos, por tanto, nada imposibilita la existencia de un objeto con millones de peers. Por ejemplo, el proyecto P2P-Next [34] es un proyecto a nivel europeo financiado, entre otros, por la cadena de televisión BBC, que pretende usar una red P2P para retransmitir eventos televisivos en directo. Eventos internacionales como Eurovision o los Juegos Olímpicos, que son seguidos por millones de personas, podrían dar lugar a info hashes con millones de peers.

Capítulo 4

Herramientas, experimentos y resultados

Este capítulo resume y analiza los resultados de los experimentos realizados. La definición y resultados completos de los experimentos están contenidos, desde el Apéndice G al Apéndice O, progresivamente. Además, en ellos, se puede observar que algunos utilizan un conjunto grande de info hashes. La obtención de estos info hashes y su caracterización se explica en el Apéndice D. Otros experimentos han utilizado varios nodos simultáneamente. La coordinación de estos nodos se explica en el Apéndice E.

4.1 Herramientas

En este proyecto se presenta un análisis que, posteriormente, se comprueba empíricamente. Para llevar a cabo este fin, fue necesario desarrollar una colección de herramientas. Estas herramientas son pequeños scripts con propósitos muy específicos, desarrollados modificando un cliente de Mainline DHT. En el Apéndice Q se explica brevemente el propósito de las mismas. Aunque las herramientas son una contribución importante de este proyecto, a través de los resultados de los experimentos se puede ver más claramente su funcionalidad. El Apéndice Q justifica también las elecciones, en cuanto a desarrollo, tomadas para la elaboración de las herramientas.

4.2 Configuración de los experimentos

Se han llevado a cabo 9 experimentos cuyos resultados se muestran desde el Apéndice G al Apéndice O y están resumidos en este capítulo. Para la configuración de los experimentos, se tomaron varias decisiones. Están explicadas y justificadas en el Apéndice T que explica la metodología seguida en todo el proceso.

Ente la información contenida en en Apéndice T, que explica la metodología seguida, hay un hecho que cabe destacar para entender los resultados de alguno de

los experimentos. El hecho es que durante el desarrollo del proyecto se observó que las listas de peers con bastantes peers no caben en el campo de datos de un paquete UDP (que es el protocolo utilizado para el envío de mensajes en Mainline DHT). Se encontraron dos soluciones: fragmentar los paquetes IP o limitar el tamaño de la lista de peers. El cliente DHT utilizado fue modificado para que limitara la lista de peers a los últimos 50 announcements.

4.3 Estudio del DHT

El experimento 1 estudia el número de nodos que contienen lista de peers utilizando 1200 info hashes. Según el estudio presentado en el Capítulo 3, este valor debería ser cercano a 3. Los resultados muestran que, de los info hashes observados, el 40% tiene más de 10 nodos conteniendo su lista de peers. Este comportamiento lo explican los resultados del experimento 5.

El experimento 2 comprueba si el número de announcements recibidos en los nodos más cercanos a un info hash depende de la distancia a este. Como se esperaba, se ha comprobado que la distancia no influye en el número de announcements. Sin embargo, los tres nodos más cercanos al info hash deberían ser los únicos que reciben announcements y todos los nodos están recibiendo un número elevado de announcements. Este comportamiento lo explican los resultados del experimento 5.

El experimento 3 estudia el período de los announcements. Los resultados son como se esperaba, ya que la mayoría de los nodos se anuncian periódicamente en un tiempo cercano a 30 minutos. Sin embargo, hay una excepción, ya que el 17% de los nodos anunciantes observados tiene un período inferior a 5 minutos. Este resultado es inesperado y debería ser investigado más en profundidad.

El experimento 5 da con la razón de algunos comportamientos inesperados observados en otros experimentos. Muestra que algunos clientes del DHT no están siguiendo las especificaciones de los announcements. Los clientes BitSpirit se están anunciando en más de tres nodos en todos los casos. Los clientes KTorrent están escogiendo para los announcements nodos que es casi imposible que sean los más cercanos al info hash. La agregación de estos comportamientos explica por qué en el experimento 1 el número de nodos que contienen lista de peers es tan elevado y por qué en el experimento 2 más de tres nodos reciben announcements.

4.4 Censura

El experimento 4 es un intento de censura. Los resultados obtenidos en los experimentos 1, 2 y 3 ayudaron a diseñarlo. De acuerdo con el experimento 1, un conjunto de 12 nodos debería ser suficiente para llevar a cabo la censura. De acuerdo con el experimento 2, su distancia al info hash no importa (mientras que sean los más cercanos). De acuerdo con el experimento 3 en, aproximadamente una hora, deberían controlar toda la lista de peers.

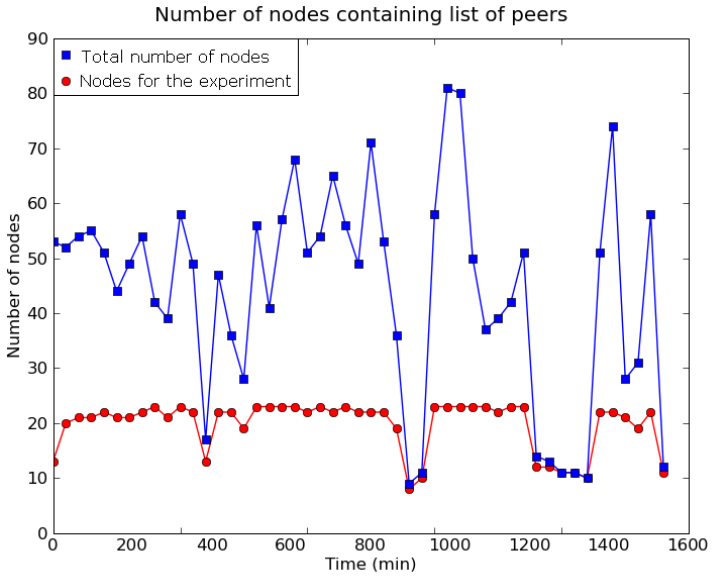

Figura 4.1. Número de nodos conteniendo la lista de peers.

La censura no ha sido posible en ningún caso. Aunque en un intento hubo un pequeño período de tiempo en que lo fue, dado que no había más nodos que los del experimento conteniendo la lista de peers (como se puede observar en la Figura 4.1, extraída del Apéndice J), aun siendo los nodos del experimento los más cercanos al info hash y manteniéndose vivos todo el tiempo, siempre se unen nuevos nodos conteniendo la lista de peers. La razón de este resultado (que fue investigada a posteriori) son, de nuevo, los resultados obtenidos en el experimento 5. La existencia de clientes que se anuncian en más de tres nodos posibilitaría la censura. Sin embargo, al haber clientes que no se anuncian sólo en los nodos más cercanos al info hash, es imposible llevar a cabo la censura añadiendo nodos cerca del info hash. Dado que no todos los clientes se comportan igual, la heterogeneidad de clientes en el DHT y sus comportamientos hacen que la censura sea casi imposible.

4.5 Influencia en el DHT de los nodos que no siguen las especificaciones

Dado que en el experimento 5 se descubrió que algunos nodos no siguen las especificaciones del diseño, los experimentos 6 y 7 estudian el impacto en el DHT de estos nodos.

El experimento 6 muestra que hay nodos conteniendo listas de peers que es casi imposible que sean parte de los más cercanos al info hash. La distribución de las distancias de estos nodos al info hash parece ser una distribución normal con la

excepción de que los nodos más lejanos son demasiados. El alto número de nodos lejanos se debe, probablemente, a los resultados obtenidos en el experimento 5.

El experimento 7 analiza la distribución de la lista de peers entre los nodos que la contienen. El resultado muestra que los tres nodos más cercanos al info hash son aquellos que contienen la mayor parte de la lista de peers. Sin embargo, hay nodos conteniendo la lista de peers que no son parte de los tres más cercanos. El porcentaje de la lista de peers parece decrecer exponencialmente en función del orden de distancia al info hash. Una hipótesis acerca de la causa de este resultado es que algunos clientes se anuncian en los nodos que encuentran durante la búsqueda. En [2] se demuestra que, si el DHT tiene n nodos, el número de iteraciones para encontrar un valor es $O(log(n))$. Esto podría explicar el decrecimiento exponencial.

4.6 Escalabilidad y ataques DDoS

El experimento 8 comprueba empíricamente los problemas de escalabilidad y ataques DDoS. Muestra que, en info hashes populares, el impacto del tráfico entrante en el nodo más cercano al info hash no es demasiado crítico con los info hashes encontrados (de hasta 33.000 peers). No es una tasa de tráfico desmesurada para conexiones domésticas, pero sí que es demasiado alta para estar generada tan sólo por el DHT. El problema del tráfico saliente es más serio, ya que las tasas generadas no son razonables. Dado que la lista de peers puede llegar a ser enorme en este tipo de info hashes, la tasa de tráfico saliente llega a ser enorme.

Para reducir el impacto de este problema se ha propuesto una solución sencilla, consiste en limitar el tamaño de la lista de peers en las respuestas a las peticiones de peers. Como se ha anteriormente en ese capítulo, listas con más de 50 peers no encajan en el campo de datos de un paquete UDP. La opción de limitar la lista a 50 peers suaviza el problema de escalabilidad. A través de esta solución se ha podido soportar el tráfico del info hash más popular encontrado (aunque las tasas de tráfico todavía siguen siendo demasiado altas). Esta solución reduce el impacto del problema pero este todavía existe.

En el futuro pueden existir info hashes con millones de peers. Proyectos como P2P-Next [34] (cuyo objetivo es emitir transmisiones televisivas en directo usando una red P2P) pueden dar lugar a info hashes con millones de peers en la transmisión de eventos internacionales. Además, BitTorrent tiene hoy en día millones de usuarios, nada impide que pueda existir un objeto con un millón de peers. El tráfico en este tipo de nodos haría el problema mucho más serio. En los resultados del experimento 8 se proveen datos para estimar este posible tráfico.

4.7 Detección de nodos sospechosos

Ya que la distribución obtenida en el experimento 6 no se ajusta a ninguna conocida debido a los nodos que no siguen las especificaciones, en el experimento 9 se ha estudiado la distancia entre el info hash y su nodo más cercano. Esta medición no puede

ser alterada por los distintos clientes, ya que el nodo más cercano a un info hash siempre debería contener la lista de peers del objeto. Los datos obtenidos muestran una distribución que parece ser normal en estas distancias. El conocimiento de esta distribución puede ayudar a comprobar si el conjunto de los nodos más cercanos a un info hash concuerda con la distribución y detectar así nodos sospechosos.

Capítulo 5

Trabajo futuro

El trabajo de este proyecto estudia una parte poco explorada de Mainline DHT. En este capítulo se presentan todos los hechos que, a partir del desarrollo de este proyecto, se consideran interesantes para estudiar y que han estado fuera del alcance del trabajo desarrollado.

5.1 Estudio de comportamientos anómalos

Uno de los hechos más importantes descubiertos durante este proyecto ha sido que algunos clientes de BitTorrent no respetan las especificaciones del DHT (como muestra, por ejemplo, el experimento 5). La coexistencia de varias implementaciones diferentes de clientes DHT y sus respectivos comportamientos hacen que algunas premisas deducidas a partir del diseño del DHT sean incorrectas, influyendo así en su comportamiento real.

Algunos de estos comportamientos son beneficiosos (por ejemplo, el experimento 5 muestra que, dado que hay clientes que no eligen sólo los nodos más cercanos al info hash para anunciarse, es casi imposible censurar el acceso a la lista de peers). Por el contrario, estos comportamientos generan tráfico extra. Se debería comprobar si existen razones por las que estos comportamientos sean razonables y estudiar más a fondo su influencia en el DHT. Los comportamientos cuyo estudio se considera más interesante son los siguientes:

- El número de nodos que un nodo elige para anunciarse no es el mismo en todos los clientes de BitTorrent (como muestra el experimento 5).

- Algunos clientes de BitTorrent eligen nodos que no son parte de los más cercanos al info hash para anunciarse (como muestra el experimento 5).

- Hay clientes que, a la hora de anunciarse, envían varios announcements en unos pocos segundos cuando sólo uno es necesario (como muestra el experimento 3).

- Algunos clientes proveyendo listas de peers proporcionan listas con peers repetidos (como se ha observado en varios experimentos).

5.2 Gestión de la lista de peers

Un hecho cuya importancia destaca en este proyecto es la gestión de la lista de peers. Al principio se descubrió que las listas de peers con más de 50 peers no caben dentro del campo de datos de un paquete UDP. Existen dos alternativas para afrontar este problema: limitar el tamaño de la lista de peers o fragmentar los paquetes IP para que el destinatario los ensamble. El experimento 8 muestra que el problema de escalabilidad se reduce notablemente utilizando una lista de peers limitada. En el caso de este proyecto la lista de peers se ha limitado a los 50 announcements más recientes. Sin embargo, distintas políticas de gestión de lista de peers deberían ser analizadas. Dicho estudio debería tener en cuenta el intercambio de peers que existe dentro del protocolo BitTorrent por el que los peers pueden comunicarse entre ellos para obtener más peers (como se detalla en [30]).

5.3 Estudio más exhaustivo de Mainline DHT

En este proyecto se ha estudiado una parte específica del DHT y, con ello, se han descubierto varias situaciones que sería interesante estudiar. Antes del comienzo de este proyecto no se contaba con demasiada información acerca del comportamiento real del DHT. A través de la observación de sus comportamientos es más fácil entender lo que ocurre dentro de él.

El hecho de que haya clientes que no siguen las especificaciones, implica que el modelo teórico del DHT no concuerde con el comportamiento real. En estudios más exhaustivos, probablemente se encontrarían más comportamientos anómalos. Sería muy útil obtener una clasificación de clientes y sus respectivos comportamientos. Se proponen dos modos de estudio del DHT: observaciones de su estado real y observaciones del comportamiento individual de los clientes.

5.4 Prevenir y solucionar las vulnerabilidades

Aunque el experimento 8 muestra que el problema de escalabilidad es reducible, este todavía existe. Objetos con millones de peers podrían implicar que los nodos que contuvieran su lista de peers no pudieran soportar el tráfico generado. En los resultados del experimento 8 se proporcionan datos para estimar el tráfico generado por objetos con un número de peers no existente en la actualidad.

5.4.1 Solucionar el problema de escalabilidad

Una posible solución al problema de escalabilidad radica en el hecho de hacer dinámico el número de nodos conteniendo la lista de peers en función del número de

peers. Este mecanismo debería asegurar que todos los nodos del DHT soportaran una cantidad similar de tráfico generado por el DHT.

Una solución completa se podría alcanzar desde aquellos nodos participando en la búsqueda del info hash. Podrían detectar que demasiados nodos están buscando un info hash y limitar de algún modo el acceso a estos.

Otra técnica que podría ser útil a la hora de diseñar una solución es el caching de valores en el DHT. En el diseño de Kademlia [2] hay una explicación de cómo puede llevarse a cabo esto. Se explica que, dado que todas las búsquedas convergen a lo largo de un mismo camino, el caching de valores aliviaría el trabajo de los nodos que los contienen. Esto no se hace actualmente en Mainline DHT. Sería sencillo de implementar si en él los valores fueran estáticos, pero son dinámicos así que la implementación de esta solución no es trivial dada la posible incoherencia de valores.

5.4.2 Prevenir ataques DDoS

En el experimento 8 se ha estudiado también el problema de los ataques DDoS. Esta vulnerabilidad está presente y tendría un impacto en la víctima similar al causado por el tráfico entrante en un nodo que sufre el problema de escalabilidad.

Para prevenir este problema, los experimentos 6 (Apéndice L) y 9 (Apéndice O) ofrecen datos para detectar nodos posicionados sospechosamente cerca de un info hash. Aún así, los nodos atacantes podrían colocarse a una distancia prudencial del info hash. Por tanto, se debería buscar una solución completa. Si dicha solución limitara el tráfico que se puede encaminar como ataque DDoS, valdría también como solución para el problema de escalabilidad.

5.5 Modificaciones a largo plazo

Algunas de las vulnerabilidades estudiadas en este proyecto se deben a la posibilidad que tienen los nodos de elegir cualquier identificador. La existencia de un mecanismo para detectar nodos ilegítimos resolvería algunos problemas.

Una posible solución para resolver este problema sería obligar a los nodos a calcular su identificador como una función hash de su dirección IP (como se hace en Azureus DHT [24]). Esto permitiría comprobar si un nodo ha elegido su identificador de forma legítima. Los nodos que no lo hagan podrían ser ignorados.

El problema de esta solución es que es global y todos los nodos deberían aplicarla a partir de un momento dado, lo cual es difícil en un DHT con millones de nodos. Para aliviar este problema podría establecerse un período de transición usando al principio técnicas menos agresivas para ignorar a los nodos.

Capítulo 6

Conclusiones

Este capítulo presenta las conclusiones extraídas de los resultados obtenidos en los experimentos y, en general, desde la experiencia del desarrollo de este proyecto. Compara los resultados con los objetivos propuestos en el Capítulo 1 y explica la contribución del trabajo.

6.1 Cumplimiento de los objetivos y contribución

En el Capítulo 3 se presentó un estudio acerca de la generación, distribución y obtención de las listas de peers en el DHT. Los experimentos presentados desde el Apéndice G al Apéndice O y resumidos en el Capítulo 4 exploran esta parte del DHT y comparan los resultados con el análisis. La combinación de todos los experimentos presenta un perfil de esta parte del DHT. Este perfil ayuda a entender mejor cómo funciona el DHT.

Se ha desarrollado un conjunto de herramientas para interaccionar con el DHT. Se presentan en el Apéndice Q. Dichas herramientas han servido para comprobar empíricamente el análisis presentado en el Capítulo 3. Estas herramientas se pueden utilizar o adaptar para recopilar información del DHT. Su uso puede ser útil para futuras investigaciones como las propuestas en el Capítulo 5. Las herramientas son Open Source de modo que puedan ser utilizadas y modificadas para futuras investigaciones.

El experimento 4 muestra que la censura no ha sido posible en ningún caso. Aunque el análisis indicaba que era posible, el experimento 5 muestra que existe un mecanismo implícito que la evita, basado en que hay clientes BitTorrent que no se anuncian sólo en los nodos más cercanos al info hash.

El experimento 8 documenta que algunos nodos del DHT pueden sufrir un tasa de tráfico mucho más alta que el resto, debido a la popularidad de algunos objetos. Aunque este tráfico puede no ser crítico para algunos usuarios hoy en día, detectar que algunos nodos deben soportar una tasa de tráfico mayor que el resto es una contribución importante, ya que rompe la propiedad de equidad en el DHT.

Se han encontrado algunos clientes que no respetan las especificaciones, como

muestran los experimentos 3 (que muestra que hay clientes que no respetan el período de los announcements) y 5 (que muestra que hay clientes que no se anuncian en los tres nodos más cercanos al info hash). Estas inconsistencias suponen una carencia de equidad en el DHT ya que no todos los clientes lo están utilizando de la misma manera.

6.2 Lecciones aprendidas

Estas son algunas de las lecciones aprendidas desde la experiencia del desarrollo de este proyecto:

- **El correcto funcionamiento de Mainline DHT se basa en la igual cooperación de todos los participantes**. Las vulnerabilidades y el perfil de Mainline DHT estudiados en este proyecto han ayudado a detectar algunas situaciones de falta de equidad. La documentación y solución de estas situaciones puede suponer una mejora en el rendimiento y seguridad en el software de millones de usuarios.

- **Proponer cambios globales no es fácil en un DHT como Mainline DHT**. Los cambios globales han de ser adoptados en todo el DHT al mismo tiempo, lo cual hace difícil su implantación. Sin embargo, un período de transición en el que se utilice una solución intermedia es razonable.

- **La influencia de UTorrent en Mainline DHT es muy importante**. El experimento 2 parece indicar que UTorrent es el cliente más popular en Mainline DHT. Probablemente, los cambios globales deberían empezar por UTorrent porque es una parte muy importante de Mainline DHT.

- **Algunos de los problemas estudiados en este proyecto existen debido a la posibilidad de que los nodos elijan un identificador dado**. En el futuro, debería existir un mecanismo para evitar que los nodos puedan escoger cualquier identificador.

Los resultados de este proyecto han sido de ayuda para el grupo de investigación en el que se ha trabajado de cara al diseño de una red P2P en la que los objetos pueden llegar a tener millones de peers. El perfil obtenido de Mainline DHT, el estudio de la censura y las pruebas de escalabilidad proporcionan datos para mejorar el rendimiento y evitar problemas futuros.

En cuando a las impresiones personales del proyecto, entender el funcionamiento del protocolo del DHT a la perfección fue una de las partes más difíciles, ya que cada paso aportaba nuevos conocimientos. Además, los experimentos se han obtenido interactuando con un sistema real en el que hay millones de nodos activos, por lo que había que asegurar el correcto funcionamiento de las herramientas ya que no era fácil diferenciar casos anómalos de bugs.

Apéndice A

Glosario de términos

- DDoS attack: a denial of service attack (DoS attack) or distributed denial of service attack (DDoS attack) is an attempt to make a computer resource unavailable to its intended users.
- DHT: a structured overlay that uses key-based routing for put and get index operations and in which each peer is assigned to maintain a portion of the index.
- Hash: a hash function is any well-defined procedure or mathematical function that converts a large, possibly variable-sized amount of data into a small datum.
- Info hash: 160-bit SHA-1 hash of an object.
- IP address: an Internet Protocol (IP) address is a numerical label that is assigned to devices participating in a computer network that uses the Internet Protocol for communication between its nodes.
- Key: sequence of bits used to index a value, usually much smaller than the value.
- Leecher: a peer or any client that does not have 100% of the object.
- Node: computer connected to the Internet running a BitTorrent client using the DHT.
- Object: chunk of data.
- P2P: Peer-to-Peer systems are distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server or authority.

- Peer: an end system, node, or host that is a member of a P2P system.
- Seeder: a seeder is a peer that has a complete copy of the torrent and still offers it for upload.
- SHA-1: is a cryptographic hash function designed by the National Security Agency (NSA) and published by the NIST as a U.S. Federal Information Processing Standard. SHA stands for Secure Hash Algorithm.
- Swarm: together, all peers (including seeders) sharing a torrent. For example, six ordinary peers and two seeders make a swarm of eight.
- Torrent: a small meta-data file which contains information about the object to download, not the object itself. It is downloaded from a web site (BitTorrent file extension is .torrent). Part of the meta-data is the info hash of the object.
- Tracker: server on the Internet that coordinates the action of BitTorrent Clients. Upon opening a torrent, the tracker is contacted and a list of peers to connect to is received. Throughout the transfer, the client will query the tracker, telling it how much it has downloaded and uploaded and how much before finishing.
- Value: chunk of data indexed by a key.

Apéndice B

Acrónimos

- DDoS: Distributed Denial of Service.
- DHT: Distributed Hash Table.
- IP: Internet Protocol.
- P2P: Peer-to-Peer.
- SHA: Secure Hash Algorithm.
- TCP: Transmission Control Protocol.
- UDP: User Datagram Protocol.
- URL: Uniform Resource Locator.

Apéndice C

Formato de paquetes en Mainline DHT

The packet format defined here is extracted from [20].

PING query:

```
"t": token
"y": "q"
"q": "ping"
"a": "id": sender_id
```

"t": parameter is a token.
token: transaction ID.
"y": the query contains two additional keys: "q" and "a".
"q": the message is a query.
"ping": the query is a ping query.
"a": there are arguments.
"id": parameter is an identifier.
sender_id: node identifier of the querier.

PING response:

```
"t": token
"y": "r"
"r": "id": responder_id
```

"t": parameter is a token.
token: transaction ID (must be the same sent in the query).
"y": the response contains an additional key: "r".
"r": the message is a response.
"r": there are returned values.
"id": parameter is an identifier.

responder_id: node identifier of the responder.

FIND_NODE query:

```
"t": token
"y": "q"
"q": "find_node"
"a": "id": sender_id
     "target": target_node
```

"t": parameter is a token.
token: transaction ID.
"y": the query contains two additional keys: "q" and "a".
"q": message is a query.
"find_node": the query is a find_node query.
"a": there are arguments.
"id": parameter is an identifier.
sender_id: node identifier of the querier.
"target": parameter is the node to find.
target: identifier of the node to find.

FIND_NODE response:

```
"t": token
"y": "r"
"r": "id": responder_id
     "nodes": info_node_1
              info_node_2
              ...
              info_node_8
```

"t": parameter is a token.
token: transaction ID (must be the same sent in the query).
"y": the response contains an additional key: "r".
"r": the message is a response.
"r": returned values.
"id": parameter is an identifier.
responder_id: node identifier of the responder.
"nodes": parameter is a list of nodes.
info_node_i: information about node i (identifier, IP address and UDP port) compacted.

GET_PEERS query:

```
"t": token
"y": "q"
"q": "get_peers"
"a": "id": sender_ID
     "info_hash": target_info_hash
```

"t": parameter is a token.
token: transaction ID .
"y": the query contains two additional keys: "q" and "a".
"q": message is a query.
"get_peers": the query is a get_peers query.
"a": there are arguments.
"id": parameter is an identifier.
sender_id: node identifier of the querier.
"info_hash": parameter is an info hash.
target_info_hash: info hash to get_peers.

GET_PEERS response if the queried node has no peers (in that case will provide closer nodes):

```
"t": token
"y": "r"
"r": "id": responder_id
     "token": token_ann
     "nodes": info_node_1
              info_node_2
              ...
              info_node_8
```

"t": parameter is a token.
token: transaction ID (must be the same sent in the query).
"y": the response contains an additional key: "r".
"r": the message is a response (must be the same sent in the query).
"r": there are returned values.
"id": parameter is an identifier.
responder_id: node identifier of the responder.
"token": parameter is a token to send in case the querier wants to announce.
token_ann: value of the token.
"nodes": parameter is a list of nodes.
info_node_i: information about node i (identifier, IP address and UDP port) compacted.

GET_PEERS response if the queried node has peers:

```
"t": token
"y": "r"
"r": "id": responder_id
     "token": token_ann
     "values": peer_1
               peer_2
               ...
               peer_n
```

"t": parameter is a token.
token: transaction ID (must be the same sent in the query).
"y": the message contains an additional key: "r".
"r": message is a response.
"r": there are returned values.
"id": parameter is an identifier.
responder_id: node identifier of the responder.
"token": parameter is a token to send in case the querier wants to announce.
token_ann: value of the token.
"values": parameter is a list of peers.
peer_i: information about peer i (IP address and TCP port) compacted.

ANNOUNCE_PEER query:

```
"t": token
"y": "q"
"q": "announce_peer"
"a": "id": sender_ID
     "info_hash": info_hash_ann
     "port": TCP_port
     "token": token_gp
```

"t": parameter is a token.
token: transaction ID.
"y": the message contains two additional keys: "q" and "a".
"q": message is a query.
"announce_peer": the query is an announce_peer query.
"a": there are arguments.
"id": parameter is an identifier.
sender_id: node identifier of the querier.

"info_hash": parameter is the info hash of the announcement.
info_hash_ann: value of the info hash.
"port": parameter is TCP port of the BitTorrent protocol.
TCP_port: value of the TCP port.
"token": parameter is the token provided in the get_peers query.
token_gp: value of the token.

ANNOUNCE_PEER response:

```
"t": token
"y": "r"
"r": "id": responder_id
```

"t": parameter is a token.
token: transaction ID (must be the same sent in the query).
"y": the message contains an additional key: "r".
"r": message is a response.
"r": there are returned values.
"id": parameter is an identifier.
responder_id: node identifier of the responder.

Apéndice D

Obtención de un número elevado de info hashes

Some of the experiments had to use a large set of info hashes in order to get reliable results, for this kind of experiments it's been used a set of 1200 info hashes obtained from The Pirate Bay [8]. The way to get them has been using the following Unix command:

```
wget http://thepiratebay.org/browse/100/ -r -x --no-parent
```

This command retrieves all the files contained in the folder browse/100 of the webpage. In the webpage, it's possible to see that all the torrents are contained in folders with the prefix /browse/100, /browse/200, etc. So, repeating this command for five or six times, a large set of html files containing info hashes is obtained.
Once all this html files have been downloaded, it's necessary to parse them and get the info hash from them (it was previously checked that it was contained there). In all of them it is part of the line 155 of the file. To get it, it's been used the following Unix command:

```
sed '155q;d;' file.html | sed -n 's/.*\([a-f0-9]\{40\}\).*/\1/p'
```

The first part (before the vertical bar) gets the line number 155 of the file and uses it as input for the second command. The second command parses that line searching a sequence of 40 hexadecimal digits and prints it.
With this, it's been developed a Python script to parse all the downloaded files and print all the info hashes in a single file.

A summary of the number of peers for this set of info hashes is summarized in the Table D.1. The number of peers is all those peers observed in all the lookups.

Number of peers	Number of info hashes
0	298
1-49	658
50-99	61
100-149	47
150-199	41
200-249	30
250-299	20
300-349	15
350-399	6
400-449	9
450-499	6
500-549	1
550-599	2
600-649	3
650-699	1
700+	2

Tabla D.1. Large set of info hashes and their number of peers.

Apéndice E

Experimentos con varios nodos

Some of the experiments require a set of nodes. In order to perform this experiments, some of the nodes of the Planetlab network have been used. Planetlab provides access to hundreds of computers all around the world and with different IP addresses. For some experiments up to 30 nodes have been used at the same time.
Nodes have been accessed using SSH [18]. It's been used one node to coordinate the others, by using SSH, this node sends commands included in the Unix SSH command to the other nodes.

Apéndice F

Probabilidades de identificadores

- XOR distance is the result of the bitwise XOR of two sequences. Log distance performs the same operation, however, the result is not the number itself but the position (counting from the right and starting from zero) of the first bit which is 1 (counting from the left).
- In an identifier space of a bits, log distances belong to the interval $[-1, a)$.
- Two sequences have a log distance of -1 between them in an identifier space of a bits if they are equal (i.e. they have a bits equal).
- Two sequences have a log distance of $a-1$ between them in an identifier space of a bits if their first left bit is different.
- Two sequences have a log distance of d between them where $0 \leq d < a-1$ in an identifier space of a bits if their first $a-d-1$ left bits are equal and the next one is different.
- The probability that two random sequences of n bits are equal is $\frac{1}{2^n}$.
- The probability of two random sequences to have a log distance equal to -1 between them in an identifier space of a bits is $\frac{1}{2^a}$.
- The probability of two random sequences to have a log distance equal to $a-1$ between them in an identifier space of a bits is $\frac{1}{2}$.
- The probability of two random sequences to have a log distance equal to d between them where $0 \leq d < a-1$ in an identifier space of a bits is the probability that their first $a-d-1$ left bits are equal and the next one is different.
- The probability of two random sequences to have a log distance equal to d between them where $0 \leq d < a-1$ in an identifier space of a bits is $\frac{1}{2^{a-d-1}} * \frac{1}{2} = \frac{1}{2^{a-d}}$

- The probability of two random sequences to have a log distance equal or less to d between them in an identifier space of a bits is the probability that they have the $a - d - 1$ or more left bits in common and one different.
- The probability of two random sequences to have a log distance equal or less to d between them where $0 \leq d < a - 1$ in an identifier space of a bits is $\frac{1}{2^a} + \sum_{i=0}^{d} \frac{1}{2^{a-i}} = 2^{d-a+1}$.

Apéndice G

Experimento 1 - Número de nodos conteniendo una lista de peers

The number of nodes where a node should announce itself is not clearly defined in the bibliography. This experiment aims to clarify it and document it. The number of nodes containing list of peers of an info hash should be similar to that value since the nodes chosen for the announcement have to be the closest to the info hash.

G.1 Expected results

According to [7], when a peer announces itself, it does it in the three closest nodes to the info hash of the object. However, according to the design of Kademlia [2], a node stores a value in the k closest nodes to the key and [20] specifies that in Mainline DHT k=8. In Mainline DHT values are the lists of peers, so, nodes should announce themselves in the 8 closest nodes to the info hash. As in the design of Mainline DHT the number of nodes where a node should announce itself isn't specified explicitly, for this experiment we suppose that the value is three nodes, as asserted in [7].

The number of nodes containing list of peers should be similar to the number of nodes chosen for the announcement since the closest nodes to the info hash are chosen for the announcement (both [7] and [2] assert it). Our deduction points out that **the number of nodes containing list of peers for an info hash is about three in all the cases**.

G.2 Experiment definition

An active node has been used for this experiment. For every info hash, the node has performed lookups over it and has counted the number of different nodes that return a list of peers. A node is identified by its IP address and UDP port.

- Number of active nodes: 1.

- Number of info hashes to lookup: 1200.
- Number of lookups over every info hash: 10 (1 every 5 seconds).
- Data to measure: number of nodes containing list of peers for every info hash.

The tools complete_lookup and display_average_number_nodes have been used for this experiment.

G.3 Results

The data obtained in the experiment is summarized in Table G.1 and represented graphically in Figure G.1. The result hasn't been as we expected. Even though the most typical case is 5 nodes which is close to 3 and 3 is the second most typical case, there are too many cases with too many nodes containing list of peers. The 40.23% of the observed info hashes have more than 10 nodes containing list of peers. The average number of nodes containing list of peers is 11.8 but the standard deviation is 18.47, which is too high to take into account the average. According to the expected results we show in Chapter 3, the average should be approximately 3 and the standard deviation a small number.

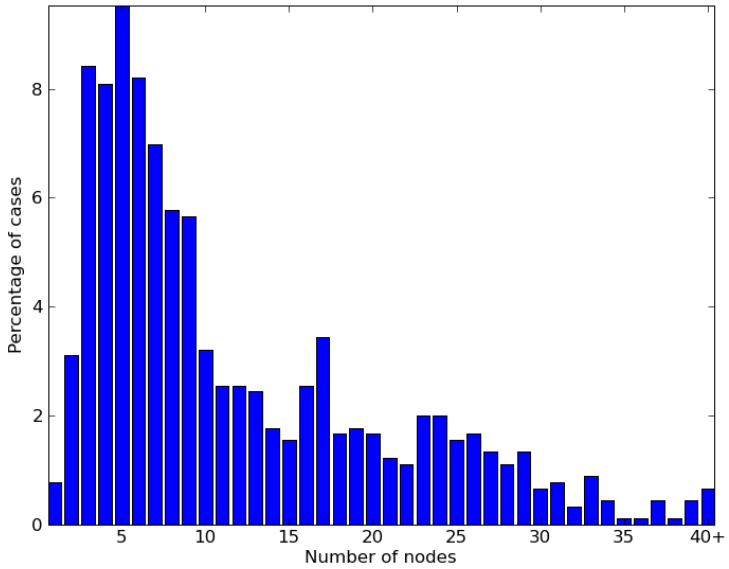

Figura G.1. Number of nodes containing list of peers.

Number of nodes	Ocurrences	Percentage
1	7	0.78%
2	28	3.10%
3	76	8.43%
4	73	8.09%
5	86	9.53%
6	74	8.20%
7	63	6.98%
8	52	5.76%
9	51	5.65%
10	29	3.22%
11	23	2.55%
12	23	2.55%
13	22	2.44%
14	16	1.77%
15	14	1.55%
16	23	2.55%
17	31	3.44%
18	15	1.66%
19	16	1.77%
20	15	1.66%
21	11	1.22%
22	10	1.11%
23	18	2.00%
24	18	2.00%
25	14	1.55%
26	15	1.66%
27	12	1.33%
28	10	1.11%
29	12	1.33%
30	6	0.67%
31	7	0.78%
32	3	0.33%
33	8	0.89%
34	4	0.44%
35	1	0.11%
36	1	0.11%
37	4	0.44%
38	1	0.11%
39	4	0.44%
40+	6	0.67%

Tabla G.1. Number of nodes containing list of peers.

Apéndice H

Experimento 2 - Crecimiento de una lista de peers en función de la distancia al info hash de los nodos que la contienen

This experiment intends to corroborate that nodes positioned as the closest to an info hash start having a list of peers of it and containing a big part of the list. This experiment checks whether the result changes depending on the distance or not.

H.1 Expected results

The closest nodes to the info hash will receive the largest amount of announcements independently of their distance to the info hash. According to [2] and [20], nodes announce in the closest nodes to the info hash. There's not any mention in the bibliography about any different behavior according to the distance of the nodes to the info hash.

H.2 Experiment definition

A set of nodes has been positioned close to an info hash in three different scenarios. In every scenario, one of the nodes was the closest and consecutive pairs of nodes had a distance between them of 2 units.

We didn't check our nodes were the closest all the time, however, according to the probabilities we present in Appendix F, the probability for a random node identifier to have distance 0 or -1 to the info hash is 2^{-159}, to have distance 22 or less to the info hash 2^{-137} and to have distance 80 or less to the info hash 2^{-79}. So, it's extremely improbable that a random node is closer to the info hash than our nodes in this experiment.

- Number of passive nodes: 11 in the first scenario, 29 in the second scenario and 31 in the third scenario.

- First scenario: the log distance between the info hash and its the closest node is 0.

- Second scenario: the log distance between the info hash and its the closest node is 22.

- Third scenario: the log distance between the info hash and its the closest node is 80.

- Log distance between consecutive pairs of nodes in every scenario: 2 (if the distance to the info hash of the closest node is x, the distance of the second closest is $x + 2$, the distance of the third closest $x + 3$, etc).

- Number of different info hashes tested: 1. This experiment requires too much time to be performed with a large set of info hashes. It has been chosen a popular info hash to have a large enough set of announcements.

- Time for the experiment: 24 hours in every scenario and another 24 hours between consecutive scenarios.

- Data to measure: number of announcements in every node as well as the versions of the clients announcing (when they provide it).

The tools announcements and display_announcements_distance have been used for this experiment.

H.3 Results

The results of this experiment are shown graphically the first scenario in Figure H.1, the second scenario in Figure H.2 and the third scenario in Figure H.3 (the Y axes in the figures has a logarithmic scale). They show the total number of announcements and the versions of the clients announcing.

In these experiments we use the abbreviated name of BitTorrent clients (for example, "UT" is UTorrent). "None" are those clients which don't provide their version (more information about the meaning of these abbreviated names can be found in [21]).

In the scenario 1 and the scenario 3 the closest node is receiving a very small amount of announcements. This behavior should be investigated more in depth. If the closest node is ignored, the closest nodes are those receiving the largest amount of announcements in all the scenarios. This result seems to be caused by UTorrent. The rest of the clients send more or less the same number of announcements to all the nodes.

The nodes receiving the largest amount of announcements aren't only the three closest (ignoring the closest in the scenarios 1 and 3), they are more than three in some case. However, our goal is accomplished in this experiment, we intended to show that the number of announcements doesn't depend on the distance to the info hash of the closest nodes. We can see in all the scenarios that the number of announcements in the closest nodes is similar.

In both the scenario 1 and the scenario 2 some points of the graphic seem to be missing. They are not missing, this result is due to the logarithmic scale. In those points where there graphic seems to be missing its value is 0. As there aren't two real numbers x and y such that $x^y = 0$ we omit the value of the graphic in those points.

In the scenario 3, the node with distance 120 to the info hash has a very low number of announcements compared with the rest. We don't know the cause of this behavior, perhaps it's due to some problem in the Planetlab node.

Independently of the versions of the clients announcing themselves, nodes which aren't part of the three closest are receiving too many announcements. This behavior could be related with the limitation in the list of peers. A mechanism could exist to detect this and choose other nodes to announce. These unexpected behaviors are investigated in later experiments. This experiment also shows that UTorrent is the most popular BitTorrent client, this is important since the behavior of Mainline DHT is highly influenced by UTorrent.

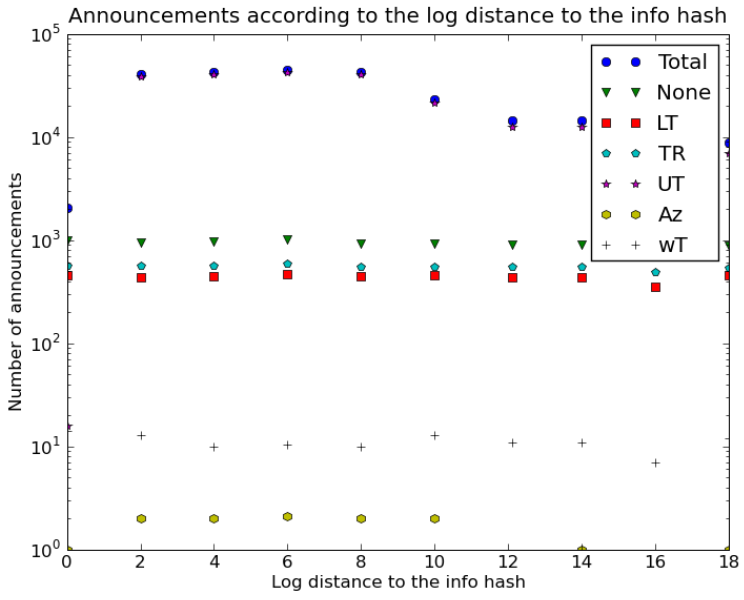

Figura H.1. Announcements in the first scenario.

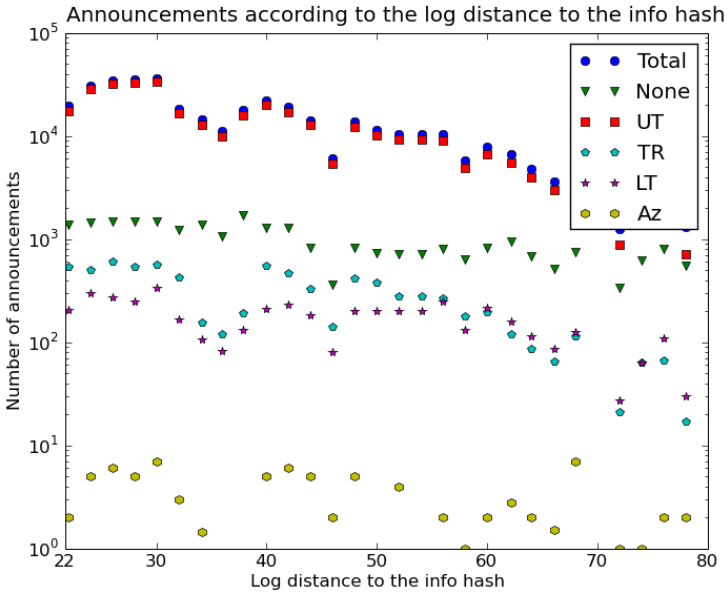

Figura H.2. Announcements in the second scenario.

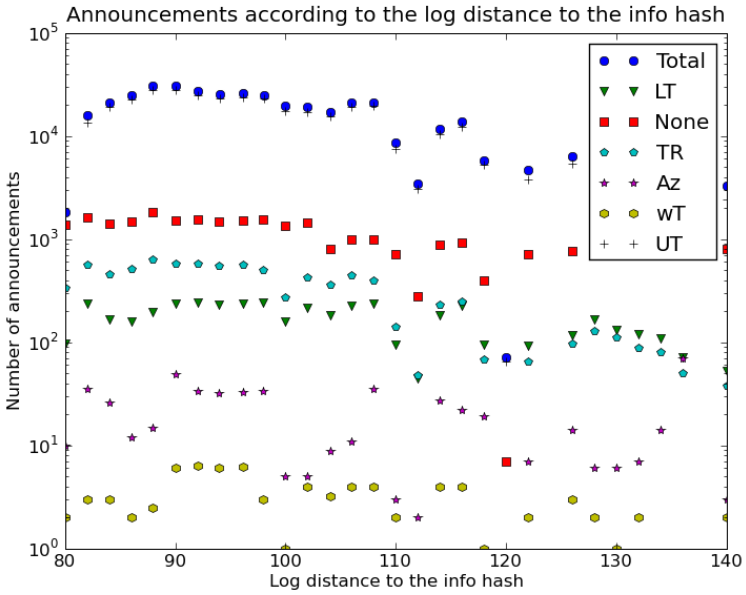

Figura H.3. Announcements in the third scenario.

Apéndice I

Experimento 3 - Período de los announcements

This experiment checks if the period of the announcements is the expected one. In this experiment a node has been positioned close to a popular info hash. It has tracked the announcements it received to observe the period of nodes announcing on it.

Furthermore, this parameter may serve to check if nodes are coordinated. The period of announcements is supposed to be the same value than their expiration time. If a node sends an announcement to a node with a different period to its period, it may stay out of the list of peers during some intervals of time. Giving a popular value for the period may help to nodes to stay in the list of peers.

I.1 Expected results

According to [7], **the period of an announcement (as well as its expiration time) is 30 minutes**.

I.2 Experiment definition

One node has been positioned in passive mode close enough to an info hash to contain its list of peers. It has recorded all the announcements it has received and has tracked those sent by the same node (identifying a node by its IP address and DHT port). From all the announcements sent by the same node, it has calculated the average time between every pair of consecutive announcements.

- Number of passive nodes: 1.
- Number of info hashes: 1 (we have chosen a popular enough info hash to get enough announcements to obtain a reliable result).
- Distance to the info hash: 130 (being the closest one).

- Time for the experiment: 3 hours. We haven chosen a longer time because, if a node sends an announcements, leaves the DHT, joins again afterwards and sends another announcements, we could wrongly think that its period is the time the node has left the DHT. We consider the time we have chosen is enough to get a reliable result.
- Data to measure: average period of all announcements for every node announcing.

The tools announcements and display_freq_announcements have been used for this experiment.

I.3 Results

We have observed a set of 581 different nodes announcing. The result of the experiment is summarized in Table I.1 and represented graphically in Figure I.1. A big part of the announcements (the 47,16%) have a period between 20 minutes and 35 minutes (which is as expected).

A strange result is the high number of nodes announcing in less than 5 minutes. This behavior is unexpected and we haven't found any reason for it, but it's very improbable that this is their actual period and expiration time. Also in other experiments we observed that many nodes send several announcements in some seconds, this is probably the cause of this result. An hypothesis about this behavior is that these nodes have restrictions in the incoming network traffic (which can be caused, for example, by a firewall). Firewalls or routers using a NAT (Network Address Translation) have some behavior which limit the incoming traffic in the computer. In this case, nodes sending several announcements may not receive the responses and that would be the reason why they send the query several times, but it should be investigated more in depth.

In this experiment it's not possible to differ a node which stays alive and sends announcements periodically from a node which leaves the DHT and joins again after some time. For example, if there are two nodes announcing, the first node sends an announcement and 30 minutes later sends another one, the second node sends an announcement, leaves the DHT, joins again after 30 minutes and sends another announcement, their behavior will be the same from the point of view of the node where they announce.

For the design of a Mainline DHT client, this result suggests a period of announcements of 20 minutes. Even though the expiration time of a queried node is 30 minutes, it'll probably refresh the list of peers with the announcement so the querier will stay on it all the time.

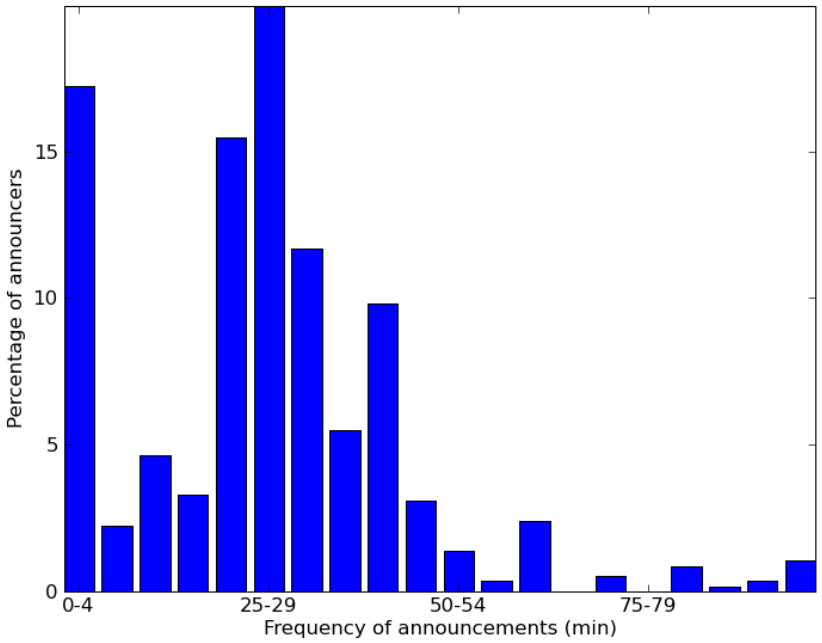

Figura I.1. Period of announcements.

Period (min)	Number of nodes
0 - 4	100 (17.21%)
5 - 9	13 (2.24%)
10 - 14	27 (4.65%)
15 - 19	19 (3.27%)
20 - 24	90 (15.49%)
25 - 29	116 (19.97%)
30 - 34	68 (11.70%)
35 - 39	32 (5.51%)
40 - 44	57 (9.81%)
45 - 49	18 (3.10%)
50 - 54	8 (1.38%)
55 - 59	2 (0.34%)
60 - 64	14 (2.41%)
65 - 69	0 (0.00%)
70 - 74	3 (0.52%)
75 - 79	0 (0.00%)
80 - 84	5 (0.86%)
85 - 89	1 (0.17%)
90 - 94	2 (0.34%)
95 +	6 (1.03%)

Tabla I.1. Period of announcements.

Apéndice J

Experimento 4 - Intento de control total de una lista de peers

This experiment checks the problem of censorship. A set of passive nodes has been positioned closer to an info hash than those containing its lists of peers. After some time, these nodes should take the total control of the list of peers. Besides, an active node has been enabled to monitor the evolution of the passive nodes and check if they get the total control of the list of peers.

We did not check if during the experiment the nodes enabled for it were all the time the closest to the info hash, however, according to the probabilities we present in Appendix F, the probability for a random node identifier to have 130 or less of distance to the info hash is 2^{-29}.

J.1 Expected results

When an announcement is received, after its expiration time, the peer which sent it will be removed from the list of peers. In order to stay on the list of peers the peer will have to send it to a set of the closest nodes to the info hash periodically, as explained in [7]. The experiment 3 shows that after one hour, all the current entries in a list of peers will probably have expired. The experiment 1 shows that the 82% of the info hashes have 20 or less nodes containing list of peers. The experiment 2 shows that, if a set of nodes are the closest to an info hash, independently of their distance, they will receive the largest amount of announcements. So, **if 23 nodes join the DHT as the closest to an info hash and they stay alive, after less than 24 hours, they will be the only nodes containing the list of peers of the info hash**.

J.2 Experiment definition

- Number of passive nodes: 23.

- Distance to the info hash of the passive nodes: from 107 to 130. Each node has been positioned in a different distance in that interval.
- Estimation of peers for the info hash: 671.
- Number of active nodes: 1.
- Frequency of lookups by the active node: 2 minutes.
- Time for the experiment: 24 hours.
- Number of info hashes: we have tried several times to control an info hash and it has not been possible in any case. For this reason, this experiment shows a typical case of how it has not been possible to control an info hash.
- Data to measure: nodes containing list of peers for the info hash and the size of their list.

The tool censor has been used for the passive nodes and the tools evolution and display_evolution for the active node to watch the passive nodes.

J.3 Results

Controlling a whole list of peers of any info hash has not been possible in any case. Even though 23 nodes have been positioned as the closest to the info hash and they have stayed alive for 24 hours, new nodes joined all the time. Our nodes, the nodes trying to control the info hash, have controlled the most of the list of peers, Figure J.1 shows it. In that figure, red circles are our nodes and blue squares are the rest. The size of the list of peers of nodes which are not ours always tends to be small whereas our nodes control the most of the list of peers peers. Our nodes never have a list of peers with more than 50 nodes due to the limitation established to make the list of peers fit in a UDP packet.

A clearer way to observe the result is by watching the number of nodes containing list of peers independently of the size of list of peers. This view shows the addition of new nodes when they are not supposed to join. Figure J.2 shows the number of nodes in every moment during the experiment. In this case, the red circles are the number of nodes positioned for this experiment (our nodes) and the blue squares are the total number of nodes. There are two intervals of half an hour where the control of the list of peers is almost total and another one where it is completely total, however, there are always new nodes joining. According to the our expected results new nodes should not join. This fact is explained in later experiments.

The number of the rest of nodes increases and decreases several times (there is even an interval of time when they are zero). This proves that the existence of other nodes containing list of peers is not due to a high expiration time or a lack of expiration of peers in their list. The reason is probably that peers send announcements to nodes which are not the closest.

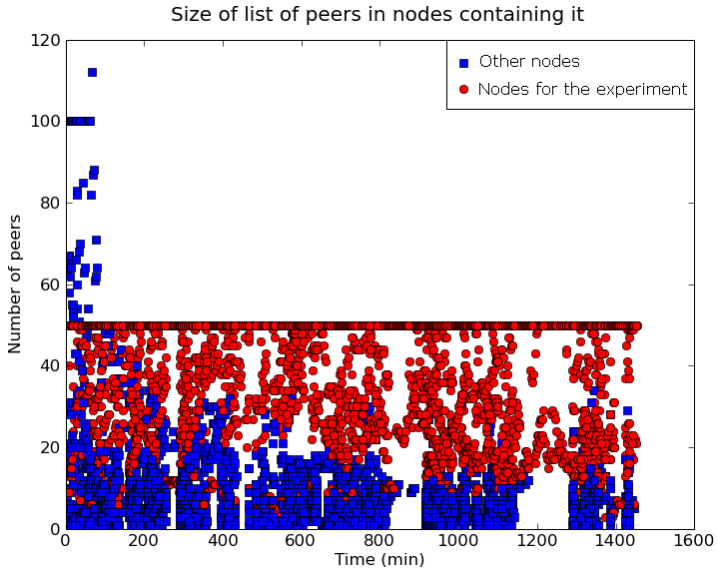

Figura J.1. Size of list of peers in nodes containing it. Red circles are our nodes enabled for this experiment and blue squares lines are the rest.

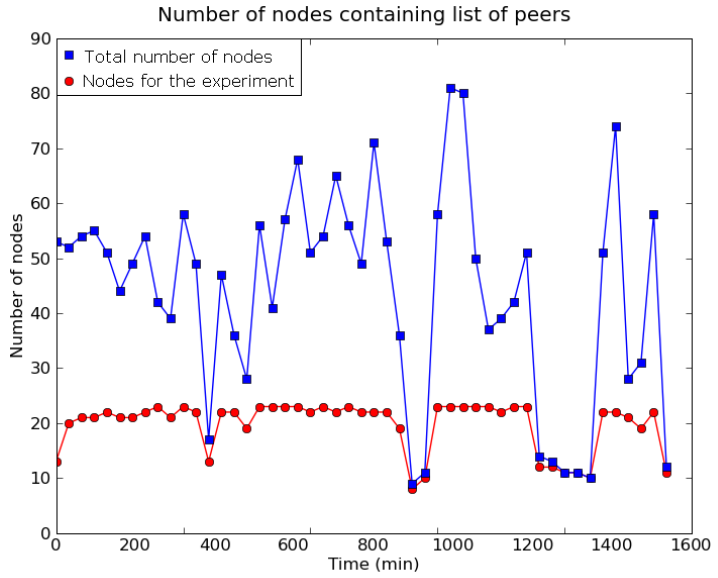

Figura J.2. Number of nodes. The red circles are the number of nodes enabled for this experiment and the blue squares are the total.

Apéndice K

Experimento 5 - Announcements desde el punto de vista del anunciante

Due to the results obtained in the experiment 4 (where it has not been possible to get the total control of a list of peers in any case), this experiment seeks the reason in the behavior of some of the most popular BitTorrent clients. This experiment studies the distance to the info hash of nodes where some clients choose to announce. The results obtained in the experiment 4 are probably due to some clients which do not respect the design rules. This experiment tries to identify those clients misusing the network.

K.1 Expected results

Every DHT client should announce itself in a set of the closest nodes to the info hash (three nodes according to [7] and eight nodes according to [20]).

K.2 Experiment definition

- BitTorrent clients used: UTorrent, KTorrent and BitSpirit.
- Info hashes used per client: 8.
- Data to measure: distance to the info hash of nodes containing list of peers as well as of those chosen for the announcement.

The tool parse_announcements has been used for this experiment.

K.3 Results

The results of this experiment are presented in Table K.1, Table K.2 and Table K.3 (the info hashes used are different in every client). They are not enough to extract

a pattern of how these clients choose the nodes to announce but it is possible to assert two facts:

- There are clients which are not announcing in three nodes as they should (according to the specifications). In the most of the cases UTorrent announces itself in three nodes but BitSpirit and KTorrent announce themselves in more than three nodes the most of the times.
- There are clients which do not announce themselves only in the closest nodes. UTorrent seems to announce itself in the three closest nodes. BitSpirit announces itself in more than three nodes but those nodes where it announces itself might be the closest to the info hash. It is almost sure that KTorrent is not choosing the closest nodes to the info hash to announce itself. Even though it announces itself in some nodes which seem to be the closest, it also announces itself in some nodes far from the info hash (later experiments show that it is almost impossible that those nodes KTorrent chooses for the announcement are the closest). In some cases it announces itself in nodes with distance 155 or 152 to the info hash, it is almost impossible that do not exist closer nodes. The nodes KTorrent chooses for the announcements could be those providing nodes in the lookup, but this is not sure.

In the results obtained using UTorrent we can observe one case where it sends 5 announcements, we do not know the reason of this behavior.

It would be interesting to do a complete analysis of the behavior of the most popular clients using the DHT and describing how they actually behave. The results we have obtained are just an approach of their actual behavior.

Info hash	Distance of nodes containing list of peers	Distance of nodes where it announces
IH1	134, 138, 139, 139	138, 138, 139
IH2	136, 137, 139, 139, 140	136, 137, 139
IH3	131, 137, 138, 138, 138	136, 137, 139
IH4	136, 138, 139	139, 140, 140
IH5	135, 139, 140	135, 135, 139
IH6	138, 138, 138	138, 138, 138, 139, 140
IH7	140, 140	138, 139, 140
IH8	138, 138, 139, 140, 140, 141, 141	138, 138, 139

Tabla K.1. Nodes where UTorrent announces itself.

Info hash	Distance of nodes containing list of peers	Distance of nodes where it announces
IH1	137, 138, 139, 139	136, 138, 139, 139, 139, 139
IH2	132, 137, 137, 138, 139, 139, 139, 139, 140	132, 137, 137, 138, 139, 139, 139, 139, 139, 140
IH3	(no nodes)	134, 136, 138, 139, 141
IH4	134, 137, 138, 138, 138, 138	134, 137, 138, 138, 139, 139, 140
IH5	138, 140, 140, 140, 140, 140, 140, 141, 141, 141, 141	138, 140, 140, 140, 140, 141, 141
IH6	137, 138, 138, 138, 139, 139, 139, 139, 140	138, 138, 138, 139, 139, 139
IH7	138, 139	138, 139, 139, 139, 140, 140, 140, 140, 140
IH8	137, 137, 137, 137, 137, 137, 138, 139, 140	136, 137, 137, 137, 137, 138, 139

Tabla K.2. Nodes where BitSpirit announces itself.

Info hash	Distance of nodes containing list of peers	Distance of nodes where it announces
IH1	147, 149, 149, 149, 150, 150, 151, 151	147, 149, 149, 149, 150
IH2	144, 144, 147, 149, 149, 150, 150, 152, 155	144, 144, 149, 149, 150, 152
IH3	142, 145, 146, 146, 146, 146, 147, 147, 148	142, 146, 146, 146, 147, 147
IH4	147, 149, 151, 151, 152, 152, 152, 152, 153	149, 151, 151, 152, 152, 152
IH5	144, 146, 147, 147, 147, 147, 147, 148, 148, 149, 150, 150, 150, 150, 151, 152	146, 147, 147, 147, 147, 148, 149, 150, 150, 150, 152
IH6	146, 147, 148, 148, 149, 151, 152, 155	146, 147, 149, 151, 152, 155
IH7	137, 141, 142, 146	137, 142, 146
IH8	143, 145, 148, 148, 148, 148, 149, 149, 150	143, 145, 148, 148, 148, 149, 149, 150

Tabla K.3. Nodes where KTorrent announces itself.

Apéndice L

Experimento 6 - Distancia al info hash de los nodos que contienen lista de peers

The goal of this experiment is getting data about the distance to the info hash of nodes containing list of peers. We have performed lookups over a large set of info hashes. This experiment has helped to observe that there are many nodes containing list of peers which are not (or at least it is very improbable that they are) the closest to the info hash. It studies how the results obtained in the experiment 5 affect the distance to the info hash of nodes containing list of peers. It also provides some data to detect suspicious situations where nodes containing list of peers are too close to the info hash.

L.1 Goal

This experiment tries to obtain data about the distance of nodes containing list of peers to the info hash. We ca not give a complete estimation of how the distribution of nodes should be since they are not always the closest, as proved in the experiment 5.

L.2 Expected results

We do not expect to find node identifiers closer than 130 to the info hash. According to the probabilities we present in Appendix F, the probability for a random node identifier to be closer to the info hash than 130 is 2^{-30}.

L.3 Experiment definition

- Number of info hashes to lookup: 1200.

- Number of active nodes: 1.
- Number of lookups over every info hash: 10 (1 every 5 seconds).
- Data to obtain: number of nodes containing list of peers for every info hash.

The tools distribution and display_distribution have been used for this experiment.

L.4 Results

The result of this experiment is represented graphically in Figure L.1. The distribution of nodes seems to be a normal distribution, however, we do not have enough data to corroborate this. Also, the number of nodes of distance higher than the average is too high. The reason may be those results obtained in the experiment 5 which prove that some clients do not announce themselves in the closest nodes. As well, it is very probable that in the lookups we did not find all the nodes containing list of peers. The nodes containing list of peers far from the info hash have been found casually following a particular path to the info hash. To find all the nodes containing list of peers we should lookup all the nodes in the DHT. We observed a total of 10768 nodes in this experiment.

With these results, we can consider suspicious nodes closer to the info hash than 130. Also, watching at the distribution, we can detect suspicious situations. For example, a situation of nodes containing list of peers at distances 135, 137, 140, 140, 141, 141 could be normal, but if these distances were 130, 130, 131, 131, 131, 132 it would be very suspicious since it would not fit with the distribution.

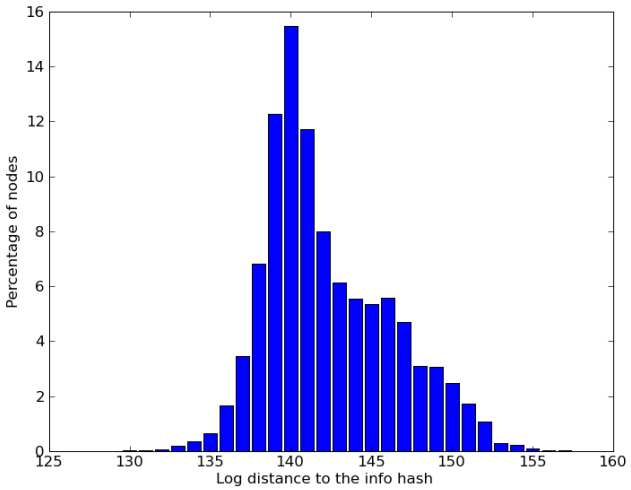

Figura L.1. Distance of nodes containing list of peers to their info hash.

Apéndice M

Experimento 7 - Porcentaje de la lista de peers contenida en los nodos en función de su orden de distancia al info hash

This experiment aims to see how the lists of peers are distributed among the nodes. It observes the percentage of the list of peers contained in every node according to its distance order to the info hash (for example, what percentage of peers has the closest, the second closest, etc). It is not expected that they add up to 100% because every node announces itself several times. It studies how the results obtained in the experiment 5 affect to the distribution of the list of peers.

M.1 Expected results

The experiment 5 shows that not all the nodes follow the rules for the design. But the three nodes to the info hash almost always get the announcements (this is also observed in the experiment 2). So, **each one of the three closest nodes to the info hash should contain the 100% of the list of peers**. In the case the three closest nodes are limiting the size of the list of peers to make it fit in a UDP packet, it could be less. This experiment also checks how many peers contain those nodes which are not part of the closest because they should not contain list of peers.

M.2 Experiment definition

- Number of info hashes to lookup: 1200.
- Number of active nodes: 1.
- Number of lookups over every info hash: 10 (1 every 5 seconds).

- Data to obtain: percentage of peers contained in every list of peers according to the distance of the node to the info hash.

The tools complete_lookup and display_list_distribution have been used for this experiment.

M.3 Results

Due to the possible existence of limitation in the lists of peers, we have divided the data of this experiment in three scenarios: a general view (Figure M.1), info hashes with 50 or less peers (Figure M.2) and info hashes with more than 50 peers (Figure M.3). The general view is based in an observation of a total of 54766 peers contained in the lists of peers.

Even though the most of the lists of peers are contained in the three closest nodes in all the scenarios, there is an important part of it contained nodes which are not the closest. In the graphic of lists of peers with 50 or less peers the peers are more contained in the three closest nodes than in the other scenarios. In the scenario of info hashes with more than 50 peers, probably the complete list of peers of the closest nodes is not obtained because they may be limiting it. Anyway, there are still some nodes containing list of peers far from the info hash, but they do not contain a high percentage of peers. The percentage of peers seems to decrease exponentially according to the closeness order of the nodes.

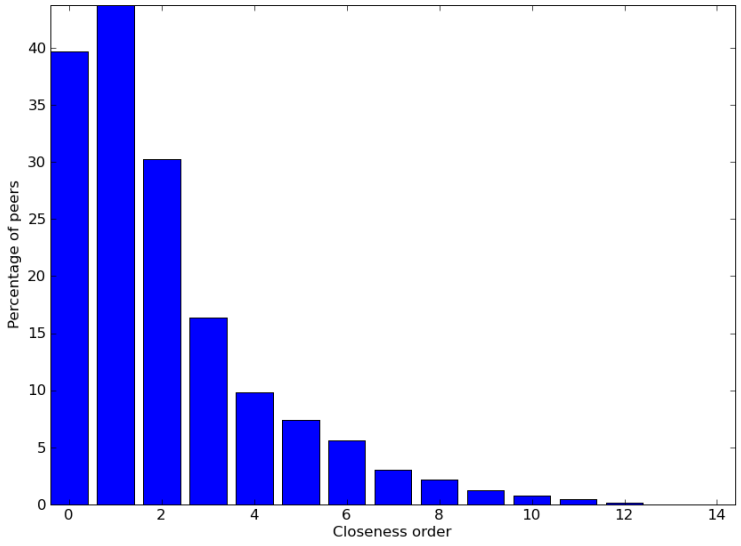

Figura M.1. Distribution of the list of peers according to the closeness to the closest node to the info hash.

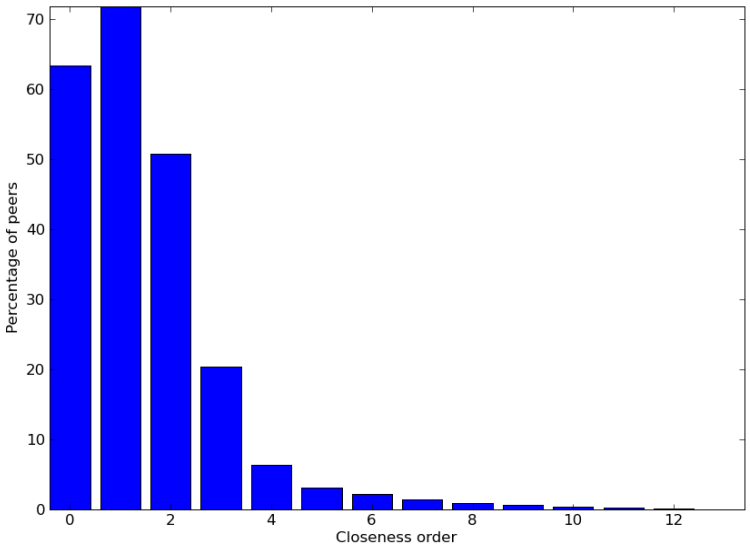

Figura M.2. Distribution of the list of peers according to the closeness to the closest node to the info hash for info hashes with 50 or less peers.

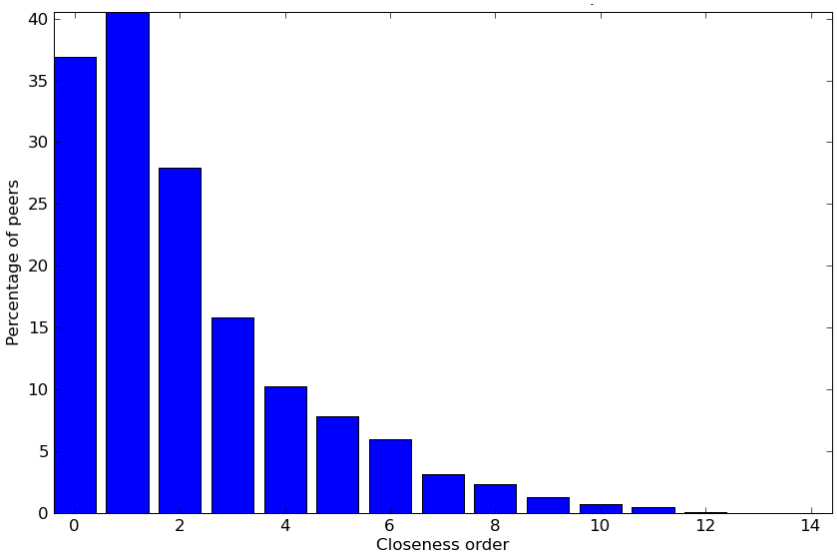

Figura M.3. Distribution of the list of peers according to the closeness to the closest node to the info hash for info hashes with more than 50 peers.

Apéndice N

Experimento 8 - Número de mensajes en función del número de peers

This experiment analyzes the scalability problem we presented in Chapter 3. In a very popular info hash, the set of the three closest nodes to the info hash may receive a huge number of messages per second generating too much traffic. This is a scalability problem because these nodes would have more overload than the rest in the DHT. It would also break with the principle of all nodes having the same responsibilities in the DHT. This experiment studies how serious this problem is and also if it may become more serious than it is.

We have positioned a node as the closest to a popular info hash to study this problem.

N.1 Goal

This experiment intends to quantify the traffic in a node containing the list of peers of a popular info hash. As peers have to announce periodically (as defined in [7]), the more popular an info hash is, the more announcements the nodes containing its list of peers will receive. Also, these nodes will receive a similar amount of get_peers queries. This experiment intends to quantify this traffic.

N.2 Experiment definition

- Info hashes to lookup: one with 107 peers, one with 536 peers, one with 1083 peers, one with 2287 peers, one with 4001 peers, one with 6076 peers, one with 11228 peers, one with 14760 peers and one with 33239 peers. To know the number of peers it has been used the estimation of peers in the torrents of The Pirate Bay.

- Number of passive nodes: 1 for each info hash.

- Time for the experiment: 11 hours and 30 minutes for every info hash starting with all of them at the same time.

- Data to measure: number of messages received of each type.

The tools incoming_traffic and incoming_traffic_statistics have been used for this experiment.

N.3 Results

The result of the experiment is summarized in Table N.1 and Table N.2. Figure N.1 shows the growth of the total number of messages and the number of messages in the minute with the highest incoming traffic according to the number of peers. In the case of info hashes with few peers, the traffic is not very high. For the info hash with 33239 the traffic is very high. The number of ping and find nodes messages does not seem to vary according to the number of peers, those making the difference are the get_peers and announcement messages.

If we focus on the minute with the maximum traffic and we watch the announcements and the get_peers messages in that minute for the info hash with the largest number of peers (2924 get_peers and 1320 announcements), knowing that an incoming get_peers message is 94 Bytes and and announcement message is 141 Bytes (in the application level, as it can be deducted from [20]), the incoming traffic rate is 7.5 KB/s which is not very high, however, the outgoing traffic is higher.

If we just take into account the get_peers messages for the outgoing traffic and just count the size of the list of peers (without the application level header), knowing that every peer is an IP address and a port number (in total 8 Bytes), and with the size of the list of peers limited to 50 (obviously, in this case it is always 50 due to the high number of announcements), the outgoing traffic in that minute is 19 KB/s. This rate is quite high just for the DHT but it can be reasonable.

In the case the list of peers were not limited to 50 peers, if we suppose that in that minute the size of the list is 10000 (if in that single minute 1320 announcements were received, in 30 minutes it could be even higher) and for every get_peers message the whole list is tried to be sent (even though the IP packets are fragmented and it can carry to errors), the outgoing traffic rate for this case would be 3.72 MB/s which is not reasonable. So, limiting the list of peers is essential to reduce the problem of scalability. Even though the problem still exists, fixing a maximum size in the list of peers raises its limitations.

Bearing the incoming traffic for the most popular info hash in The Pirate Bay with a limited list of peers has been possible. However, with a list of peers without a fix size it would have generated a huge outgoing traffic. Anyway, if a rate is reasonable or not depends on the context where it is used. In this case we have supposed the usage of Internet connections for domestic users, perhaps for a mobile device those rates are not reasonable.

The growth of the number of messages seems to be linear looking at Figure N.1, but this experiment is not large enough to prove this. The average number of messages is almost a straight line, enough samples would probably stabilize it. A large set of info hashes is necessary to study the growth. It would be useful in order to estimate the traffic in info hashes with a non existing number of peers nowadays.

We performed another experiment with an info hash having about 30000 peers as well without limiting the list of peers. The node enabled for the experiment sent about 9.8 GB in 20 hours just for the traffic of the DHT. We decided not to carry out again this experiment because we received a warning from Planetlab telling us that we had to reduce the traffic rate.

Our results prove that the problem of scalability represents a threat for the DHT. In the future info hashes with millions of peers may exist. Projects like P2P-Next [34] (which intends to broadcast live transmissions using P2P) can originate swarms with millions of peers in the broadcast of international events.

In the case of redirecting the incoming traffic as a DDoS attack (by using a fake list of peers as a response for the get_peers queries), the victim node would receive the incoming traffic corresponding to the get_peers queries. In the case of the info hash with 33239 peers it would be 4.5 KB/s which is not a dangerous rate. An info hash with many more peers or many nodes doing this at the same time would be necessary to make it serious.

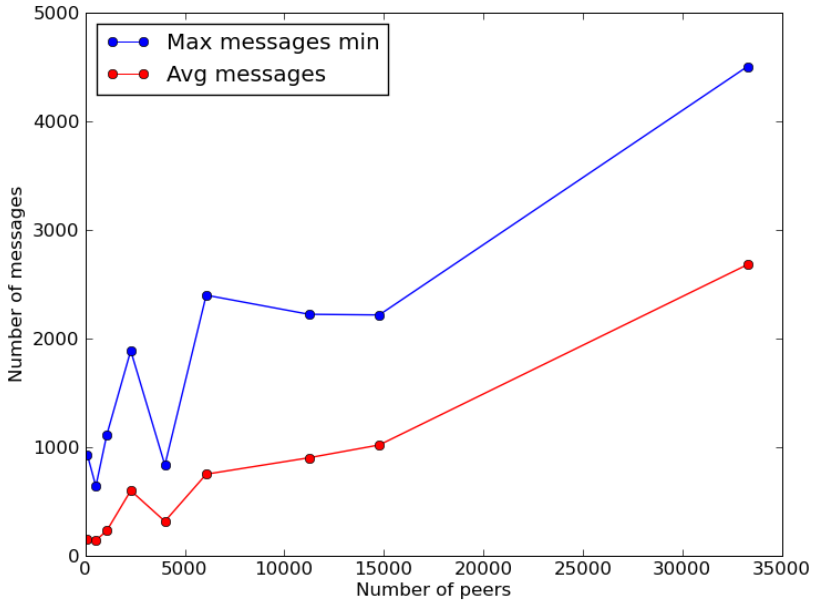

Figura N.1. Number of messages according to the number of peers.

Data	IH 1	IH 2	IH 3	IH 4	IH 5
Estimation of peers	107	536	1083	2287	4001
Experiment duration	10h 29m	10h 30m	10h 28m	10h 32m	10h 33m
Total messages	97056	88424	143447	377758	199085
Total pings	1344	1372	1810	2656	2164
Total find nodes	48182	31824	64217	73128	78157
Total get_peers	37691	46610	64660	236569	105727
Total announcements	9839	8618	12760	65405	13037
Announcements for the info hash	7452	7097	9682	61457	9373
get_peers for the info hash	31620	42979	57071	226960	97173
Announcements other hashes	2387	1521	3078	3948	3664
get_peers other hashes	6071	3631	7589	9609	8554
Maximum of messages in one minute	929	637	1110	1886	833
Pings in that minute	3	1	2	8	4
Find nodes in that minute	106	71	121	168	144
get_peers in that minute	495	431	500	1112	478
Announcements in that minute	325	134	487	598	207
Average messages per minute	154	140	228	598	315

Tabla N.1. Incoming messages for different info hashes (part1).

Data	IH 6	IH 7	IH 8	IH 9
Estimation of peers	6076	11228	14760	33239
Experiment duration	10h 31m	10h 36m	10h 34m	10h 39m
Total messages	472562	572521	645336	1711189
Total pings	2164	5298	3961	5984
Total find nodes	67398	115781	100675	136317
Total get_peers	323594	381844	446312	1228111
Total announcements	79406	69598	94388	340777
Announcements for the info hash	75980	62384	88409	330880
get_peers for the info hash	313825	360642	431533	1201341
Announcements other hashes	3426	7214	5979	9897
get_peers other hashes	9769	21202	14779	26770
Maximum of messages in one minute	2397	2221	2215	4502
Pings in that minute	2	8	8	8
Find nodes in that minute	134	254	223	250
get_peers in that minute	1367	1746	1297	2924
Announcements in that minute	894	213	687	1320
Average messages per minute	749	900	1018	2678

Tabla N.2. Incoming messages for different info hashes (part2).

Apéndice O

Experimento 9 - Distancia entre info hashes y su nodo más cercano

This experiment has measured the distance between info hashes and their closest node. This data is is useful to detect suspicious situations where nodes are too close to an info hash. It may provide a parameter to exclude suspicious nodes.

O.1 Expected results

As the identifier space is huge and its occupation is supposed to be very small (as we deduce in Appendix R). We do not expect to find node identifiers closer than 130 to the info hash. According to the probabilities we present in Appendix F, the probability for a random node identifier to be closer to the info hash than 130 is 2^{-30}.

This experiment can not be affected by clients which do not respect the rules for the announcements because, in all the cases, the closest node to an info hash will contain list of peers (as show the experiment 2 and the experiment 5). Independently of which nodes are chosen for the announcements, this experiment just focuses on the closest.

O.2 Experiment definition

- Number of active nodes: 1.
- Number of info hashes to lookup: 1200.
- Number of lookups over every info hash: 10 (1 every 5 seconds).
- Data to measure: distance between info hashes and their closest node.

The tools complete_lookup and display_distance_closest have been used for this experiment.

O.3 Results

The result of the experiment is summarized in Table O.1 and represented graphically in Figure O.1. The distribution seems a normal distribution but our data is not large enough to prove it. The most typical distance is 138. As there is not any case of nodes with distance less than 131, this data can be used as a parameter to detect suspicious situations. If we observe a node containing a list of peers which is closer than 131 to the info hash, this node can be ignored (exists the possibility that a random identifier is closer than 131 to an info hash, but it is very improbable. It does not matter if the node is ignored in this case), excluding by this way suspicious nodes (only nodes in relation with that info hash will exclude it. The rest of the DHT will treat it as a normal node, so it will not be excluded from the DHT).

Distance	Number of times
131	1 (0.11%)
132	6 (0.67%)
133	14 (1.55%)
134	26 (2.88%)
135	63 (6.98%)
136	124 (13.75%)
137	188 (20.84%)
138	227 (25.17%)
139	146 (16.19%)
140	69 (7.65%)
141	28 (3.10%)
142	7 (0.78%)
143	2 (0.22%)
144	1 (0.11%)

Tabla O.1. Distance of the closest node to the info hash to the info hash.

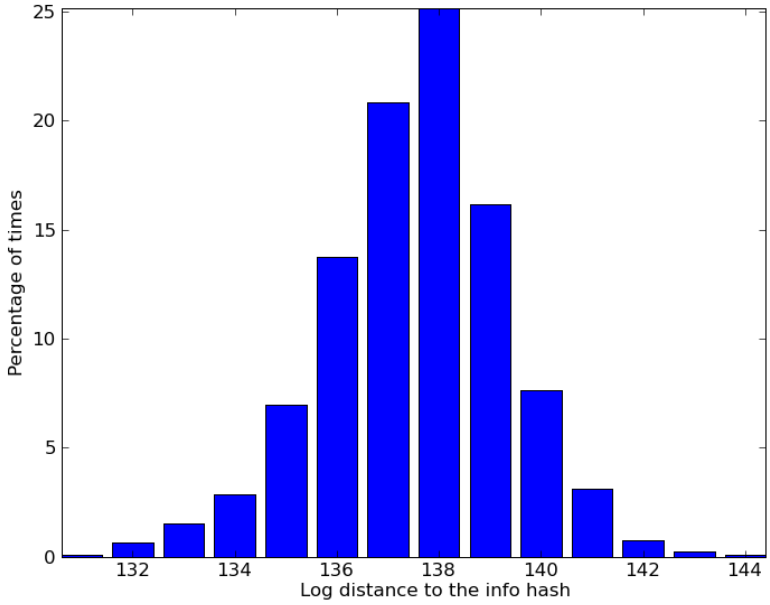

Figura O.1. Distance of the closest node to the info hash to the info hash.

Apéndice P

Diferencia de mensajes y almacenamiento entre Kademlia y Mainline DHT

Figure P.1 and Figure P.2 show the difference between a STORE query in Kademlia and an ANNOUNCE_PEER query in Mainline DHT. Figure P.3 and Figure P.4 show the difference between a FIND_VALUE query in Kademlia and a GET_PEERS query in Mainline DHT.

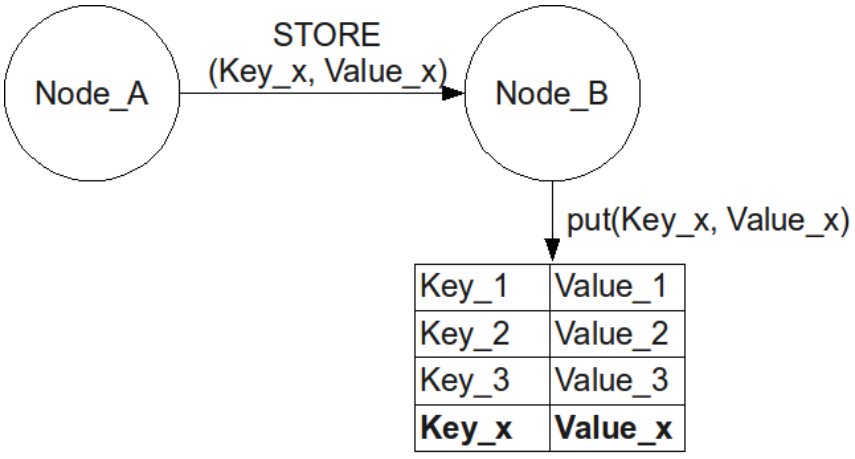

Figura P.1. Kademlia STORE query.

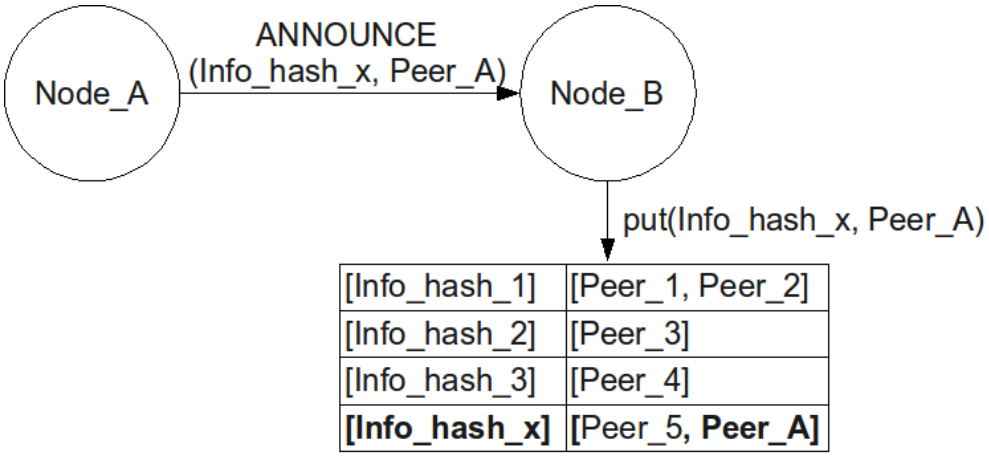

Figura P.2. Mainline DHT ANNOUNCE_PEER query.

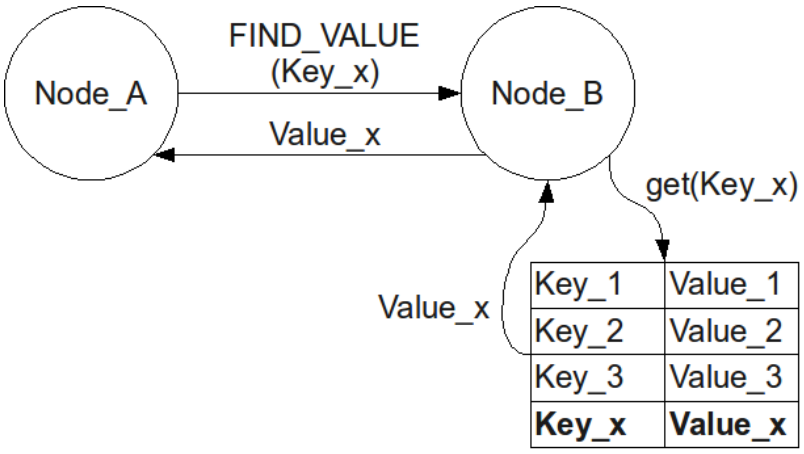

Figura P.3. Kademlia GET_VALUE query.

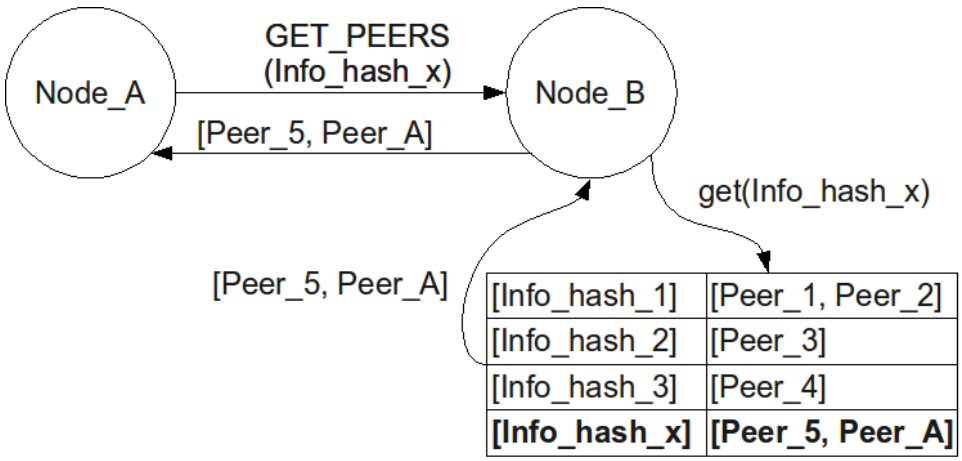

Figura P.4. Mainline DHT GET_PEERS query.

Apéndice Q

Desarrollo de las herramientas

This appendix is complemented with Appendix T which explains the methodology used. It's recommended to read Appendix T before this.

In this appendix we present the development of the tools and we give an overview of their usage. All our tools have been developed using the standalone library kadtracker (explained in Appendix T). The main reason to use kadtracker is that we had direct contact with the developers, however, there are other reasons why we have decided to use it. In this appendix we present all the reasons why we have used kadtracker instead of other choices.

Q.1 Requirement of active tools

This thesis consists on an experimental view of a part of Mainline DHT based on the analysis we present in Chapter 3. Part of the analysis we presented consists on studying different data obtained in a lookup in Mainline DHT. By analyzing this data, we can study the behavior of a set of nodes getting a global view of a part of the DHT.

Q.1.1 Choices

An important part of the data to analyze in the experiments can be obtained by looking up an info hash in the DHT. Any Mainline DHT client could perform a lookup over an info hash. Some other experiments required lookups as well but also other information like the identifiers of nodes providing list of peers or their distance to the info hash.

Using a Mainline DHT client to obtain this data was not easy since they do not usually provide the data we needed in the user interface. Furthermore, in this thesis, watching the data was not enough, we had to process. So, a Mainline DHT client had to be modified again in order to log this data.

Q.1.2 Final choice

Kadtracker offered a very easy interface to do this operation and it lacks of a user interface since it is a standalone library. Getting and processing the data we needed by modifying another client would have been a harder task.

Q.2 Requirement of passive tools

This thesis claims to check some possible vulnerabilities in Mainline DHT. The first requirement to test the vulnerabilities was collecting data from a node close to an info hash, this is a common point in all of them. Also, some experiment of profiling needs to pick up information from a node containing list of peers.

From the existing closest nodes to an info hash we could collect some data but there is some other data that we could not. The total traffic generated in this kind of nodes could not be studied unless they were ours. Also, we could not choose how far the closest nodes to the info hash were. We had to study the behavior of nodes from different distances to the info hash, it would be impossible to do that without adding new nodes in the DHT. Then, for our experiments, we had to add new nodes to the DHT.

Q.2.1 Choices

We could have used any Mainline DHT client to position a node in the DHT, but clients choose a random identifier, so its position in the DHT would be random. It would have made impossible to choose an info hash in order to contain its list of peers. Then, we had to modify any Mainline DHT to choose its identifier.

Q.2.2 Final choice

Among all the available Mainline DHT clients to modify, we chose kadtracker for the reasons explained in Appendix T. We modified it to choose its identifier, but also some data had to be logged from its incoming and outgoing packets. To test the scalability problem, we had to know the number of messages generated and their type. We could have obtained this data by using a tool to capture the network traffic (like Wireshark [19]). We discarded this choice because it would have taken more time than modifying kadtracker.

The decision was modifying kadtracker to log this data itself. It was easy to find the code where the incoming messages were received and the outgoing messages were sent. These parts of the code were modified to log the data we needed (without modifying any functionality).

Q.3 Modifying kadtracker

In this thesis, the goal is not changing the functionality of the DHT client (mainly because it has to follow the protocol). Only three modifications have been done in the passive mode:

- Choice of the node identifier. Instead of letting the client to choose its identifier randomly, the option to choose the identifier when the client is started has been added. But, instead of applying for the identifier (which can be quite tricky to calculate), an info hash and the log distance the node has to be from it must be provided. This is an easy way to position nodes as close to an info hash as wanted (otherwise the calculation should be made in advance).

- Censorship list. When a get_peers query is received, the node checks a list with IP addresses contained in a file. If the querier's IP address is contained in this list, an empty list of peers is returned.

- Limit in the size of the list of peers. This change was made when it was observed that, when the list of peers grew up to more than 50 peers, it did not fit in a UDP packet. At the same time it was observed that many clients returned always lists of 50 or less peers. It was decided to limit the list of peers to the last 50 announcements. This patch has been integrated in kadtracker.

The passive node with these modifications has been called *censor*. The active node has not had any functional modification, this node has been called *observer*. Using these clients, in every experiment one or both of them have been modified properly to obtain the wished results. These modifications do not affect to any functionality, they have consisted on logging that data of interest for the experiments.

For the most of the experiments, another complementary tool has been developed to display the information graphically or summarized.

Q.4 Set of tools

The most of the experiments use a pair of tools: one to log information (called logger) and one to process the logged information (called displayer). The loggers can be split in two groups: active loggers and passive loggers. Active loggers are those which are a modification of the observer (providing the functionality of an active node). Passive loggers are those which are a modification of the censor (providing the functionality of a passive node).

Table Q.1 presents a classification of the pairs of tools with passive loggers and Table Q.2 the same with active loggers. A summary of the purpose of every pair of tools is given in both the tables. Each pair of tools has been used for one experiment. The functionalities a tool needs arose in the definition of the experiments.

Q.5 Tool parse_announcements

We developed another tool independent from the others. The purpose of this tool is showing those nodes where a DHT client chooses to announce itself. All the BitTorrent clients have to respect the message format, and their network traffic can be captured in a pcap file (using, for example, Wireshark [19]). The tool parse_announcements processes this pcap files and, for every lookup, shows the distance to the info hash of nodes containing list of peers for it and the distance to the info hash of nodes where the client chooses to announce.

This tool parses a pcap file providing an interface to get the fields of the application level packets. The packet format in the application level is explained in Appendix C.

Logger	Displayer	Result
censor	-	Positions a node with a given distance to an info hash in the DHT. It is the base to develop the other passive loggers.
announcements	display_announcements_distance	Displays graphically the number of announcements received in a node according to its distance to the info hash. It requires a log for every distance to display.
announcements	display_freq_announcements	Calculates the average frequency of the announcements for every node announcing and displays the statistics of these frequencies.
incoming_traffic	display_incoming_traffic_statistics	Summarizes the received traffic in a node showing the number of received messages for every type of message.

Tabla Q.1. Tools with passive loggers and their purpose.

Logger	Displayer	Result
complete_lookup	display_list_distribution	Displays how the list of peers is distributed among all the peers containing it for the given info hashes.
complete_lookup	display_distance_closest	Displays the average distance between the given info hashes and their closest node.
complete_lookup	display_average_number_nodes	Displays the average number of nodes containing list of peers for the given info hashes.
distribution	display_distribution	Displays the distance between the given info hashes and all nodes containing list of peers for them.
evolution	display_evolution	Displays how a list of peers grows for every node containing list of peers for a given info hash.

Tabla Q.2. Tools with active loggers and their purpose.

Apéndice R

Ocupación del espacio de identificadores

Positioning new nodes closer to an info hash than the closest one should be easy. According to the estimation of nodes made in [7], the number of nodes in the DHT is 1.3 million. It is a quite old estimation, supposing that nowadays it has grown a lot and we overestimate it to $16,000,000 \approx 2^{24}$, and knowing that the identifier space is 2^{160}, the identifier space occupation would be approximately $\frac{2^{24}}{2^{160}} \approx 2^{-136}$. Supposing that nodes are uniformly distributed, the average space between two consecutive nodes would be $\frac{2^{160}}{2^{24}} \approx 2^{136}$.

Even though since that estimation the number of nodes had grown much more or the distribution of nodes were not uniform, this average distance would still be huge. In conclusion, distance between an info hash and its closest node is supposed to be big enough to add there millions of nodes.

Apéndice S

Trabajos relacionados

In this appendix we present some work about BitTorrent and Mainline DHT related with this thesis.

S.1 Profiling work

An important part of this thesis is profiling Mainline DHT. We profile some parts of the DHT by analyzing it and checking the analysis empirically. Profiling helps to understand more in depth how the DHT works looking at its actual behavior. It also helps both to design some experiments and to understand their results.

In the specifications of Mainline DHT, there are two parameters which are not very clear. The design of Kademlia explains that values should be stored in the k closest nodes to a key (k is a free choice parameter) and fixes the expiration time in one hour (but it says it can be modified for optimizations). In the specifications of Mainline DHT k is equal to 8 (as asserted in [20]) and values are lists of peers, so, the eight closest nodes to the info hash should be chosen for the announcements. But, Crosby and Wallach [7], which analyze both Mainline DHT and Azureus DHT, say explicitly that nodes in Mainline DHT announce in the three closest nodes to the info hash. Crosby and Wallach [7] also assert that the frequency of announcements is 30 minutes. The expiration time fits with the design of Kademlia since in explains that it can be modified. However, both the definitions in the number of nodes chosen for the announcements are inconsistent. Nevertheless, the value of these parameters does not affect too much the rest of the analysis we made.

Existing profiling works may help to design experiments and to do some estimations. For example, as we have explained in this section, [7] is the base for some of our hypothesis. This document also gives a global profiling of different aspects of Mainline DHT like an estimation of the total number of nodes. Although this is a quite old data, it has been useful to calculate the approximate occupation of the identifier space that we present in Chapter 3 (although since the estimation the number of nodes may have changed). In [15], some aspects of Azureus DHT are profiled, this DHT has many properties in common with Mainline DHT, some

measurements in Azureus DHT may have a similar value in Mainline DHT.

Yu and Fang et al. [16] study the identifier distribution in KAD DHT (the DHT of eMule [17]). KAD is also a DHT based on Kademlia and its node identifiers are also supposed to be random, any hypothesis about KAD can be interesting to check in Mainline DHT since they are similar DHTs. This work is very related with this thesis since the identifier distribution is an important part of our work. In the work of Yu and Fang et al. [16] it is interesting to watch that the repetition of node identifiers is higher than it should. It could be interesting to study the repetition of identifiers in Mainline DHT, it could influence in the analysis of nodes distribution. In general, the knowledge of other DHTs can help to find solutions to problems and arise questions that could be interesting to investigate in Mainline DHT.

S.2 Vulnerabilities

The analysis of the possible vulnerabilities we present in Chapter 3 is mainly deducted from [2] and supported by [20] and [7]. Although, as said before, they have inconsistent definitions of some parameters, both of them still support our analysis. The differences they have only change the resources necessary to carry out the vulnerabilities, and this difference is not very relevant. In these documents there is not any documented mechanism to prevent the vulnerabilities we study. However, it could exist any implicit mechanism to prevent them.

The most of the work about vulnerabilities in this thesis is possible due to the possibility of choosing an identifier in Mainline DHT. In the specifications of Mainline DHT there is no any mention to this possibility, it is just said that node identifiers should be random. Also, it is not possible to know if a single identifier is random or not.

Azureus DHT is the other DHT of BitTorrent based on Kademlia. Even though it is not used in this thesis, its policies about node identifiers are very interesting in the context of this thesis. In [24] it is explained how node identifiers in Azureus DHT are a hash of the IP address of the node. This provides a mechanism to find nodes an identifier not calculates as a hash of the IP address. This mechanism gave us the idea to study what the implications of choosing an identifier in Mainline DHT are since non random identifiers can not be detected. It should be studied if there is any reason why Mainline DHT chooses random identifiers.

S.3 Documented vulnerabilities

One of the goals of this thesis is studying some possible vulnerabilities of Mainline DHT. Mainline DHT is something recent inside BitTorrent. The BitTorrent protocol has several documented vulnerabilities like those presented in [11], where they show some vulnerabilities of the protocol, [12], where they use the protocol to perform DDoS attacks, [13], where they exploit the protocol to achieve higher download rates

and [14], where they show another exploit to get higher rates without contributing by uploading.

For Mainline DHT, there are not many documents studying vulnerabilities. Any document studying a vulnerability (either for BitTorrent, for Mainline DHT or for another DHT) is useful in order to see how a vulnerability has to be analyzed and tested. Also by reading about vulnerabilities in other DHTs we can wonder if it would possible in Mainline DHT.

Apéndice T

Metodología

This appendix presents and justifies the decisions we made for the development of the tools and the design of the experiments. Performing the experiments could be achieved by different ways, we had to make some decisions for their execution. For each decision, we evaluate the different alternatives and we try to choose the most simple and efficient. We also identify the implications the methodology may have. This appendix is complemented with Appendix Q, which motivates the development of the tools and explains their usage.

T.1 Choice of technologies and resources

In Chapter 3 we present a theoretical analysis about Mainline DHT. We needed to check experimentally this analysis throughout some experiments. These experiments required a way to interact with Mainline DHT. We couldn't find any tool which offered the functionalities we were looking for (we explain these requirements in Appendix Q).

We decided to develop a set of tools. We did it using a Python [33] library which implements a Mainline DHT client called kadtracker provided by the TSLab department of KTH. Another choice to develop the tools was modifying any existing BitTorrent client since a lot of clients are Open Source. The reasons why we chose kadtracker are mainly two:

1. Kadtracker is implemented in Python. BitTorrent was originally designed to be implemented in Python, aspects like the message format (described in [20]) are described using native Python data structures (as lists, dictionaries, etc). Using Python as programming language makes easier to understand and to implement the protocol.

2. Kadtracker is a standalone library. This fact makes easier to modify it and adapt it as a tool. If we adapted another BitTorrent client for our purposes, we should locate the specific part of code necessary to modify among all the functionalities.

Kadtracker uses the version 2.5 of Python, so, a the version 2.5 or a later version is needed to run it properly. In the development of our tools we did not add any functionality which needed a more recent version of Python.

Some experiments required the usage of a large set of nodes (up to about 30 nodes). As nodes in the DHT may be identified by their IP address, for this kind of experiments it was not enough to run the set of nodes in one computer. We needed a way to run several nodes behind different IP addresses. We have used the network Planetlab [3] to run this kind of experiments. Planetlab is a set of computers located all around the world used to perform experiments using up to hundreds of machines at the same time.

T.1.1 Planetlab

Planetlab is a set of computers (called nodes) located all around the world used for investigation purposes. We got access to it throughout KTH. We had access to hundreds of computers but we did not use more than 30 in any case.

All the nodes in Planetlab use a Unix based operating system. Furthermore, all of them have the version 2.5 or later of Python, which is the one needed to run out tools. Nodes in Planetlab have been chosen trying to have them distributed around the world.

Nodes in Planetlab are supposed to stay alive during the experiments, but we have not monitored if they face some problems during the execution of the experiments like connectivity problems or system reboots.

T.2 Kadtracker

Kadtracker is a standalone library which implements a Mainline DHT client. It offers two possible behaviors for a node in the DHT:

- Passive mode: the node stays in the DHT and responds the queries it receives.
- Active mode: the node performs lookups. Given an info hash, the node gets its list of peers by finding those nodes containing it (in case they exist).

Every execution of a node must include the passive mode even if the node has also an active behavior. Every node has a responsibility in the storage of lists of peers and routing, this tasks must not be avoided. If they were avoided perhaps the results obtained could be altered due to the lack of participation in the DHT. In general, we tried to influence as less as possible in the normal behavior of the DHT in order not to alter the results of the experiments.

T.3 Experiments

In order to check experimentally the analysis presented in Chapter 3, we had to carry out some experiments. To do these experiments we developed a set of tools

modifying kadtracker. Once the tools were ready we executed the experiments. All the experiments have been performed using Planetlab nodes.

T.3.1 Limitation in the list of peers

Something which may affect the results of the experiments is the limitation of the list of peers to 50 peers as maximum. We discovered that a list of peers with more than 50 peers does not fit in a UDP packet. Many nodes we observed in the experiments also limit their list of peers to 50 peers. We modified kadtracker so when a list of peers grows up to more than 50 peers, the 50 most recent announcements are returned. This policy intends to keep in the list of peers the highest amount of alive nodes. Other policies could have been used like returning random peers.

We found some nodes returning list of peers with more than 50 peers. This implies an IP packet fragmentation and many times we observed that the packets were not received properly. We do not know if the decision of limiting the list of peers may affect somehow to the behavior of the nodes and so to the results of the experiments.

T.3.2 Types of experiments

The experiments can be classified into three types:

- Active experiments: are those where we generate lookup queries over an info hash. We use an active node to execute them.
- Passive experiments: are those where we positioned a node in the DHT and we analyze the incoming queries it gets. We use a passive node to execute them.
- Active-passive experiments: are a combination of the other two kinds. We position a passive node in the DHT to analyze the incoming queries and an active node to send queries to the other.

In some passive experiments we position a node as the closest to an info hash. They are supposed to be the closest to the info hash all the time during the experiment, however, we did not check it explicitly. Nodes in the DHT are supposed to choose their identifier randomly. The probability that a node with a random identifier is closer to the info hash than our nodes is very reduced according to the results we obtained in a experiment where we measured the distance between info hashes and their closest nodes.

The passive experiments get an observation of an altered scenario, this means that the data is collected by taking part in the scenario and not just by observing it. Moreover, the nodes used for this experiment chose their own identifier. However, there was not any other possible choice to watch the behavior of a node containing a given list of peers. Any BitTorrent client with a random identifier may contain a list of peers but to contain the list of peers we wanted we had to choose the identifier.

Some of the active experiments which collect data from the DHT perform 10 lookups (1 every 5 seconds) over each info hash of a large set of info hashes. This strategy may seem an aggressive way to obtain the information because a single lookup could be enough. The reason to use this strategy is trying to give time to all the nodes to respond and finding all the possible nodes containing the list of peers. Some experiments did not find all the nodes containing list of peers in the first lookup, sometimes more than one lookup was necessary. In our active experiments it is important to find all (or at least, the most of) the nodes containing list of peers. It is still possible that in any case we did not find all the nodes containing list of peers.

T.3.3 Validity of experiments

When we tested the problem of censorship, no IP addresses were censored. It consisted on trying to control a whole list of peers. Perhaps, if we had censored the access to the list of peers to all the querying nodes, the result would have been different.

The reason why we did not carry out censorship is that we were not able to control a whole list of peers in any case. In case we had been able, we would have narrowed in the problem studying different scenarios of the problem and different usages of controlling a list of peers.

A similar case is the case of DDoS attacks. In this case we have demonstrated that it can be serious since the incoming traffic rate in a node containing list of peers may become huge, however, we did not provide fake lists of peers to perform a DDoS attack. We do not know if the nodes asking for peers would have performed the DDoS attacks but we do not know any reason why they would not since they can not know in advance if a list of peers is fake or not.

T.4 Information management

Most of the experiments use two tools to obtain the information: a Mainline DHT client modified to log information and a displayer to process the logged data. None of the modifications of the clients to log information implies a functional modification. The steps to carry out this kind of experiments are the following (also shown in Figure T.1):

1. The modified client generates an output containing the logged data we want to process.

2. A displayer program parses the log file, obtains the results and shows them graphically or summarized.

The reasons why the experiments are carried out using two programs and a log file instead of doing everything in a single program are the following:

Figura T.1. Schema of how the tools manage the information.

- Maximizing the independence of the DHT client with the experiment.
- Some experiments require a lot of time (up to a couple of days). If there is any problem why a test should stop, it is preferable to have some results stored instead of losing everything and running the experiment again.
- In case of finding an error in a displayer program, this mechanism avoids the necessity of doing the experiment again.
- Interest on keeping the log file because, in the future, from the same experiment, more information may be extracted, so, in that case, the experiment should not be repeated again.
- The displayer programs can be used as a back end for logs generated using other DHTs based on Kademlia.

T.5 The log distance metric

As we explained in Chapter 2, distance between an info hash and a node identifier is measured as the bitwise XOR of both 160-bit sequences. As distances can be very large numbers, since they belong to the interval $[0, 2^{160})$, in this thesis distances are measured using log distance.

XOR distance is the result of the bitwise XOR of two 160-bit sequences. Log distance performs the same operation, however, the result is not the number itself but the position (counting from the right and starting from zero) of the first bit which is 1 (counting from the left). For example, if the distance between two sequences is in the interval $[2^{30},2^{31})$, the log distance will be 30. We show some examples of log distance using 8-bit identifiers in Figure T.2.

Using log distance, distances will always belong to the interval $[-1, 159]$ which is much easier to manage than distances in the interval $[0, 2^{160})$ obtained using XOR distance. We consider the special case of log distance equal to -1 when the XOR distance is equal to 0.

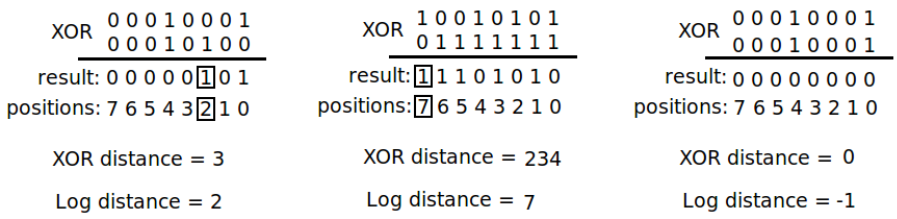

Figura T.2. Examples of log distance using 8-bit identifiers.

Apéndice U

Planificación

This thesis had planned the following phases:

- Literature study: understanding the background and context of the work.
- Theoretical analysis: analyzing the results we expected to find with the experiments.
- Understanding kadtracker: reading the code of the Mainline DHT client to understand how it works and how we could modify it.
- First contact with the DHT: doing some modifications in the Mainline DHT client interacting with the DHT in order to understand better how the client worked.
- Experiments of censorship and DDoS attacks: checking experimentally the problems of censorship and DDoS attacks.
- Analysis of results: once the results about censorship and DDoS attacks were obtained, we had to analyze the results collected.
- Experiments of scalability: quantifying experimentally the problem of scalability.
- Analysis of results: once the results of scalability were obtained, we had to analyze the results collected.
- Summary of global results: summarizing all the results from a global context.
- Report writing: preparing the final report of the thesis.
- Presentation: preparing the final presentation of the thesis.

Table U.1 shows the estimated planning and Table U.1 shows the real time used for the thesis. The work took longer than expected.

Activity	Weeks
Literature Study	5 weeks
Theoretical analysis	1 week
Understanding kadtracker	2 weeks
First contact with the DHT	2 weeks
Experiments of censorship and DDoS attacks	2.5 weeks
Analysis of results	1 week
Experiments of scalability	2.5 weeks
Analysis of results	1 week
Summary of global results	1 week
Report writing	6 weeks
Presentation	1 week
Estimated hours per week	25 hours
Total weeks	**25 weeks**
Total estimated hours	**625 hours**

Tabla U.1. Planning

We realized that in the experiments of censorship we were not obtaining the results we expected so, we stopped those experiments. Then, we studied the environment of the problem by studying other parameters experimentally in order to design better the experiments. Once we did this, we observed that censorship was not possible. In order to find the reason of these results, we checked the behavior of different clients and we observed that some clients were not behaving as expected. That explained why censorship was not possible. This result explained the reason of some unexpected behaviors we had observed, and then, we analyzed how they influenced the DHT. Finally we studied the problem of scalability.

It also took longer than expected writing the report. As a scientific work, justifying an motivating the ideas was harder than expected. It was different than writing the report of a thesis about software development.

Activity	Weeks
Literature study	5 weeks
Theoretical analysis	1 week
Understanding kadtracker	2 weeks
First contact with the DHT	1 weeks
First phase of experiments of censorship and DDoS attacks	1 week
Experiments of the environment	2 weeks
Second phase of experiments of censorship and DDoS attacks	2 weeks
Looking for a reason of the results and influence of different behaviors of clients in the DHT	2 weeks
Analysis of results	1 week
Experiments of scalability	2 weeks
Analysis of results	1 week
Summary of global results	1 week
Report writing	9 weeks
Presentation	1 week
Estimated hours per week	25 hours
Total weeks	**31 weeks**
Total estimated hours	**775 hours**

Tabla U.2. Real work

Bibliografía

[1] B. Cohen. Incentives build robustness in BitTorrent. In Workshop on Economic of Peer-to-Peer Systems, Berkeley, CA, June 2003.

[2] P. Maymounkov and D Mazières. Kademlia: A peer-to-peer Information System Based on the XOR Metric. In proceedings of IPTPS, Cambridge, MA, IEEE, March 2002.

[3] Planetlab. http://www.planet-lab.org/ (last visited May 2010).

[4] R. Schollmeier. A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. In proceedings of the First International Conference on Peer-to-Peer Computing, IEEE, 2002.

[5] Stoica and Ion et al. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In proceedings of SIGCOMM (ACM Press New York, NY, USA), 2001.

[6] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, November 2001.

[7] S. Crosby and D. Wallach. An Analysis of BitTorrent's Two Kademlia-Based DHTs. In department of Computer Science, Rice University, Houston, Texas, USA, 2007.

[8] The Pirate Bay. http://thepiratebay.org/ (last visited May 2010).

[9] Mininova. http://www.mininova.org/ (last visited May 2010).

[10] A. Legout, G. Urvoy-Kellerand and P. Michiardi. Understanding bittorrent: An experimental perspective. Technical Report (inria-00000156, version 3 - 9 November 2005), INRIA. Sophia Antipolis, France, November 2005.

[11] K. El Defrawy, M. Gjoka and A. Markopoulou. BotTorrent: misusing BitTorrent to launch DDoS attacks. In proceedings of the 3rd USENIX workshop on Steps to reducing unwanted traffic on the internet, p.1-6, Santa Clara, CA, June 18, 2007.

[12] K. Cheung Sia. DDoS Vulnerability Analysis of Bit-Torrent Protocol. In UCLA Tech. Report, Spring 2006.

[13] N. Liogkas, R. Nelson, E. Kohler and L. Zhang. Exploiting BitTorrent for fun (but not profit). In Proc. of IPTPS, 2006.

[14] M. Sirivianos, J. H. Park, R. Chen and X. Yang. Free-riding in BitTorrent Networks with the Large View Exploit. In Proc. of IPTPS, Bellevue, WA, February 2007.

[15] J. Falkner, M. Piatek, J. P. John, A. Krishnamurthy and T. Anderson. Profiling a million user dht. In proceedings of the 7th ACM SIGCOMM conference on Internet measurement, San Diego, California, USA, October 24-26, 2007.

[16] J. Yu, C. Fang, J. Xu, E. C. Chang and Z. Li. ID repetition in KAD. In Citeseer, 2010.

[17] eMule. http://www.emule-project.net/ (last visited May 2010).

[18] RFC 4251. The Secure Shell (SSH) Protocol Architecture. http://www.ietf.org/rfc/rfc4251.txt.

[19] Wireshark. http://www.wireshark.org/ (last visited May 2010).

[20] BEP0005: DHT Protocol - http://www.bittorrent.org/beps/bep_0005.html (last visited May 2010).

[21] BEP0020: Peer ID convention. http://www.bittorrent.org/beps/bep_0020.html (last visited June 2010).

[22] Mainline - http://www.bittorrent.com/ (last visited June 2010).

[23] Azureus - http://azureus.sourceforge.net/ (last visited June 2010).

[24] M. Steiner and E. W. Biersack. Crawling Azureus. In Technical Report RR-08-233, 2008.

[25] RFC3174. US Secure Hash Algorithm 1 (SHA1). http://www.ietf.org/rfc/rfc3174.txt.

[26] UTorrent. http://www.utorrent.com/ (last visited June 2010).

[27] KTorrent. http://ktorrent.org/ (last visited June 2010).

[28] BitSpirit - http://bitspirit.uptodown.com (last visited June 2010).

[29] BEP0003: The BitTorrent Protocol Specification. http://www.bittorrent.org/beps/bep_0003.html (last visited June 2010).

[30] D. Wu, P.Dhungel, X. Hei, C. Zhang, K. W. RossUnderstanding Peer Exchange in BitTorrent Systems. In Proc. of IEEE International Conference on Peer-to-Peer Computing (IEEE P2P), Delft, Netherlands, Aug 2010.

[31] I. Kelényi and J. K. Nurminen. Energy aspects of peer cooperation - Measurements with a mobile DHT system. In Proc. Cognitive and Cooperative Wireless Networks Workshop in the IEEE International Conference on Communications Beijing, China, 2008, pp. 164 - 168, 2008.

[32] Ares. http://aresgalaxy.sourceforge.net/ (last visited June 2010).

[33] Python. http://www.python.org/ (last visited June 2010).

[34] P2P-Next. http://www.p2p-next.org/ (last visited June 2010).

KTH Information and
Communication Technology

Exploring Mainline DHT: an experimental approach

ISMAEL SAAD GARCÍA

Master Thesis at TSLab
Supervisor: Raúl Jiménez
Examiner: Björn Knutsson

Abstract

BitTorrent is nowadays one of the most popular object sharing P2P networks, it has millions of users. It provides an efficient way to distribute objects to a large number of clients, encouraging clients who download an object to share it with the rest.

To obtain peers to exchange an object with, recent versions of BitTorrent start using a DHT. The DHT is a mechanism to distribute the storage of the lists of peers participating in the distribution of the object among all the nodes participating in the P2P network. BitTorrent has two DHTs: Mainline DHT and Azureus DHT. We study Mainline DHT.

We study the actual generation, distribution and obtaining of lists of peers in Mainline DHT. We make a theoretical analysis of this part of the DHT and we compare it with the actual behavior. We obtain an experimental profile where we identify unexpected situations and some cases where the performance of the DHT may be improved.

Furthermore, according to our analysis, there are situations where the DHT is vulnerable, making possible: censorship by denying the access to the object exchange, routing traffic as a DDoS attack and a problem of scalability. We check these problems experimentally and document them. The analysis helped us to design the experiments which show a robustness of the DHT against censorship and, on the other hand, a serious problem of scalability.

We also present a set of tools we have developed to interact with the DHT in order to carry out the experiments. These tools are Open Source and they can be used to do further investigations.

Acknowledgments

I want to thank my supervisor Raúl Jiménez and my examiner Björn Knutsson for all their help during the development of this thesis. I also thank my colleague Sara Dar for her support.

Thanks to my father Mahmoud Saad, my mother Teresa García and my brother Carlos Saad who where always supporting me from the distance.

Finally, thanks to all the friends who were next to me during this fantastic time: David, Javier, Daniel C, Miguel, Peio, Silvana, Luigi, Elena, Daniel W, Jens, Sebastian, Alexandra, Paolo, Victor, Adrian, Marta, Marcos, Tony, Xandra, Luis, Pablo, Beatriz and José.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

P2P (Peer-to-Peer) networks are a kind of network where all the participants have the same responsibilities. BitTorrent is nowadays one of the most popular P2P networks, it has millions of users. This thesis explores the actual generation, distribution and obtaining of values in a DHT (Distributed Hash Table) of the P2P network BitTorrent [1]. We present a theoretical analysis of that part of the DHT and we compare it with its actual behavior.

The theoretical analysis arises some vulnerabilities of the DHT about: censorship of contents, using the DHT to peform DDoS attacks and a problem of scalability. We have tested them empirically. In P2P networks tasks are supposed to be equally distributed among all the participants, the vulnerabilities we study imply a lack of equality. So, discovering vulnerabilities in a P2P network like BitTorrent can improve the performance and security of the software in benefit of millions of users.

1.1 Background summary

P2P (Peer-to-Peer) networks are a kind of network which intend to lack of centralized entities, so, their service responsibility is distributed among all the entities participating on them. BitTorrent is one of the most popular P2P networks nowadays, it has millions of users. Its purpose is providing a distributed, scalable and fault tolerant mechanism to share files (like software, movies, music, etc).

In BitTorrent, files are obtained following three steps: discovering the file, getting nodes to download the file from (called peers) and, finally, starting the file download as well as uploading to other peers the already downloaded parts of the file. A list of peers of the file is provided by a centralized server (called tracker).

The existence of a tracker as a centralized entity may carry to problems of scalability and fault tolerance. To avoid the necessity of trackers, recent versions of

BitTorrent tend to be trackerless by delegating this task little by little to a DHT (Distributed Hash Table). BitTorrent has two DHTs which implement Kademlia [2] (which is the design of a DHT): Mainline DHT (developved by [22]) and Azureus DHT (developed by [23]). This thesis studies Mainline DHT.

Mainline DHT is a distributed mechanism to manage the lists of peers downloading or containing files. Each file has an identifier called info hash. When a node starts downloading a file, it sends periodically a query (called announcement) to a set of the closest nodes to the info hash in order to be included in the list of peers. Therefore, the nodes which store the list of peers are chosen deterministically.

When a node wants to obtain the list of peers of a file, it asks the closest nodes to the info hash of the file about closer nodes to the info hash until it finds those nodes containing the list of peers.

1.2 Scope

The first part of this thesis is a profiling of a part of the DHT. When a node starts downloading a file, sends periodically an announcement to a set of the closest nodes to the info hash of the file in order to be included in the list of peers. Focusing on this fact we study different aspects about it: how many nodes contain a list of peers, how many nodes are chosen for the announcement and how they are chosen, what is the frequency of the announcements, how lists of peers are distributed among nodes and the distance to the info hash of nodes containing list of peers.

The profiling work is firstly analyzed theoretically, we present an analysis based on the specifications and design of Mainline DHT. The analysis has to be checked empirically because, even though it can provide a reasonable approach, the actual behavior of the DHT may not match with the analysis for two reasons:

- The coexistence of different clients in the DHT: Mainline DHT is an open P2P network, then, anyone can develop a software to join it. There are several implementations of Mainline DHT clients coexisting, so, if they do not behave following the specifications, they may influence the actual behavior of the DHT.
- The popularity of some info hashes: info hashes with a large amount of peers may imply an imbalance in the DHT (an info hash with 10 peers may influence the DHT in a different way than a info hash with a million of peers). This imbalance may alter the behavior of the DHT.

The second part of the thesis is studying some possible vulnerabilities deducted in the analysis. Again, the analysis points out they are real but it is necessary to check them empirically since, due to the reasons explained previously in this section, the actual behavior may be different from the theoretical.

All the vulnerabilities are a different point of view of the same scenario. As we explained before, the list of peers of a file is stored in a deterministic set of nodes (the closest to the info hash). Nodes are able to choose its position in the DHT (as we explain in Chapter 3), then, they can position itself close to an info hash to contain the list of peers of a file. This possibility may carry the following problems:

- Censorship: if a set of nodes agrees to position close to an info hash, they may become the only entity responsible for the storage of the list of peers of the file. By controlling the list of peers of an info hash, they can deny the access to the file download to other nodes.

- DDoS attacks: a list of peers is the IP address and a TCP port of one or more computers. Nodes asking for peers will query the IP addresses they find in the list. So, if a node contains the list of peers of a very popular file, by providing a fake list of peers, it can redirect all the incoming queries which ask it for peers routing them as a DDoS attack.

- Scalability: if a node contains the list of peers of a very popular file, the incoming and outgoing traffic it will have may be huge. In the DHT there are millions of nodes so, if a file becomes very popular, its list of peers will be huge and, consequently, the number of queries the nodes containing its list of peers will receive will be huge as well.

We also present a set of tools we have developed in order to study the actual behavior of the DHT empirically as well as the possible vulnerabilities. These tools are Open Source so anyone can use it for any research related with Mainline DHT.

1.3 Goals

The goals this thesis intends to reach are the following:

- Comparing the theoretical analysis with the actual behavior of Mainline DHT.

- Providing an experimental profile of the actual generation, distribution and obtaining of the lists of peers in Mainline DHT.

- Providing a set of tools to monitor Mainline DHT.

- Exploring the problem of censorship in Mainline DHT.

- Quantifying the problems of scalability and DDoS attacks in Mainline DHT.

1.4 Motivation

The main motivation of this thesis is exploring Mainline DHT. With the exploration, we expect to get some long term results which are also a motivation for this work. These expected long term results are:

- Finding situations where the performance of the DHT may be improved.
- Providing a profile of an unexplored area of the DHT in order to contribute in future investigations.
- Ensuring equality in the distribution of tasks among nodes in the DHT.
- Improving the scalability in the DHT.
- Improving the security of the DHT.

1.5 Structure

This thesis is organized with the following structure: Chapter 2 explains the context of the thesis, describing the concepts necessary to understand it. Chapter 3 presents all the theoretical analysis which is tested with the experiments. Chapter 4 references the work related with this thesis and all the texts which can help to understand it better. Chapter 5 defines the methodology used for the experiments and justifies the decisions made. Chapter 6 classifies all the tools developed and explains their purpose. Chapter 7 shows the results obtained in the experiments. Chapter 8 summarizes all the results of the experiments giving a global view of all of them. Chapter 9 presents all the interesting facts found during this thesis requiring more work. Chapter 10 explains the conclusion of the work.

Chapter 2

Background

This chapter presents the context of this thesis and all the previous knowledge needed to understand our work. It gives an overview of the global context narrowing in the specific points which we have studied.

2.1 Distributed systems

The classic model of service providing between computers has traditionally been the existence of one server and several clients (as shown in Figure 2.1). This model is used in several services. In those cases where a large number of clients apply for the same service, this model can carry to problems of robustness and scalability. If a server providing a service stops working or there are too many clients overloading it, the service may become unavailable.

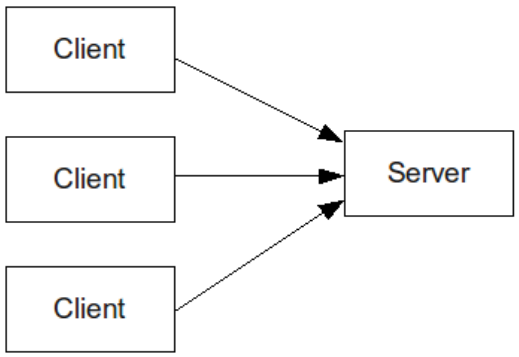

Figure 2.1. Classic client server architecture.

A way to prevent problems of robustness and scalability is by replicating the

server, creating then a redundancy in the service (as shown in Figure 2.2). This solution improves the robustness and scalability but still may have the same problems. The existence of centralized entities usually implies a limit in the robustness and scalability since they always can fail or being overloaded (even though redundancy reduces that possibility).

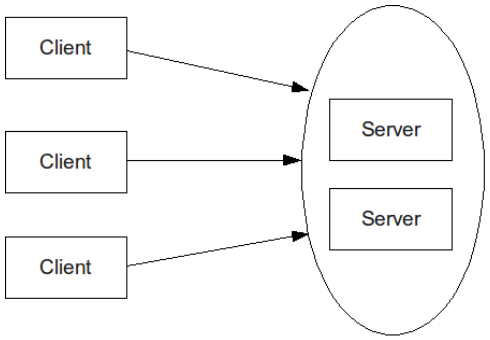

Figure 2.2. Replicated server architecture.

One of the main goals of P2P (Peer-to-Peer) networks is addressing problems of robustness and scalability by proposing a completely different architecture. They avoid or minimize the existence of centralized entities by distributing the service tasks among all the participants. Figure 2.3 shows the architecture of a P2P networks without centralized entities.

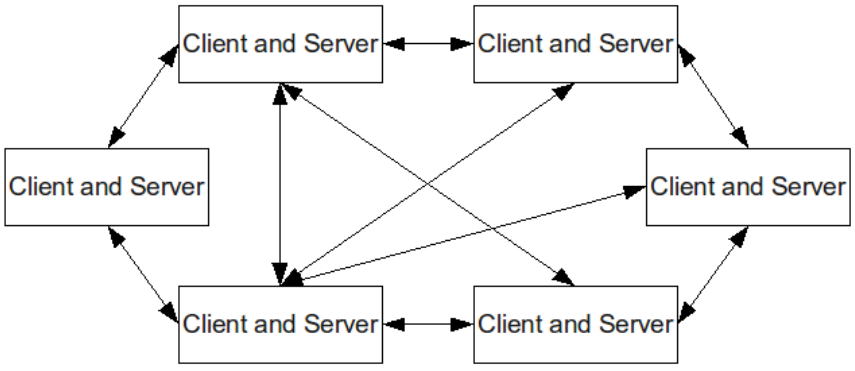

Figure 2.3. P2P networks decentralized architecture.

2.2 P2P networks and DHTs

As defined in [4], "*a P2P (Peer-to-Peer) network is a kind of network where participants share a part of their own hardware resources. These shared resources are necessary to provide the service and content offered by the network. Due to this distribution, P2P networks do not require the intermediation or support of a global centralized server or authority*". In P2P networks, service responsibility is distributed among all the participants.

A DHT (Distributed Hash Table) is a mechanism to store and retrieve values indexed by keys. This task is distributed among all the participating entities. Every node is responsible for the storage of a set of values whose key is in a given range. A DHT provides a mechanism to find the node or the set of nodes responsible for the storage of a value using its key. The methodology used to manage the storage of the (key, value) pairs is the main point in the design of a DHT. There are several designs of DHTs among which it is possible to distinguish Chord [5], Pastry [6] or Kademlia [2]. Kademlia is the one studied in this thesis.

2.3 Kademlia

Kademlia is a design of a DHT. On it, every value has a key associated to it which is a 160-bit identifier. Besides, every node chooses a random 160-bit identifier when joins the DHT. A value is stored in a set of the closest nodes to its key (this property is one of the most important design aspects to understand the work presented in this thesis since it is the base for part of the analysis we present in Chapter 3).

2.3.1 XOR distance

Distance between a node identifier and a key is measured as the bitwise XOR function of both of the 160-bit sequences. Also, the XOR metric is symmetric, so $distance(x, y) = distance(y, x)$. Table 2.1 and Table 2.2 show some examples of XOR distance supposing 8-bit identifiers.

The XOR metric is unidirectional because, for any given node x and distance $z > 0$, there is exactly one point y such that $distance(x, y) = z$ (as explained in [2]).

X	00000001 (1)	00001010 (10)	00000001 (1)
Y	00000010 (2)	00010100 (20)	11001000 (200)
XOR distance	00000011 (3)	00011110 (30)	11001001 (201)

Table 2.1. Examples of XOR distance using 8-bit identifiers (part 1).

X	11111010 (250)	11111111 (255)	00000001 (1)
Y	11111111 (255)	00000001 (1)	11111111 (255)
XOR distance	00000101 (5)	11111110 (254)	11111110 (254)

Table 2.2. Examples of XOR distance using 8-bit identifiers (part 2)

2.3.2 Finding nodes

When a node wants to find those nodes where a value is supposed to be stored (either to retrieve it or to send it for its storage), it has to know the value's key. Using this key, the node asks iteratively to the closest known nodes to the key about closer nodes until it finds the closest nodes to the key. As the XOR metric is unidirectional, all the lookups for the same key converge along the same path, regardless the original node.

2.3.3 Messages

All the messages in Kademlia are sent using the UDP protocol. Kademlia consists of four query types:

- PING: probes a node to see if it is online.
- FIND_NODE: takes a key and returns the IP address, UDP port and node identifier of the k closest nodes to the key.
- STORE: instructs a node to store a key and its value.
- FIND_VALUE: behaves like FIND_NODE with one exception, if it is found a node that had previously received a STORE query for the key, it returns the stored value.

To store a (key, value) pair, a node finds the k closest nodes to the key and sends them a STORE query. Besides, each node has to republish periodically the (key, value) pairs it has every hour because the nodes responsible for its storage will remove it after a fix time (which is one hour). We clarify that it is not the same to have a (key, value) pair than being responsible for its storage.

2.3.4 Routing table

Every node in the DHT manages a routing table where information about a set of known nodes is stored and updated. The routing table stores, for each $0 \leq i < 160$, the IP address, UDP port and node identifier of k (this parameter is a free choice

in the implementation) nodes of distance in the interval $[2^i, 2^{i+1})$ from itself in what they call k-bucket. This table is updated with information contained in the incoming messages and sending ping messages to nodes on it. It tries to keep on it those nodes which have been alive for longer.

2.4 BitTorrent

BitTorrent [1] is an object-sharing P2P network (objects are, in most of cases, files like software, movies, music, etc). Its goal is providing an efficient way to distribute objects to a large number of clients, encouraging clients who download an object to share it with the rest. A client participating in the exchange of an object is called peer of the object. When a client wants to download an object it needs a torrent file (which is a file containing meta-data about the object). It can be downloaded, for example, from a web server (some of the largest torrent distributors are The Pirate Bay [8] or Mininova [9]). Once the torrent file is obtained, the client can start downloading the object.

2.4.1 How BitTorrent works

In order to obtain an object in BitTorrent, it's necessary to follow these steps:

1. Content discovery: getting a torrent file with metadata about the object.
2. Peer discovery: getting peers to exchange the object with.
3. Data exchange: exchanging the object with the peers.

The first setp in the object obtaining is getting a torrent file. The file is not obtained using BitTorrent, it has to be downloaded, for example, from a web server. Part of the data contained in a torrent file is optionally the URL of a tracker (which is a server that assists in the communication between peers). The client uses this link to ask the tracker for peers of the object it wants to obtain. The client establishes a communication with the peers and it starts downloading the object, as well as providing to other clients the chunks of the object it has. The object exchange is performed using the BitTorrent protocol, which is explained in [1].

The peer discovery was originally performed by asking the tracker for peers, however, trackers, as a centralized entity, may suppose a problem of scalability and fault tolerance (even though they do not take part in the data exchange). In recent versions of BitTorrent, the peer discovery is being delegated little by little to a DHT. Figure 2.4 summarizes all the process of the object obtaining.

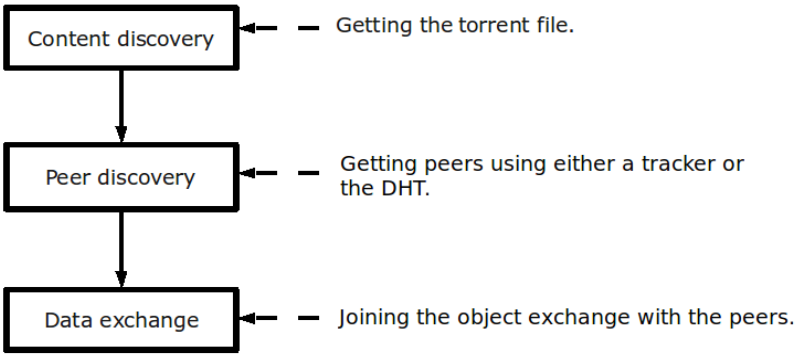

Figure 2.4. How to obtain an object in BitTorrent.

The BitTorrent protocol (which is specified in [29]) is in charge of the object exchange. The BitTorrent protocol encourages peers to upload the parts of the objects they have. This protocol uses TCP protocol for the communications.

The set of all the peers sharing an object is called swarm. When a peer joins the object exchange it is said that it joins the swarm. Peers can be classified in two types: seeders and leechers. Seeders are those peers containing a complete copy of the object. Leechers are those peers who are downloading an object and do not have the 100% of it. Besides, each peer knows some other peers and they can exchange the peers they know, [30] explains this mechanism.

There are several BitTorrent clients. A client is a software implementing the BitTorrent protocol. Most of the clients implement both the BitTorrent protocol and the DHT protocol. Although they usually still use trackers to obtain peers, most of clients have also compatibility with the DHT and they use it to get more peers. Some of the most popular BitTorrent clients are: UTorrent [26], BitSpirit [27] and KTorrent [28].

2.5 Mainline DHT

There are two DHTs in BitTorrent and both of them are an implementation of Kademlia, they are Mainline DHT (developed by [22]) and Azureus DHT (developed by [23]). This thesis studies Mainline DHT.

2.5.1 Implementation of Kademlia

As an implementation of Kademlia, the most of the aspects in the design of Kademlia are used in Mainline DHT, however, some details (specified in [20]) are added in the implementation. They are the following:

- Keys are a 160-bit SHA-1 hash [25] derived from the object called info hash. Using SHA-1 hashes ensures that it is almost impossible that two info hashes have the same value. Beides, hash functions guarantee uniform distribution by definition. The info hash is contained in the torrent file of the object.
- It chooses k = 8 (as we explained before, k is in Kademlia the number of nodes stored in a k-bucket as well as the number of nodes chosen to send them a STORE query)
- The expiration time is 30 minutes instead of one hour, as explained in [7] (the design of Kademlia [2] specifies that this time can be modified to optimize the DHT).

2.5.2 Lists of peers as values

In Mainline DHT, values are lists of peers, so, what provides a lookup in Mainline DHT is a list of peers (IP address and TCP port for the BitTorrent protocol for every peer). With this list, it is possible to join to the data exchange starting a communication with the peers by using the BitTorrent protocol. The retrieval of a list of peers is called get_peers lookup.

Having list of peers as values means that a node does not query another in order to store a value, but to be included in the list of peers of the object when it joins the data exchange. In this case, we say that this node is announcing and the query is called announcement or announce peer. Announcements have as well an expiration time.

Although lists of peers are values too, they are dynamic because they grow and decrease with the announcements and their expirations. Figure 2.5 and Figure 2.6 show the difference between a STORE query in Kademlia and an ANNOUNCE_PEER query in Mainline DHT. Figure 2.7 and Figure 2.8 show the difference between a FIND_VALUE query in Kademlia and a GET_PEERS query in Mainline DHT.

2.5.3 Messages

Like Kademlia, Mainline DHT consists of four very similar query types (the packet format of these queries is explained in Appendix C and defined in [20]):

- PING: probes a node to see if it is online.
- FIND_NODE: takes an info hash and returns the IP address, UDP port and node identifier of the k(8) closest nodes to the info hash.
- ANNOUNCE_PEER: instructs a node to be added in the list of peers of an info hash.
- GET_PEERS: behaves like FIND_NODE with one exception, if it is found a node that had previously received any ANNOUNCE_PEER query for the info hash, it returns the list of peers.

2.5.4 Other issues

There is an important fact to highlight, the design of Kademlia [2] says explicitly that values are stored in the k closest nodes to the key. In the specifications of Mainline DHT [20], it says that chooses k = 8. According to this, in Mainline DHT, peers should announce in the eight closest nodes to the info hash (because the document does not say explicitly anything about the number of nodes where a peer should announce). However, Crosby and Wallach [7] say that in Mainline DHT nodes announce in the three closest nodes to the info hash. These definitions are inconsistent between them. We have checked this parameter experimentally in order to know its value.

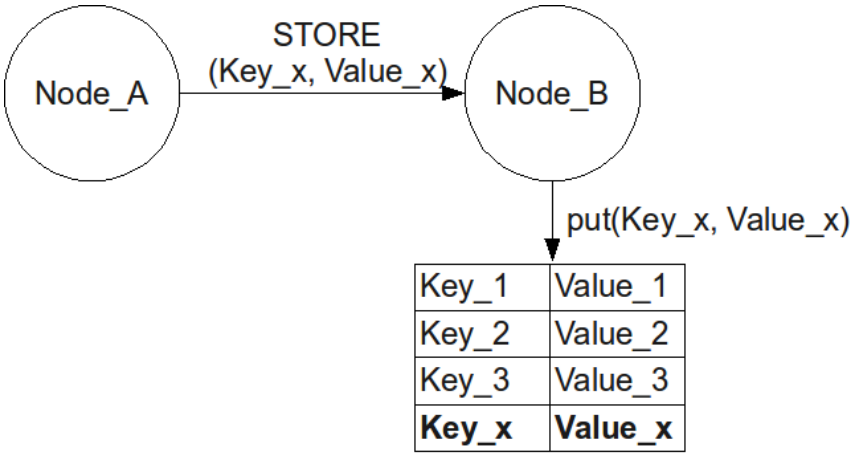

Figure 2.5. Kademlia STORE query.

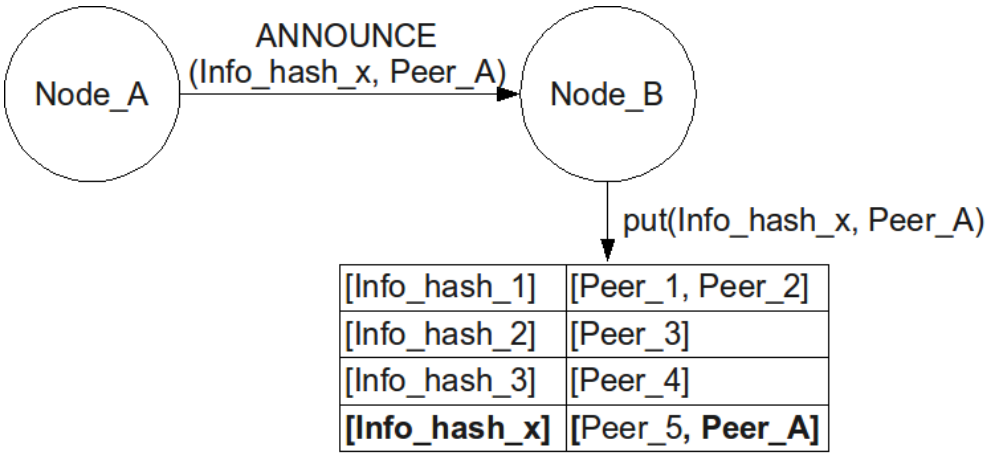

Figure 2.6. Mainline DHT ANNOUNCE_PEER query.

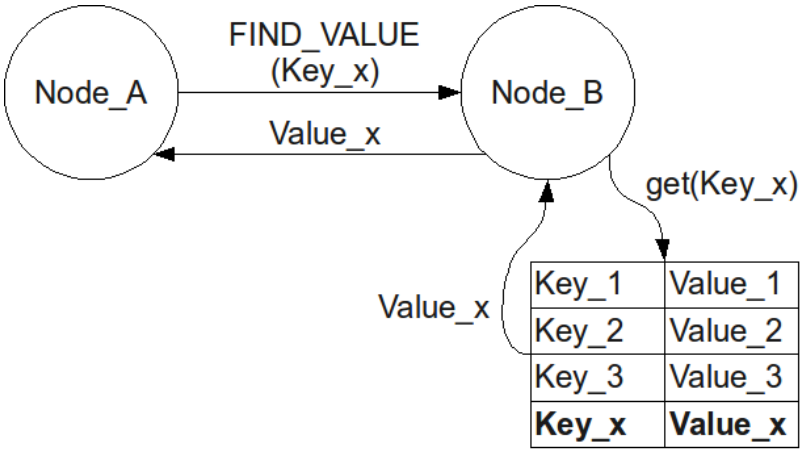

Figure 2.7. Kademlia GET_VALUE query.

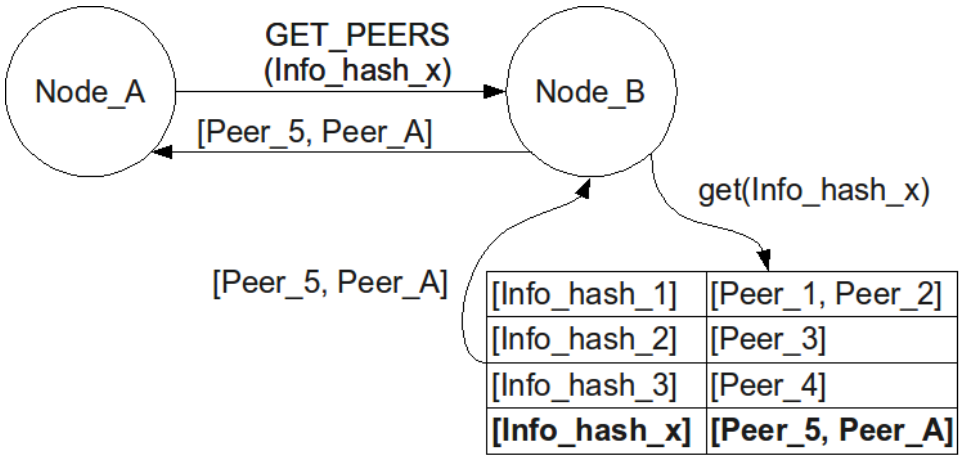

Figure 2.8. Mainline DHT GET_PEERS query.

Chapter 3

Analysis

This chapter presents the theoretical analysis that we check empirically in Chapter 7.

3.1 Frequency of announcements

The expiration time of an announcement must be the same value than its frequency so that when it expires it is sent again. In the specifications of Mainline DHT this value is not explicitly defined, but [7] explains that the expiration time is 30 minutes (although the design of Kademlia [2] fixes it in one hour, it also says that it can be modified for optimizations).

By checking this value we can compare the analysis with the actual behavior of the DHT. Also, with this data it has been easier to design some experiments and detect unexpected behaviors in nodes announcing.

Checking the frequency of announcements is also useful to check if nodes are coordinated. For example, if a node has a frequency of 30 minutes and sends an announcement to a node whose frequency is 15 minutes, as the expiration time should be the same than the frequency of the announcements, the peer would not be in the list of peers 15 of every 30 minutes. Knowing the expiration time of the most of clients can give a value of frequency to help peers to stay in the lists of peers.

3.1.1 Expected results

The design of Kademlia [2] says that the expiration time of a STORE query is one hour, however, this value can be modified for optimizations. Crosby and Wallach [7] say explicitly that this value is 30 minutes. The specifications of Mainline DHT

[20] does not specify any value. This information points out that **the expiration time as well as the frequency of announcements is 30 minutes**.

3.2 Number of nodes containing list of peers

According to Crosby and Wallach [7], a peer announces itself in the three closest nodes to the info hash of the object. However, according to the design of Kademlia [2], a node stores a value in the k closest nodes to the key (in Mainline DHT values are the lists of peers). The specifications of Mainline DHT [20] specify that, on it, k is equal to 8 and they do not specify explicitly the number of nodes where a node should announce itself. These definitions are inconsistent between them. Independently of the number of nodes for the announcement, both [7] and [2] say that nodes announce in the closest nodes to the info hash.

We have checked this parameter experimentally in order to check the analysis and to clear it, we present the results in Chapter 7. For the analysis presented in this chapter we have assumed that the value of the parameter is three. Checking this value has also been useful to design the experiment of censorship. If we know the number of nodes containing a list of peers, the number of nodes necessary to censor an info hash would be the same.

3.2.1 Expected results

Crosby and Wallach [7] say that nodes announce themselves in the three closest nodes to the info hash, then, the number of nodes containing list of peers of an info hash should be three. It could be less than three nodes in the case that one of the nodes containing list of peers leaves the DHT and the peers have not still announced themselves again in another node. More than three nodes containing list of peers may exist if one new node joins the DHT closer to the info hash than those containing the list of peers, then, new announcements will income to this node and not anymore to the farthest one of the others.

In any case, as we have said, **the number of nodes containing list of peers should be a number close to three** due to the expiration and the periodicity of the announcements. Then, the average number of nodes containing a list of peers should be a number close to three. Also, as all the info hashes should have about three nodes containing its list of peers, the standard deviation in this value should be a small number. If nodes containing list of peers stay alive for a long time this number should be exactly three (as shown in Figure 3.1).

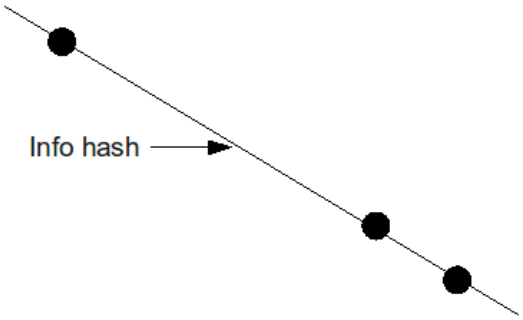

Figure 3.1. Nodes containing the list of peers.

3.3 Positioning a node in Mainline DHT to contain a list of peers

The specifications of Mainline DHT [20] explain that nodes have to choose a random 160-bit identifier. The identifier a node has will position it in a given part of the DHT. This position in the DHT is relative, this means that depending on its identifier, it will be closer or farther to other node identifiers and info hashes. The design of Kademlia [2] explains that, because node identifiers are randomly chosen, it follows that highly non-uniform distributions are unlikely.

Info hashes also follow a uniform distribution. They are a SHA-1 hash of the content of the object. By definition, a hash function must ensure a uniform distribution in its values. If all the nodes choose their identifier randomly, they will not know in advance which lists of peers they will contain.

Azureus DHT (developed by [23]) is another DHT for BitTorrent based on Kademlia, it has many characteristics in common with Mainline DHT. However, in Azureus DHT, identifiers are not chosen randomly but as a hash of the IP address of the node (as explained in [24]). Using this mechanism, it is possible to know if a node has calculated its identifier as a hash of its IP address or not. If the hash of the IP address is calculated, it can be compared to the identifier provided by the node. If they match, the node has calculated its identifier as a hash of its IP address, otherwise it has not. Nodes with an identifier which is not a hash of their IP address can be ignored, so no communication would be established with them. By doing this, only nodes with an identifier which is a hash of their IP address would stay in the DHT.

In Mainline DHT, as identifiers are random, it is not possible to check if they are have been chosen randomly or not.

3.3.1 Analysis

As Mainline DHT is an open network and any client can join, the software to implement the DHT protocol can be created by anyone. The choice of the identifier can be modified in a BitTorrent client to choose any non random identifier. Info hashes of objects are known since they are contained in the torrent files and they are immutable. By this way, a node can choose an identifier in order to be close to an info hash.

By performing a lookup over an info hash, the set of nodes containing the list of peers of an object can be found. These nodes are supposed to be the closest to the info hash (as defined in [2]). In the response to the lookup, they provide a list of peers but also information about themselves as their identifier (the message format of Mainline DHT messages is explained in Appendix C and defined in [20]).

By knowing the distance between an info hash and its closest nodes, and being able to choose the identifier for a node, the node can be positioned in the DHT closer to an info hash than the nodes containing its list of peers. The node will start receiving announcements and having a list of peers. At the same time, the farthest node to the info hash containing list of peers will not receive announcements anymore, so its list of peers will start decreasing when the announcements start expiring until it is empty. Then, it should be possible to add one node to the DHT and make it contain a list of peers of a given info hash.

An important part of the experimental work of this thesis requires to collect some data from a node containing the list of peers of an info hash. According to our analysis it is possible to make a node to contain any list of peers.

3.4 Getting the control of a whole list of peers

According to the analysis presented in this chapter, the number of nodes containing list of peers of an info hash should be about three. Also, the frequency and the expiration time of announcements should be 30 minutes. Furthermore, a node is able to choose its identifier and position itself anywhere in the DHT.

3.4.1 Analysis

Taking into account the analysis explained in the previous sections of this chapter, it should be possible to control a whole list of peers by following these steps:

1. Getting the info hash of the object contained in the torrent file of the object.

2. Performing a lookup and finding those nodes containing a list of peers (they should be about three).

3. Looking at their distance to the info hash and positioning three new nodes closer to the info hash than the closest one (the situation would be as shown in Figure 3.2)

4. Waiting the expiration time (which should be 30 minutes).

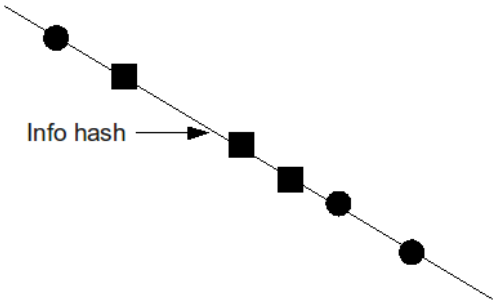

Figure 3.2. Nodes containing the list of peers after the addition of new nodes. Square shapes represent the new nodes.

After following these steps, peers would not announce themselves anymore in the nodes they were announcing but in the new nodes (since they would be in that moment the three closest nodes). Then, **three nodes positioned as the closest to an info hash will be the only entities responsible for the storage of the list of peers after the expiration time**. The space between the info hash and its closest nodes should be large enough to add there millions of nodes as we show in Appendix G.

3.4.2 Consequence

According to the analysis, a set of nodes can control a whole list of peers. This fact gives the possibility to these nodes of performing a bad usage of the list of peers. This fact has some implications which may be harmful for the DHT.

3.4.2.1 Responsibility

A set of nodes controlling a whole list of peers of an object would be the only entity responsible to let other nodes to join the swarm. When a node wants to obtain the peers of an object, it performs a lookup in the DHT. The lookup will always carry to

the nodes containing the list of peers since all lookups over the same key converge along the same path, regardless the original node (as demonstrated in [2]). The node will query the nodes containing the list of peers in order to join the swarm. There is not any documented mechanism to detect suspicious nodes so the querier can not know if those nodes have a random identifier.

3.4.2.2 Censorship

When the nodes controlling a list of peers receive get_peers queries, they may agree to reply an empty or a fake list of peers to some nodes. If they do that, it would be impossible to join the swarm for new nodes wanting to download the object. This would be considered censorship due to the denial of access to the object.

It is important to clarify that a tracker may exist containing also the list of peers for the object, if this is the case, nodes which want to join the swarm can do it using the tracker.

3.4.2.3 DDoS attacks

There is another possible malicious usage of a list of peers. The scenario for this vulnerability is the same than in the problem of censorship. The difference is that it is not necessary to control the whole list of peers, one or more nodes containing a list of peers are necessary. This malicious usage is, again, providing a fake list of peers to the get_peers queries. If this list of peers contains an IP address to attack and the info hash is popular (i.e. it has a lot of peers), it would perform a silent DDoS attack (since the guilty node would not send any message to the victim IP address).

Nodes querying for peers can not know if the address and port provided in the list of peers corresponds to a BitTorrent client. They will send a BitTorrent message to the victim IP address even though it corresponds to a computer without a BitTorrent client.

If the set of nodes is positioned close to a very popular info hash, thousands of queries can be redirected to the victim IP address. It would be hard to identify the origin of the attack because the responsible (the nodes providing a fake list of peers) would not take part directly in the attack.

3.5 Scalability of nodes containing lists of peers

Supposing that all the nodes in the DHT choose their identifier randomly, they should be uniformly distributed in the DHT (as explained in [2]). The info hashes will also be uniformly distributed in the DHT since uniformness is a property hash

functions must ensure by definition. So, a node can not know in advance which lists of peers it is going to contain. In the previous section, we have presented the problem of using incoming queries as a DDoS attack. To do this, an info hash with a large amount of peers is necessary. However, if no nodes are positioned close the info hash to contain its list of peers, these incoming queries will get to the closest nodes in that moment.

3.5.1 Analysis

This problem is another point of view of redirecting queries using the list of peers to perform DDoS attacks. The DDoS attacks were performed when a node contained the list of peers of a popular info hash and responded a fake list of peers to the get_peers queries. In this case, our goal is studying how the incoming and outgoing traffic generated in a node containing the list of peers of a very popular info hash grows according to the number of peers.

The nodes containing the list of peers of a very popular info hash would be a small set of nodes responsible of the storage of a huge list of peers (for example, in The Pirate Bay [8], for every info hash stored there, there is an estimation of peers. We could find info hashes with up to 33000 peers). These huge lists of peers can carry to a problem of scalability. If three nodes are responsible for the storage of a list of peers with 33000 peers, they may not be able to bear the high traffic rate generated.

In nodes containing the list of peers of a popular info hash, the number of announcements and get_peers queries per second can be huge making impossible that the nodes containing the list of peers serve all the requests. The incoming traffic for these nodes could be really high and the outgoing traffic even higher (because find nodes and get_peers responds are pretty bigger than their queries since they have to attach a list of nodes or peers as shown in the format message in [20]).

3.5.2 Consequence

The imbalance generated by this problem would have three important consequences.

3.5.2.1 Equality

Although nodes were able to bear the traffic rate due to the list of peers of a very popular info hash, this problem would break with the property of equality in P2P networks. Nodes close to a very popular info hash would have a traffic rate due to the DHT much higher than the most of the rest of nodes. The DHT is an effort to

index the lists of peers by the equal collaboration of all the nodes, if some nodes have a much higher traffic, the effort is not equally distributed among all the nodes.

Furthermore, a node can not know in advance which lists of peers it is going to contain. This can suppose a high unexpected traffic in that nodes due to an info hash that probably not related with it. We have studied this problem empirically from a node close to a very popular info hash to observe the incoming traffic, the result is presented in Chapter 7.

3.5.2.2 Impact

This problem may be serious in domestic Internet connections, but there are some cases were it can be more critical. Currently, there are some studies of DHT clients in mobile devices. These studies try to consume the less possible energy by having a low traffic rate (like the work presented in [31]). These devices may not to be able to face this problem. Also, the cost of some Internet connection depends on the traffic generated. Users may not realize about the traffic generated by the DHT because it is not something the final users should focus on. This problem could generate a huge unexpected Internet bill.

3.5.2.3 Future

Our studies showed that the problem of scalability is not too critical with the info hashes we found (up to 33000 peers). However, in the future, info hashes with many more peers may exist. For example, the project P2P-Next [34], is a project which intends to use a P2P network to broadcast live transmissions. International events like the Eurovision festival or the Olympic Games, which are followed by millions of people, could imply the existence of info hashes with millions of peers.

Chapter 4

Related work

In this chapter we present some work about BitTorrent and Mainline DHT related with this thesis.

4.1 Profiling work

An important part of this thesis is profiling Mainline DHT. We profile some parts of the DHT by analyzing it and checking the analysis empirically. Profiling helps to understand more in depth how the DHT works looking at its actual behavior. It also helps both to design some experiments and to understand their results.

In the specifications of Mainline DHT, there are two parameters which are not very clear. The design of Kademlia explains that values should be stored in the k closest nodes to a key (k is a free choice parameter) and fixes the expiration time in one hour (but it says it can be modified for optimizations). In the specifications of Mainline DHT k is equal to 8 (as asserted in [20]) and values are lists of peers, so, the eight closest nodes to the info hash should be chosen for the announcements. But, Crosby and Wallach [7], which analyze both Mainline DHT and Azureus DHT, say explicitly that nodes in Mainline DHT announce in the three closest nodes to the info hash. Crosby and Wallach [7] also assert that the frequency of announcements is 30 minutes. The expiration time fits with the design of Kademlia since in explains that it can be modified. However, both the definitions in the number of nodes chosen for the announcements are inconsistent. Nevertheless, the value of these parameters does not affect too much the rest of the analysis we made.

Existing profiling works may help to design experiments and to do some estimations. For example, as we have explained in this section, [7] is the base for some of our hypothesis. This document also gives a global profiling of different aspects of Mainline DHT like an estimation of the total number of nodes. Although this is a quite old data, it has been useful to calculate the approximate occupation of

the identifier space that we present in Chapter 3 (although since the estimation the number of nodes may have changed). In [15], some aspects of Azureus DHT are profiled, this DHT has many properties in common with Mainline DHT, some measurements in Azureus DHT may have a similar value in Mainline DHT.

Yu and Fang et al. [16] study the identifier distribution in KAD DHT (the DHT of eMule [17]). KAD is also a DHT based on Kademlia and its node identifiers are also supposed to be random, any hypothesis about KAD can be interesting to check in Mainline DHT since they are similar DHTs. This work is very related with this thesis since the identifier distribution is an important part of our work. In the work of Yu and Fang et al. [16] it is interesting to watch that the repetition of node identifiers is higher than it should. It could be interesting to study the repetition of identifiers in Mainline DHT, it could influence in the analysis of nodes distribution. In general, the knowledge of other DHTs can help to find solutions to problems and arise questions that could be interesting to investigate in Mainline DHT.

4.2 Vulnerabilities

The analysis of the possible vulnerabilities we present in Chapter 3 is mainly deducted from [2] and supported by [20] and [7]. Although, as said before, they have inconsistent definitions of some parameters, both of them still support our analysis. The differences they have only change the resources necessary to carry out the vulnerabilities, and this difference is not very relevant. In these documents there is not any documented mechanism to prevent the vulnerabilities we study. However, it could exist any implicit mechanism to prevent them.

The most of the work about vulnerabilities in this thesis is possible due to the possibility of choosing an identifier in Mainline DHT. In the specifications of Mainline DHT there is no any mention to this possibility, it is just said that node identifiers should be random. Also, it is not possible to know if a single identifier is random or not.

Azureus DHT is the other DHT of BitTorrent based on Kademlia. Even though it is not used in this thesis, its policies about node identifiers are very interesting in the context of this thesis. In [24] it is explained how node identifiers in Azureus DHT are a hash of the IP address of the node. This provides a mechanism to find nodes an identifier not calculates as a hash of the IP address. This mechanism gave us the idea to study what the implications of choosing an identifier in Mainline DHT are since non random identifiers can not be detected. It should be studied if there is any reason why Mainline DHT chooses random identifiers.

4.3 Documented vulnerabilities

One of the goals of this thesis is studying some possible vulnerabilities of Mainline DHT. Mainline DHT is something recent inside BitTorrent. The BitTorrent protocol has several documented vulnerabilities like those presented in [11], where they show some vulnerabilities of the protocol, [12], where they use the protocol to perform DDoS attacks, [13], where they exploit the protocol to achieve higher download rates and [14], where they show another exploit to get higher rates without contributing by uploading.

For Mainline DHT, there are not many documents studying vulnerabilities. Any document studying a vulnerability (either for BitTorrent, for Mainline DHT or for another DHT) is useful in order to see how a vulnerability has to be analyzed and tested. Also by reading about vulnerabilities in other DHTs we can wonder if it would possible in Mainline DHT.

Chapter 5

Methodology

This chapter presents and justifies the decisions we made for the development of the tools and the design of the experiments. Performing the experiments could be achieved by different ways, we had to make some decisions for their execution. For each decision, we evaluate the different alternatives and we try to choose the most simple and efficient. We also identify the implications the methodology may have. This chapter is complemented with Chapter 6, which motivates the development of the tools and explains their usage.

5.1 Choice of technologies and resources

In Chapter 3 we present a theoretical analysis about Mainline DHT. We needed to check experimentally this analysis throughout some experiments. These experiments required a way to interact with Mainline DHT. We couldn't find any tool which offered the functionalities we were looking for (we explain these requirements in Chapter 6).

We decided to develop a set of tools. We did it using a Python [33] library which implements a Mainline DHT client called kadtracker provided by the TSLab department of KTH. Another choice to develop the tools was modifying any existing BitTorrent client since a lot of clients are Open Source. The reasons why we chose kadtracker are mainly two:

1. Kadtracker is implemented in Python. BitTorrent was originally designed to be implemented in Python, aspects like the message format (described in [20]) are described using native Python data structures (as lists, dictionaries, etc). Using Python as programming language makes easier to understand and to implement the protocol.

2. Kadtracker is a standalone library. This fact makes easier to modify it and adapt it as a tool. If we adapted another BitTorrent client for our purposes, we should locate the specific part of code necessary to modify among all the functionalities.

Kadtracker uses the version 2.5 of Python, so, a the version 2.5 or a later version is needed to run it properly. In the development of our tools we did not add any functionality which needed a more recent version of Python.

Some experiments required the usage of a large set of nodes (up to about 30 nodes). As nodes in the DHT may be identified by their IP address, for this kind of experiments it was not enough to run the set of nodes in one computer. We needed a way to run several nodes behind different IP addresses. We have used the network Planetlab [3] to run this kind of experiments. Planetlab is a set of computers located all around the world used to perform experiments using up to hundreds of machines at the same time.

5.1.1 Planetlab

Planetlab is a set of computers (called nodes) located all around the world used for investigation purposes. We got access to it throughout KTH. We had access to hundreds of computers but we did not use more than 30 in any case.

All the nodes in Planetlab use a Unix based operating system. Furthermore, all of them have the version 2.5 or later of Python, which is the one needed to run out tools. Nodes in Planetlab have been chosen trying to have them distributed around the world.

Nodes in Planetlab are supposed to stay alive during the experiments, but we have not monitored if they face some problems during the execution of the experiments like connectivity problems or system reboots.

5.2 Kadtracker

Kadtracker is a standalone library which implements a Mainline DHT client. It offers two possible behaviors for a node in the DHT:

- Passive mode: the node stays in the DHT and responds the queries it receives.
- Active mode: the node performs lookups. Given an info hash, the node gets its list of peers by finding those nodes containing it (in case they exist).

Every execution of a node must include the passive mode even if the node has also an active behavior. Every node has a responsibility in the storage of lists of

peers and routing, this tasks must not be avoided. If they were avoided perhaps the results obtained could be altered due to the lack of participation in the DHT. In general, we tried to influence as less as possible in the normal behavior of the DHT in order not to alter the results of the experiments.

5.3 Experiments

In order to check experimentally the analysis presented in Chapter 3, we had to carry out some experiments. To do these experiments we developed a set of tools modifying kadtracker. Once the tools were ready we executed the experiments. All the experiments have been performed using Planetlab nodes.

5.3.1 Limitation in the list of peers

Something which may affect the results of the experiments is the limitation of the list of peers to 50 peers as maximum. We discovered that a list of peers with more than 50 peers does not fit in a UDP packet. Many nodes we observed in the experiments also limit their list of peers to 50 peers. We modified kadtracker so when a list of peers grows up to more than 50 peers, the 50 most recent announcements are returned. This policy intends to keep in the list of peers the highest amount of alive nodes. Other policies could have been used like returning random peers.

We found some nodes returning list of peers with more than 50 peers. This implies an IP packet fragmentation and many times we observed that the packets were not received properly. We do not know if the decision of limiting the list of peers may affect somehow to the behavior of the nodes and so to the results of the experiments.

5.3.2 Types of experiments

The experiments can be classified into three types:

- Active experiments: are those where we generate lookup queries over an info hash. We use an active node to execute them.
- Passive experiments: are those where we positioned a node in the DHT and we analyze the incoming queries it gets. We use a passive node to execute them.
- Active-passive experiments: are a combination of the other two kinds. We position a passive node in the DHT to analyze the incoming queries and an active node to send queries to the other.

In some passive experiments we position a node as the closest to an info hash. They are supposed to be the closest to the info hash all the time during the experiment, however, we did not check it explicitly. Nodes in the DHT are supposed to choose their identifier randomly. The probability that a node with a random identifier is closer to the info hash than our nodes is very reduced according to the results we obtained in a experiment where we measured the distance between info hashes and their closest nodes.

The passive experiments get an observation of an altered scenario, this means that the data is collected by taking part in the scenario and not just by observing it. Moreover, the nodes used for this experiment chose their own identifier. However, there was not any other possible choice to watch the behavior of a node containing a given list of peers. Any BitTorrent client with a random identifier may contain a list of peers but to contain the list of peers we wanted we had to choose the identifier.

Some of the active experiments which collect data from the DHT perform 10 lookups (1 every 5 seconds) over each info hash of a large set of info hashes. This strategy may seem an aggressive way to obtain the information because a single lookup could be enough. The reason to use this strategy is trying to give time to all the nodes to respond and finding all the possible nodes containing the list of peers. Some experiments did not find all the nodes containing list of peers in the first lookup, sometimes more than one lookup was necessary. In our active experiments it is important to find all (or at least, the most of) the nodes containing list of peers. It is still possible that in any case we did not find all the nodes containing list of peers.

5.3.3 Validity of experiments

When we tested the problem of censorship, no IP addresses were censored. It consisted on trying to control a whole list of peers. Perhaps, if we had censored the access to the list of peers to all the querying nodes, the result would have been different.

The reason why we did not carry out censorship is that we were not able to control a whole list of peers in any case. In case we had been able, we would have narrowed in the problem studying different scenarios of the problem and different usages of controlling a list of peers.

A similar case is the case of DDoS attacks. In this case we have demonstrated that it can be serious since the incoming traffic rate in a node containing list of peers may become huge, however, we did not provide fake lists of peers to perform a DDoS attack. We do not know if the nodes asking for peers would have performed the DDoS attacks but we do not know any reason why they would not since they can not know in advance if a list of peers is fake or not.

5.4 Information management

Most of the experiments use two tools to obtain the information: a Mainline DHT client modified to log information and a displayer to process the logged data. None of the modifications of the clients to log information implies a functional modification. The steps to carry out this kind of experiments are the following (also shown in Figure 5.1):

1. The modified client generates an output containing the logged data we want to process.
2. A displayer program parses the log file, obtains the results and shows them graphically or summarized.

Figure 5.1. Schema of how the tools manage the information.

The reasons why the experiments are carried out using two programs and a log file instead of doing everything in a single program are the following:

- Maximizing the independence of the DHT client with the experiment.
- Some experiments require a lot of time (up to a couple of days). If there is any problem why a test should stop, it is preferable to have some results stored instead of losing everything and running the experiment again.
- In case of finding an error in a displayer program, this mechanism avoids the necessity of doing the experiment again.
- Interest on keeping the log file because, in the future, from the same experiment, more information may be extracted, so, in that case, the experiment should not be repeated again.
- The displayer programs can be used as a back end for logs generated using other DHTs based on Kademlia.

5.5 The log distance metric

As we explained in Chapter 2, distance between an info hash and a node identifier is measured as the bitwise XOR of both 160-bit sequences. As distances can be very

large numbers, since they belong to the interval $[0, 2^{160})$, in this thesis distances are measured using log distance.

XOR distance is the result of the bitwise XOR of two 160-bit sequences. Log distance performs the same operation, however, the result is not the number itself but the position (counting from the right and starting from zero) of the first bit which is 1 (counting from the left). For example, if the distance between two sequences is in the interval $[2^{30},2^{31})$, the log distance will be 30. We show some examples of log distance using 8-bit identifiers in Figure 5.2.

Using log distance, distances will always belong to the interval $[-1, 159]$ which is much easier to manage than distances in the interval $[0, 2^{160})$ obtained using XOR distance. We consider the special case of log distance equal to -1 when the XOR distance is equal to 0. In later chapters, every time we refer to distance we mean log distance.

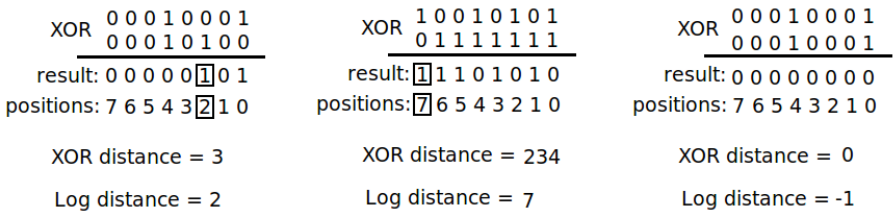

Figure 5.2. Examples of log distance using 8-bit identifiers.

Chapter 6

Tools development

In this chapter we present the development of the tools and we give an overview of their usage. All our tools have been developed using the standalone library kadtracker. The main reason to use kadtracker is that we had direct contact with the developers, however, there are other reasons why we have decided to use it. In this chapter we present all the reasons why we have used kadtracker instead of other choices.

6.1 Requirement of active tools

This thesis consists on an experimental view of a part of Mainline DHT based on the analysis we present in Chapter 3. Part of the analysis we presented consists on studying different data obtained in a lookup in Mainline DHT. By analyzing this data, we can study the behavior of a set of nodes getting a global view of a part of the DHT.

6.1.1 Choices

An important part of the data to analyze in the experiments can be obtained by looking up an info hash in the DHT. Any Mainline DHT client could perform a lookup over an info hash. Some other experiments required lookups as well but also other information like the identifiers of nodes providing list of peers or their distance to the info hash.

Using a Mainline DHT client to obtain this data was not easy since they do not usually provide the data we needed in the user interface. Furthermore, in this thesis, watching the data was not enough, we had to process. So, a Mainline DHT client had to be modified again in order to log this data.

6.1.2 Final choice

Kadtracker offered a very easy interface to do this operation and it lacks of a user interface since it is a standalone library. Getting and processing the data we needed by modifying another client would have been a harder task.

6.2 Requirement of passive tools

This thesis claims to check some possible vulnerabilities in Mainline DHT. The first requirement to test the vulnerabilities was collecting data from a node close to an info hash, this is a common point in all of them. Also, some experiment of profiling needs to pick up information from a node containing list of peers.

From the existing closest nodes to an info hash we could collect some data but there is some other data that we could not. The total traffic generated in this kind of nodes could not be studied unless they were ours. Also, we could not choose how far the closest nodes to the info hash were. We had to study the behavior of nodes from different distances to the info hash, it would be impossible to do that without adding new nodes in the DHT. Then, for our experiments, we had to add new nodes to the DHT.

6.2.1 Choices

We could have used any Mainline DHT client to position a node in the DHT, but clients choose a random identifier, so its position in the DHT would be random. It would have made impossible to choose an info hash in order to contain its list of peers. Then, we had to modify any Mainline DHT to choose its identifier.

6.2.2 Final choice

Among all the available Mainline DHT clients to modify, we chose kadtracker for the reasons explained in Chapter 5. We modified it to choose its identifier, but also some data had to be logged from its incoming and outgoing packets. To test the scalability problem, we had to know the number of messages generated and their type. We could have obtained this data by using a tool to capture the network traffic (like Wireshark [19]). We discarded this choice because it would have taken more time than modifying kadtracker.

The decision was modifying kadtracker to log this data itself. It was easy to find the code where the incoming messages were received and the outgoing messages were sent. These parts of the code were modified to log the data we needed (without modifying any functionality).

6.3 Modifying kadtracker

In this thesis, the goal is not changing the functionality of the DHT client (mainly because it has to follow the protocol). Only three modifications have been done in the passive mode:

- Choice of the node identifier. Instead of letting the client to choose its identifier randomly, the option to choose the identifier when the client is started has been added. But, instead of applying for the identifier (which can be quite tricky to calculate), an info hash and the log distance the node has to be from it must be provided. This is an easy way to position nodes as close to an info hash as wanted (otherwise the calculation should be made in advance).
- Censorship list. When a get_peers query is received, the node checks a list with IP addresses contained in a file. If the querier's IP address is contained in this list, an empty list of peers is returned.
- Limit in the size of the list of peers. This change was made when it was observed that, when the list of peers grew up to more than 50 peers, it did not fit in a UDP packet. At the same time it was observed that many clients returned always lists of 50 or less peers. It was decided to limit the list of peers to the last 50 announcements. This patch has been integrated in kadtracker.

The passive node with these modifications has been called *censor*. The active node has not had any functional modification, this node has been called *observer*. Using these clients, in every experiment one or both of them have been modified properly to obtain the wished results. These modifications do not affect to any functionality, they have consisted on logging that data of interest for the experiments.

For the most of the experiments, another complementary tool has been developed to display the information graphically or summarized.

6.4 Set of tools

The most of the experiments use a pair of tools: one to log information (called logger) and one to process the logged information (called displayer). The loggers can be split in two groups: active loggers and passive loggers. Active loggers are those which are a modification of the observer (providing the functionality of an active node). Passive loggers are those which are a modification of the censor (providing the functionality of a passive node).

Appendix H classifies all the tools and summarizes their purposes. Each pair of tools has been used for one experiment. The functionalities a tool needs arose in the definition of the experiments. Experiments are defined in Chapter 7.

6.5 Tool parse_announcements

We developed another tool independent from the others. The purpose of this tool is showing those nodes where a DHT client chooses to announce itself. All the BitTorrent clients have to respect the message format, and their network traffic can be captured in a pcap file (using, for example, Wireshark [19]). The tool parse_announcements processes this pcap files and, for every lookup, shows the distance to the info hash of nodes containing list of peers for it and the distance to the info hash of nodes where the client chooses to announce.

This tool parses a pcap file providing an interface to get the fields of the application level packets. The packet format in the application level is explained in Appendix C.

Chapter 7

Experiments

This chapter presents all the experiments carried out. For each one, we define it, we analyze it and we present its results. Some of them have used a large set of info hashes (how these info hashes were obtained and their characterization is explained in Appendix D). Others have used a set of nodes (the coordination of these nodes is explained in Appendix E).

7.1 Experiment 1 - Number of nodes tracking a given swarm

The number of nodes where a node should announce itself is not clearly defined in the bibliography. This experiment aims to clarify it and document it. The number of nodes containing list of peers of an info hash should be similar to that value since the nodes chosen for the announcement have to be the closest to the info hash.

7.1.1 Expected results

According to [7], when a peer announces itself, it does it in the three closest nodes to the info hash of the object. However, according to the design of Kademlia [2], a node stores a value in the k closest nodes to the key and [20] specifies that in Mainline DHT k=8. In Mainline DHT values are the lists of peers, so, nodes should announce themselves in the 8 closest nodes to the info hash. As in the design of Mainline DHT the number of nodes where a node should announce itself is not specified explicitly, for this experiment we suppose that the value is three nodes, as asserted in [7].

The number of nodes containing list of peers should be similar to the number of nodes chosen for the announcement since the closest nodes to the info hash are chosen for the announcement (both [7] and [2] assert it). Our deduction points out

that **the number of nodes containing list of peers for an info hash is about three in all the cases**.

7.1.2 Experiment definition

An active node has been used for this experiment. For every info hash, the node has performed lookups over it and has counted the number of different nodes that return a list of peers. A node is identified by its IP address and UDP port.

- Number of active nodes: 1.
- Number of info hashes to lookup: 1200.
- Number of lookups over every info hash: 10 (1 every 5 seconds).
- Data to measure: number of nodes containing list of peers for every info hash.

The tools complete_lookup and display_average_number_nodes have been used for this experiment.

7.1.3 Results

The data obtained in the experiment is summarized in Table 7.1 and represented graphically in Figure 7.1. The result has not been as we expected. Even though the most typical case is 5 nodes which is close to 3 and 3 is the second most typical case, there are too many cases with too many nodes containing list of peers. The 40.23% of the observed info hashes have more than 10 nodes containing list of peers. The average number of nodes containing list of peers is 11.8 but the standard deviation is 18.47, which is too high to take into account the average. According to the expected results we show in Chapter 3, the average should be approximately 3 and the standard deviation a small number.

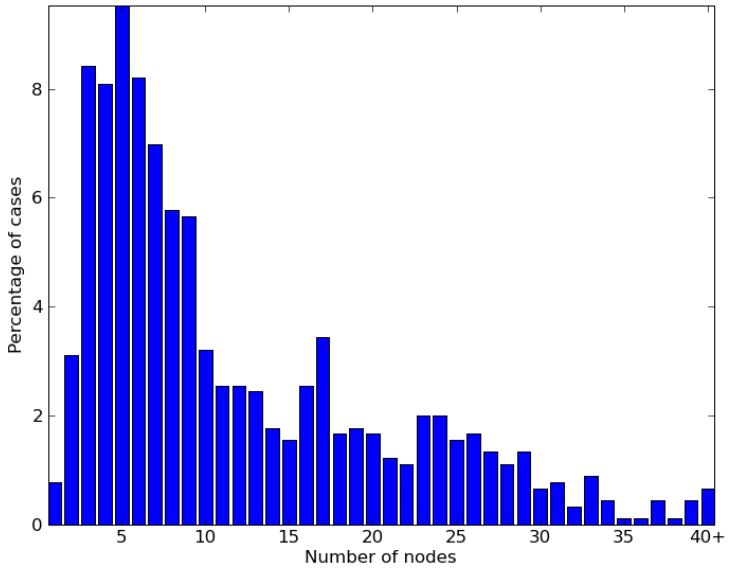

Figure 7.1. Number of nodes containing list of peers.

Number of nodes	Ocurrences	Percentage
1	7	0.78%
2	28	3.10%
3	76	8.43%
4	73	8.09%
5	86	9.53%
6	74	8.20%
7	63	6.98%
8	52	5.76%
9	51	5.65%
10	29	3.22%
11	23	2.55%
12	23	2.55%
13	22	2.44%
14	16	1.77%
15	14	1.55%
16	23	2.55%
17	31	3.44%
18	15	1.66%
19	16	1.77%
20	15	1.66%
21	11	1.22%
22	10	1.11%
23	18	2.00%
24	18	2.00%
25	14	1.55%
26	15	1.66%
27	12	1.33%
28	10	1.11%
29	12	1.33%
30	6	0.67%
31	7	0.78%
32	3	0.33%
33	8	0.89%
34	4	0.44%
35	1	0.11%
36	1	0.11%
37	4	0.44%
38	1	0.11%
39	4	0.44%
40+	6	0.67%

Table 7.1. Number of nodes containing list of peers.

7.2 Experiment 2 - Growth of a list of peers according to the distance of the nodes containing it to the info hash

This experiment intends to corroborate that nodes positioned as the closest to an info hash start having a list of peers of it and containing a big part of the list. This experiment checks whether the result changes depending on the distance or not.

7.2.1 Expected results

The closest nodes to the info hash will receive the largest amount of announcements independently of their distance to the info hash. According to [2] and [20], nodes announce in the closest nodes to the info hash. There is not any mention in the bibliography about any different behavior according to the distance of the nodes to the info hash.

7.2.2 Experiment definition

A set of nodes has been positioned close to an info hash in three different scenarios. In every scenario, one of the nodes was the closest and consecutive pairs of nodes had a distance between them of 2 units.

We did not check our nodes were the closest all the time, however, according to the probabilities we present in Appendix F, the probability for a random node identifier to have distance 0 or -1 to the info hash is 2^{-159}, to have distance 22 or less to the info hash 2^{-137} and to have distance 80 or less to the info hash 2^{-79}. So, it is extremely improbable that a random node is closer to the info hash than our nodes in this experiment.

- Number of passive nodes: 11 in the first scenario, 29 in the second scenario and 31 in the third scenario.
- First scenario: the log distance between the info hash and its the closest node is 0.
- Second scenario: the log distance between the info hash and its the closest node is 22.
- Third scenario: the log distance between the info hash and its the closest node is 80.
- Log distance between consecutive pairs of nodes in every scenario: 2 (if the distance to the info hash of the closest node is x, the distance of the second closest is $x + 2$, the distance of the third closest $x + 3$, etc).

- Number of different info hashes tested: 1. This experiment requires too much time to be performed with a large set of info hashes. It has been chosen a popular info hash to have a large enough set of announcements.

- Time for the experiment: 24 hours in every scenario and another 24 hours between consecutive scenarios.

- Data to measure: number of announcements in every node as well as the versions of the clients announcing (when they provide it).

The tools announcements and display_announcements_distance have been used for this experiment.

7.2.3 Results

The results of this experiment are shown graphically the first scenario in Figure 7.2, the second scenario in Figure 7.3 and the third scenario in Figure 7.4 (the Y axes in the figures has a logarithmic scale). They show the total number of announcements and the versions of the clients announcing.

In these experiments we use the abbreviated name of BitTorrent clients (for example, "UT" is UTorrent). "None" are those clients which do not provide their version (more information about the meaning of these abbreviated names can be found in [21]).

In the scenario 1 and the scenario 3 the closest node is receiving a very small amount of announcements. This behavior should be investigated more in depth. If the closest node is ignored, the closest nodes are those receiving the largest amount of announcements in all the scenarios. This result seems to be caused by UTorrent. The rest of the clients send more or less the same number of announcements to all the nodes.

The nodes receiving the largest amount of announcements are not only the three closest (ignoring the closest in the scenarios 1 and 3), they are more than three in some case. However, our goal is accomplished in this experiment, we intended to show that the number of announcements does not depend on the distance to the info hash of the closest nodes. We can see in all the scenarios that the number of announcements in the closest nodes is similar.

In both the scenario 1 and the scenario 2 some points of the graphic seem to be missing. They are not missing, this result is due to the logarithmic scale. In those points where there graphic seems to be missing its value is 0. As there are not two real numbers x and y such that $x^y = 0$ we omit the value of the graphic in those points.

In the scenario 3, the node with distance 120 to the info hash has a very low number of announcements compared with the rest. We do not know the cause of this behavior, perhaps it is due to some problem in the Planetlab node.

Independently of the versions of the clients announcing themselves, nodes which are not part of the three closest are receiving too many announcements. This behavior could be related with the limitation in the list of peers. A mechanism could exist to detect this and choose other nodes to announce. These unexpected behaviors are investigated in later experiments. This experiment also shows that UTorrent is the most popular BitTorrent client, this is important since the behavior of Mainline DHT is highly influenced by UTorrent.

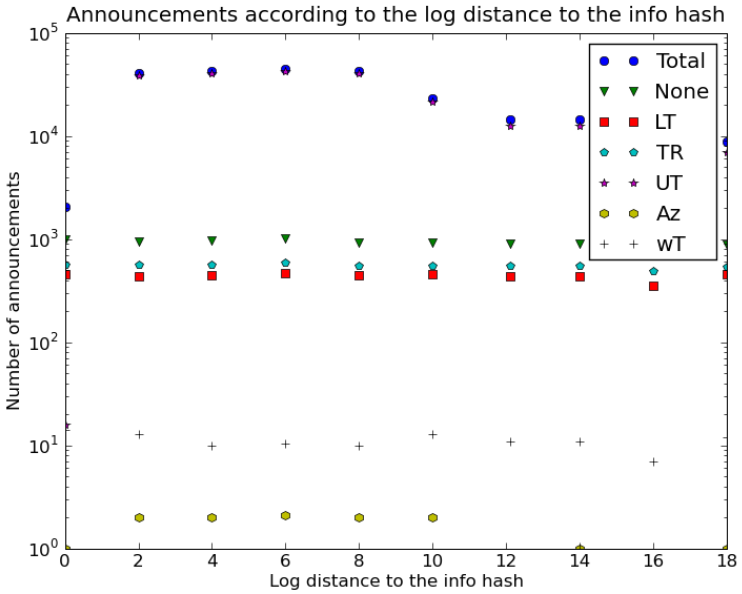

Figure 7.2. Announcements in the first scenario.

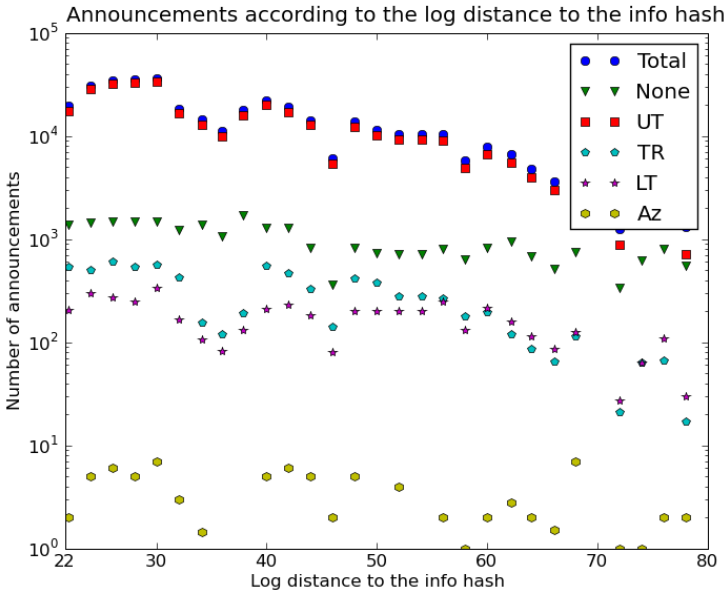

Figure 7.3. Announcements in the second scenario.

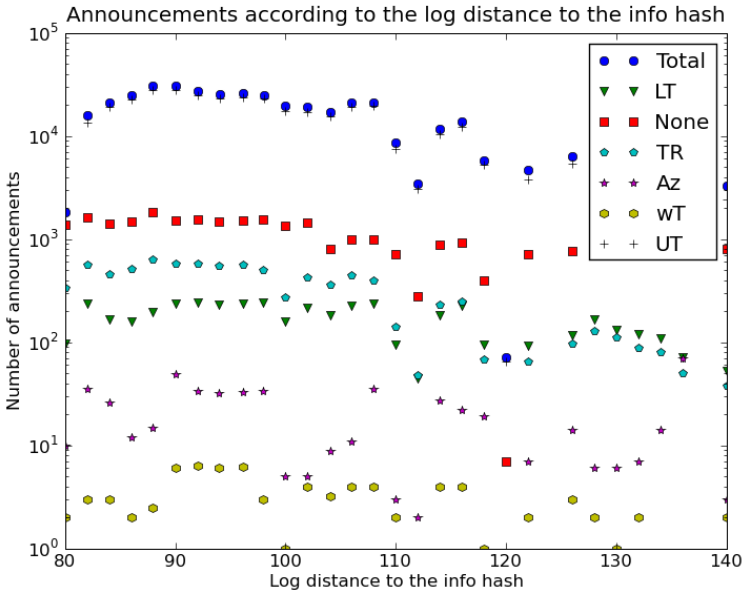

Figure 7.4. Announcements in the third scenario.

7.3 Experiment 3 - Frequency of announcements

This experiment checks if the frequency of the announcements is the expected one. In this experiment a node has been positioned close to a popular info hash. It has tracked the announcements it received to observe the frequency of nodes announcing on it.

Furthermore, this parameter may serve to check if nodes are coordinated. The frequency of announcements is supposed to be the same value than their expiration time. If a node sends an announcement to a node with different frequency to its frequency, it may stay out of the list of peers during some intervals of time. Giving a popular value for the frequency may help to nodes to stay in the list of peers.

7.3.1 Expected results

According to the design of Mainline DHT, **the frequency of an announcement (as well as its expiration time) is 30 minutes** as explained in [7].

7.3.2 Experiment definition

One node has been positioned in passive mode close enough to an info hash to contain its list of peers. It has recorded all the announcements it has received and has tracked those sent by the same node (identifying a node by its IP address and DHT port). From all the announcements sent by the same node, it has calculated the average time between every pair of consecutive announcements.

- Number of passive nodes: 1.
- Number of info hashes: 1 (we have chosen a popular enough info hash to get enough announcements to obtain a reliable result).
- Distance to the info hash: 130 (being the closest one).
- Time for the experiment: 3 hours. We haven chosen a longer time because, if a node sends an announcement, leaves the DHT, joins again afterwards and sends another announcements, we could wrongly think that its frequency is the time the node has left the DHT. We consider the time we have chosen is enough to get a reliable result.
- Data to measure: average frequency of all announcements for every node announcing.

The tools announcements and display_freq_announcements have been used for this experiment.

7.3.3 Results

We have observed a set of 581 different nodes announcing. The result of the experiment is summarized in Table 7.2 and represented graphically in Figure 7.5. A big part of the announcements (the 47,16%) have a frequency between 20 minutes and 35 minutes (which is as expected).

A strange result is the high number of nodes announcing in less than 5 minutes. This behavior is unexpected and we have not found any reason for it, but it is very improbable that this is their actual frequency and expiration time. Also in other experiments we observed that many nodes send several announcements in some seconds, this is probably the cause of this result. An hypothesis about this behavior is that these nodes have restrictions in the incoming network traffic (which can be caused, for example, by a firewall). Firewalls or routers using a NAT (Network Address Translation) have some behavior which limit the incoming traffic in the computer. In this case, nodes sending several announcements may not receive the responses and that would be the reason why they send the query several times, but it should be investigated more in depth.

In this experiment it is not possible to differ a node which stays alive and sends announcements periodically from a node which leaves the DHT and joins again after some time. For example, if there are two nodes announcing, the first node sends an announcement and 30 minutes later sends another one, the second node sends an announcement, leaves the DHT, joins again after 30 minutes and sends another announcement, their behavior will be the same from the point of view of the node where they announce.

For the design of a Mainline DHT client, this result suggests a frequency of announcements of 20 minutes. Even though the expiration time of a queried node is 30 minutes, it will probably refresh the list of peers with the announcement so the querier will stay on it all the time.

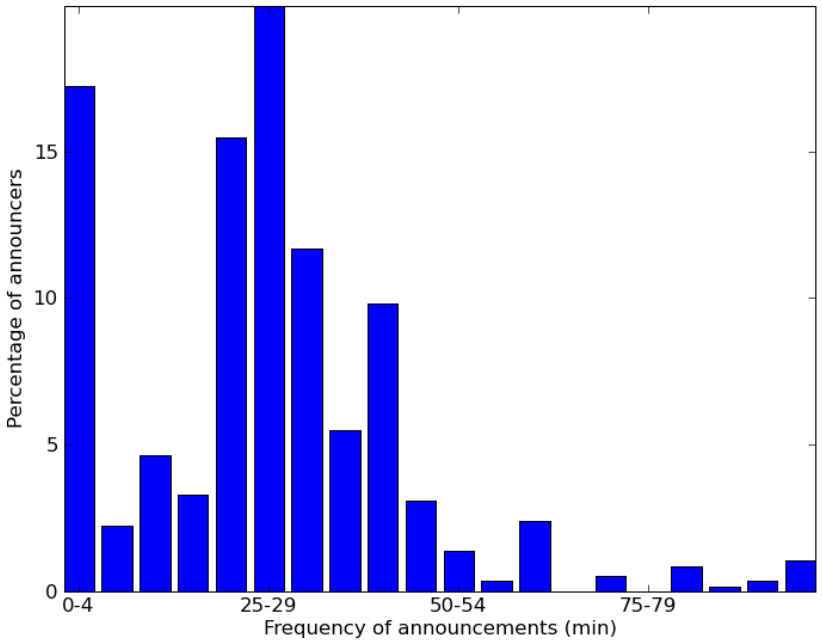

Figure 7.5. Frequency of announcements.

Frequency (min)	Number of nodes
0 - 4	100 (17.21%)
5 - 9	13 (2.24%)
10 - 14	27 (4.65%)
15 - 19	19 (3.27%)
20 - 24	90 (15.49%)
25 - 29	116 (19.97%)
30 - 34	68 (11.70%)
35 - 39	32 (5.51%)
40 - 44	57 (9.81%)
45 - 49	18 (3.10%)
50 - 54	8 (1.38%)
55 - 59	2 (0.34%)
60 - 64	14 (2.41%)
65 - 69	0 (0.00%)
70 - 74	3 (0.52%)
75 - 79	0 (0.00%)
80 - 84	5 (0.86%)
85 - 89	1 (0.17%)
90 - 94	2 (0.34%)
95 +	6 (1.03%)

Table 7.2. Frequency of announcements.

7.4 Experiment 4 - Attempt of controlling a whole list of peers

This experiment checks the problem of censorship. A set of passive nodes has been positioned closer to an info hash than those containing its lists of peers. After some time, these nodes should take the total control of the list of peers. Besides, an active node has been enabled to monitor the evolution of the passive nodes and check if they get the total control of the list of peers.

We did not check if during the experiment the nodes enabled for it were all the time the closest to the info hash, however, according to the probabilities we present in Appendix F, the probability for a random node identifier to have 130 or less of distance to the info hash is 2^{-29}.

7.4.1 Expected results

When an announcement is received, after its expiration time, the peer which sent it will be removed from the list of peers. In order to stay on the list of peers the peer will have to send it to a set of the closest nodes to the info hash periodically, as explained in [7]. The experiment 3 shows that after one hour, all the current entries in a list of peers will probably have expired. The experiment 1 shows that the 82% of the info hashes have 20 or less nodes containing list of peers. The experiment 2 shows that, if a set of nodes are the closest to an info hash, independently of their distance, they will receive the largest amount of announcements. So, **if 23 nodes join the DHT as the closest to an info hash and they stay alive, after less than 24 hours, they will be the only nodes containing the list of peers of the info hash**.

7.4.2 Experiment definition

- Number of passive nodes: 23.
- Distance to the info hash of the passive nodes: from 107 to 130. Each node has been positioned in a different distance in that interval.
- Estimation of peers for the info hash: 671.
- Number of active nodes: 1.
- Frequency of lookups by the active node: 2 minutes.
- Time for the experiment: 24 hours.
- Number of info hashes: we have tried several times to control an info hash and it has not been possible in any case. For this reason, this experiment shows a typical case of how it has not been possible to control an info hash.

- Data to measure: nodes containing list of peers for the info hash and the size of their list.

The tool censor has been used for the passive nodes and the tools evolution and display_evolution for the active node to watch the passive nodes.

7.4.3 Results

Controlling a whole list of peers of any info hash has not been possible in any case. Even though 23 nodes have been positioned as the closest to the info hash and they have stayed alive for 24 hours, new nodes joined all the time. Our nodes, the nodes trying to control the info hash, have controlled the most of the list of peers, Figure 7.6 shows it. In that figure, red circles are our nodes and blue squares are the rest. The size of the list of peers of nodes which are not ours always tends to be small whereas our nodes control the most of the list of peers peers. Our nodes never have a list of peers with more than 50 nodes due to the limitation established to make the list of peers fit in a UDP packet.

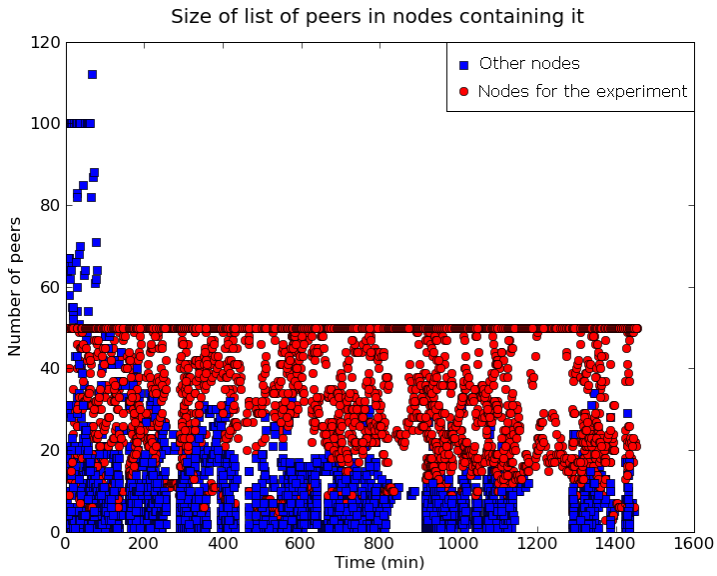

Figure 7.6. Size of list of peers in nodes containing it. Red circles are our nodes enabled for this experiment and blue squares lines are the rest.

A clearer way to observe the result is by watching the number of nodes containing list of peers independently of the size of list of peers. This view shows the addition of new nodes when they are not supposed to join. Figure 7.7 shows the number of nodes in every moment during the experiment. In this case, the red circles are the number of nodes positioned for this experiment (our nodes) and the blue squares

are the total number of nodes. There are two intervals of half an hour where the control of the list of peers is almost total and another one where it is completely total, however, there are always new nodes joining. According to the our expected results new nodes should not join. This fact is explained in later experiments.

The number of the rest of nodes increases and decreases several times (there is even an interval of time when they are zero). This proves that the existence of other nodes containing list of peers is not due to a high expiration time or a lack of expiration of peers in their list. The reason is probably that peers send announcements to nodes which are not the closest.

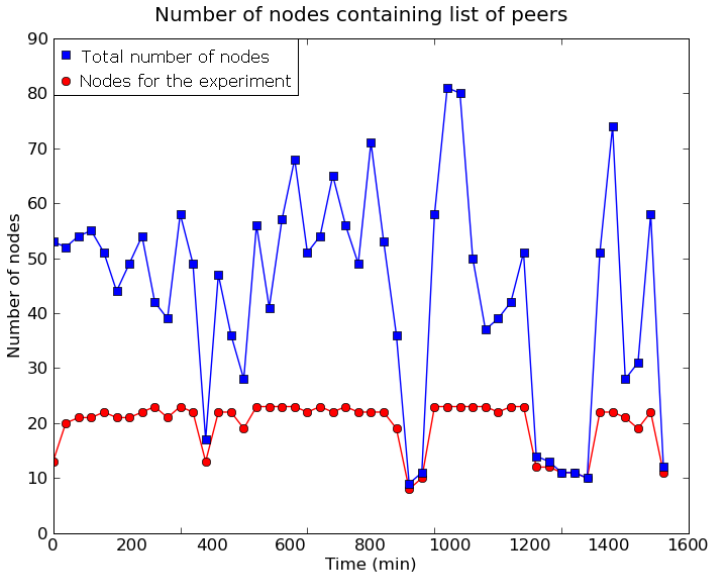

Figure 7.7. Number of nodes. The red circles are the number of nodes enabled for this experiment and the blue squares are the total.

7.5 Experiment 5 - Announcements from announcer's point of view

Due to the results obtained in the experiment 4 (where it has not been possible to get the total control of a list of peers in any case), this experiment seeks the reason in the behavior of some of the most popular BitTorrent clients. This experiment studies the distance to the info hash of nodes where some clients choose to announce. The results obtained in the experiment 4 are probably due to some clients which do not respect the design rules. This experiment tries to identify those clients misusing the network.

7.5.1 Expected results

Every DHT client should announce itself in a set of the closest nodes to the info hash (three nodes according to [7] and eight nodes according to [20]).

7.5.2 Experiment definition

- BitTorrent clients used: UTorrent, KTorrent and BitSpirit.
- Info hashes used per client: 8.
- Data to measure: distance to the info hash of nodes containing list of peers as well as of those chosen for the announcement.

The tool parse_announcements has been used for this experiment.

7.5.3 Results

The results of this experiment are presented in Table 7.3, Table 7.4 and Table 7.5 (the info hashes used are different in every client). They are not enough to extract a pattern of how these clients choose the nodes to announce but it is possible to assert two facts:

- There are clients which are not announcing in three nodes as they should (according to the specifications). In the most of the cases UTorrent announces itself in three nodes but BitSpirit and KTorrent announce themselves in more than three nodes the most of the times.
- There are clients which do not announce themselves only in the closest nodes. UTorrent seems to announce itself in the three closest nodes. BitSpirit announces itself in more than three nodes but those nodes where it announces

itself might be the closest to the info hash. It is almost sure that KTorrent is not choosing the closest nodes to the info hash to announce itself. Even though it announces itself in some nodes which seem to be the closest, it also announces itself in some nodes far from the info hash (later experiments show that it is almost impossible that those nodes KTorrent chooses for the announcement are the closest). In some cases it announces itself in nodes with distance 155 or 152 to the info hash, it is almost impossible that do not exist closer nodes. The nodes KTorrent chooses for the announcements could be those providing nodes in the lookup, but this is not sure.

In the results obtained using UTorrent we can observe one case where it sends 5 announcements, we do not know the reason of this behavior.

It would be interesting to do a complete analysis of the behavior of the most popular clients using the DHT and describing how they actually behave. The results we have obtained are just an approach of their actual behavior.

Info hash	Distance of nodes containing list of peers	Distance of nodes where it announces
IH1	134, 138, 139, 139	138, 138, 139
IH2	136, 137, 139, 139, 140	136, 137, 139
IH3	131, 137, 138, 138, 138	136, 137, 139
IH4	136, 138, 139	139, 140, 140
IH5	135, 139, 140	135, 135, 139
IH6	138, 138, 138	138, 138, 138, 139, 140
IH7	140, 140	138, 139, 140
IH8	138, 138, 139, 140, 140, 141, 141	138, 138, 139

Table 7.3. Nodes where UTorrent announces itself.

Info hash	Distance of nodes containing list of peers	Distance of nodes where it announces
IH1	137, 138, 139, 139	136, 138, 139, 139, 139, 139
IH2	132, 137, 137, 138, 139, 139, 139, 139, 140	132, 137, 137, 138, 139, 139, 139, 139, 140
IH3	(no nodes)	134, 136, 138, 139, 141
IH4	134, 137, 138, 138, 138, 138	134, 137, 138, 138, 139, 139, 140
IH5	138, 140, 140, 140, 140, 140, 140, 141, 141, 141, 141	138, 140, 140, 140, 140, 141, 141
IH6	137, 138, 138, 138, 139, 139, 139, 139, 140	138, 138, 138, 139, 139, 139
IH7	138, 139	138, 139, 139, 139, 140, 140, 140, 140, 140
IH8	137, 137, 137, 137, 137, 137, 138, 139, 140	136, 137, 137, 137, 137, 138, 139

Table 7.4. Nodes where BitSpirit announces itself.

Info hash	Distance of nodes containing list of peers	Distance of nodes where it announces
IH1	147, 149, 149, 149, 150, 150, 151, 151	147, 149, 149, 149, 150
IH2	144, 144, 147, 149, 149, 150, 150, 152, 155	144, 144, 149, 149, 150, 152
IH3	142, 145, 146, 146, 146, 146, 147, 147, 148	142, 146, 146, 146, 147, 147
IH4	147, 149, 151, 151, 152, 152, 152, 152, 153	149, 151, 151, 152, 152, 152
IH5	144, 146, 147, 147, 147, 147, 147, 148, 148, 149, 150, 150, 150, 150, 151, 152	146, 147, 147, 147, 147, 148, 149, 150, 150, 150, 152
IH6	146, 147, 148, 148, 149, 151, 152, 155	146, 147, 149, 151, 152, 155
IH7	137, 141, 142, 146	137, 142, 146
IH8	143, 145, 148, 148, 148, 148, 149, 149, 150	143, 145, 148, 148, 148, 149, 149, 150

Table 7.5. Nodes where KTorrent announces itself.

7.6 Experiment 6 - Distance to the info hash of nodes containing list of peers

The goal of this experiment is getting data about the distance to the info hash of nodes containing list of peers. We have performed lookups over a large set of info hashes. This experiment has helped to observe that there are many nodes containing list of peers which are not (or at least it is very improbable that they are) the closest to the info hash. It studies how the results obtained in the experiment 5 affect the distance to the info hash of nodes containing list of peers. It also provides some data to detect suspicious situations where nodes containing list of peers are too close to the info hash.

7.6.1 Goal

This experiment tries to obtain data about the distance of nodes containing list of peers to the info hash. We ca not give a complete estimation of how the distribution of nodes should be since they are not always the closest, as proved in the experiment 5.

7.6.2 Expected results

We do not expect to find node identifiers closer than 130 to the info hash. According to the probabilities we present in Appendix F, the probability for a random node identifier to be closer to the info hash than 130 is 2^{-30}.

7.6.3 Experiment definition

- Number of info hashes to lookup: 1200.
- Number of active nodes: 1.
- Number of lookups over every info hash: 10 (1 every 5 seconds).
- Data to obtain: number of nodes containing list of peers for every info hash.

The tools distribution and display_distribution have been used for this experiment.

7.6.4 Results

The result of this experiment is represented graphically in Figure 7.8. The distribution of nodes seems to be a normal distribution, however, we do not have enough

data to corroborate this. Also, the number of nodes of distance higher than the average is too high. The reason may be those results obtained in the experiment 5 which prove that some clients do not announce themselves in the closest nodes. As well, it is very probable that in the lookups we did not find all the nodes containing list of peers. The nodes containing list of peers far from the info hash have been found casually following a particular path to the info hash. To find all the nodes containing list of peers we should lookup all the nodes in the DHT. We observed a total of 10768 nodes in this experiment.

With these results, we can consider suspicious nodes closer to the info hash than 130. Also, watching at the distribution, we can detect suspicious situations. For example, a situation of nodes containing list of peers at distances 135, 137, 140, 140, 141, 141 could be normal, but if these distances were 130, 130, 131, 131, 131, 132 it would be very suspicious since it would not fit with the distribution.

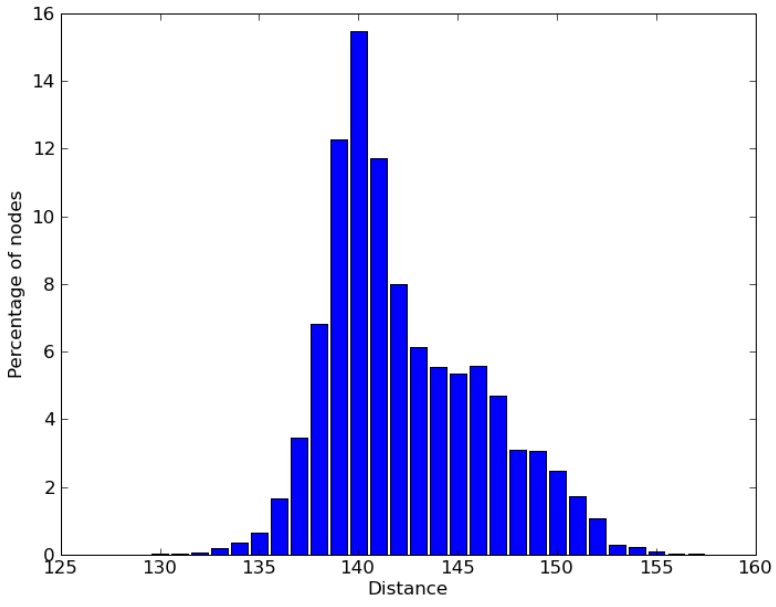

Figure 7.8. Distance of nodes containing list of peers to their info hash.

7.7 Experiment 7 - Percentage of the list of peers contained in nodes according to their distance order to the info hash

This experiment aims to see how the lists of peers are distributed among the nodes. It observes the percentage of the list of peers contained in every node according to its distance order to the info hash (for example, what percentage of peers has the closest, the second closest, etc). It is not expected that they add up to 100% because every node announces itself several times. It studies how the results obtained in the experiment 5 affect to the distribution of the list of peers.

7.7.1 Expected results

The experiment 5 shows that not all the nodes follow the rules for the design. But the three nodes to the info hash almost always get the announcements (this is also observed in the experiment 2). So, **each one of the three closest nodes to the info hash should contain the 100% of the list of peers**. In the case the three closest nodes are limiting the size of the list of peers to make it fit in a UDP packet, it could be less. This experiment also checks how many peers contain those nodes which are not part of the closest because they should not contain list of peers.

7.7.2 Experiment definition

- Number of info hashes to lookup: 1200.
- Number of active nodes: 1.
- Number of lookups over every info hash: 10 (1 every 5 seconds).
- Data to obtain: percentage of peers contained in every list of peers according to the distance of the node to the info hash.

The tools complete_lookup and display_list_distribution have been used for this experiment.

7.7.3 Results

Due to the possible existence of limitation in the lists of peers, we have divided the data of this experiment in three scenarios: a general view (Figure 7.9), info hashes with 50 or less peers (Figure 7.10) and info hashes with more than 50 peers (Figure 7.11). The general view is based in an observation of a total of 54766 peers contained in the lists of peers.

Even though the most of the lists of peers are contained in the three closest nodes in all the scenarios, there is an important part of it contained nodes which are not the closest. In the graphic of lists of peers with 50 or less peers the peers are more contained in the three closest nodes than in the other scenarios. In the scenario of info hashes with more than 50 peers, probably the complete list of peers of the closest nodes is not obtained because they may be limiting it. Anyway, there are still some nodes containing list of peers far from the info hash, but they do not contain a high percentage of peers. The percentage of peers seems to decrease exponentially according to the closeness order of the nodes.

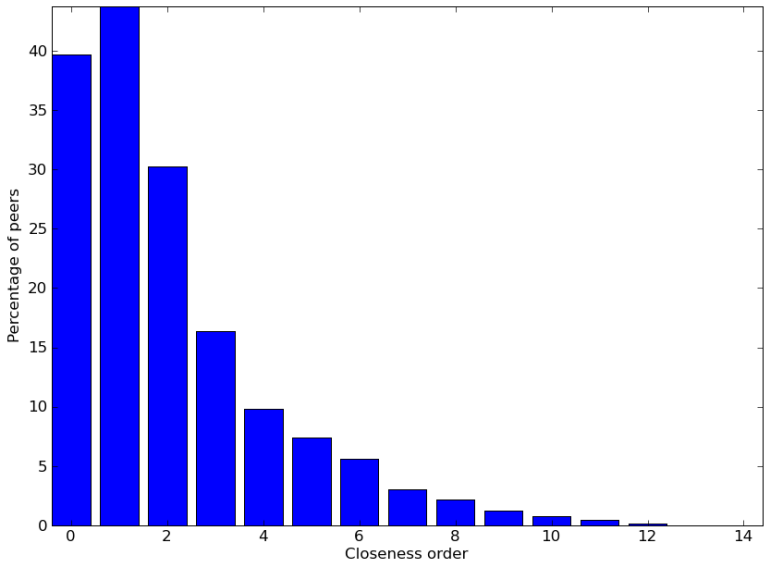

Figure 7.9. Distribution of the list of peers according to the closeness to the closest node to the info hash.

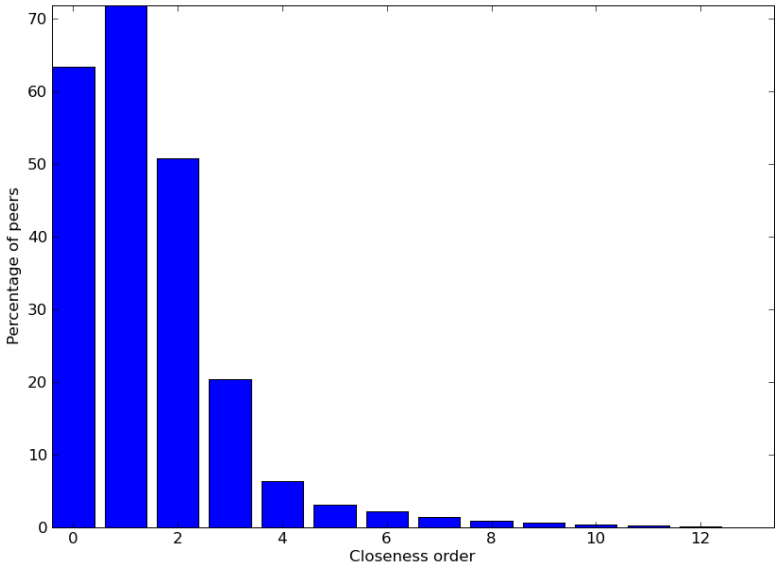

Figure 7.10. Distribution of the list of peers according to the closeness to the closest node to the info hash for info hashes with 50 or less peers.

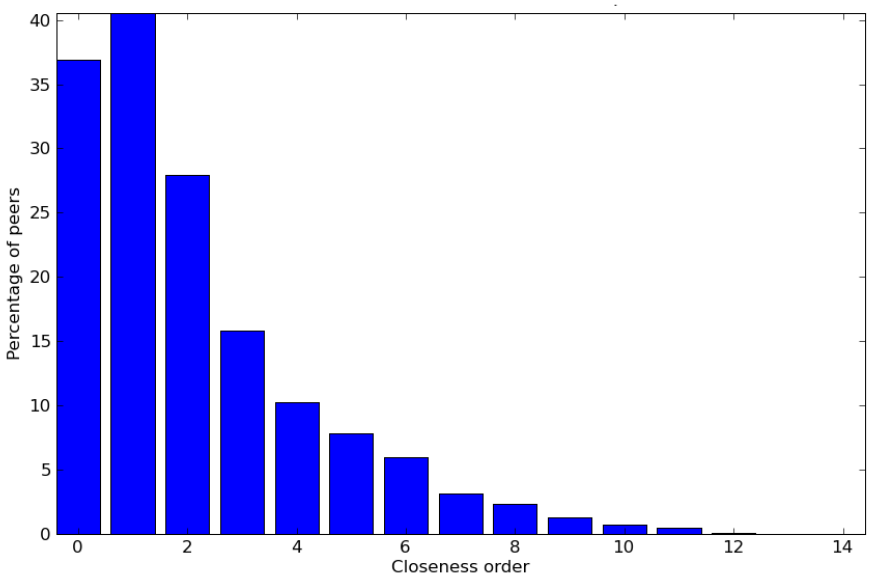

Figure 7.11. Distribution of the list of peers according to the closeness to the closest node to the info hash for info hashes with more than 50 peers.

7.8 Experiment 8 - Number of messages according to the number of peers

This experiment analyzes the scalability problem we presented in Chapter 2. In a very popular info hash, the set of the three closest nodes to the info hash may receive a huge number of messages per second generating too much traffic. This is a scalability problem because these nodes would have more overload than the rest in the DHT. It would also break with the principle of all nodes having the same responsibilities in the DHT. This experiment studies how serious this problem is and also if it may become more serious than it is.

We have positioned a node as the closest to a popular info hash to study this problem.

7.8.1 Goal

This experiment intends to quantify the traffic in a node containing the list of peers of a popular info hash. As peers have to announce periodically (as defined in [7]), the more popular an info hash is, the more announcements the nodes containing its list of peers will receive. Also, these nodes will receive a similar amount of get_peers queries. This experiment intends to quantify this traffic.

7.8.2 Experiment definition

- Info hashes to lookup: one with 107 peers, one with 536 peers, one with 1083 peers, one with 2287 peers, one with 4001 peers, one with 6076 peers, one with 11228 peers, one with 14760 peers and one with 33239 peers. To know the number of peers it has been used the estimation of peers in the torrents of The Pirate Bay.
- Number of passive nodes: 1 for each info hash.
- Time for the experiment: 11 hours and 30 minutes for every info hash starting with all of them at the same time.
- Data to measure: number of messages received of each type.

The tools incoming_traffic and incoming_traffic_statistics have been used for this experiment.

7.8.3 Results

The result of the experiment is summarized in Table 7.6 and Table 7.7. Figure 7.12 shows the growth of the total number of messages and the number of messages in

the minute with the highest incoming traffic according to the number of peers. In the case of info hashes with few peers, the traffic is not very high. For the info hash with 33239 the traffic is very high. The number of ping and find nodes messages does not seem to vary according to the number of peers, those making the difference are the get_peers and announcement messages.

If we focus on the minute with the maximum traffic and we watch the announcements and the get_peers messages in that minute for the info hash with the largest number of peers (2924 get_peers and 1320 announcements), knowing that an incoming get_peers message is 94 Bytes and and announcement message is 141 Bytes (in the application level, as it can be deducted from [20]), the incoming traffic rate is 7.5 KB/s which is not very high, however, the outgoing traffic is higher.

If we just take into account the get_peers messages for the outgoing traffic and just count the size of the list of peers (without the application level header), knowing that every peer is an IP address and a port number (in total 8 Bytes), and with the size of the list of peers limited to 50 (obviously, in this case it is always 50 due to the high number of announcements), the outgoing traffic in that minute is 19 KB/s. This rate is quite high just for the DHT but it can be reasonable.

In the case the list of peers were not limited to 50 peers, if we suppose that in that minute the size of the list is 10000 (if in that single minute 1320 announcements were received, in 30 minutes it could be even higher) and for every get_peers message the whole list is tried to be sent (even though the IP packets are fragmented and it can carry to errors), the outgoing traffic rate for this case would be 3.72 MB/s which is not reasonable. So, limiting the list of peers is essential to reduce the problem of scalability. Even though the problem still exists, fixing a maximum size in the list of peers raises its limitations.

Bearing the incoming traffic for the most popular info hash in The Pirate Bay with a limited list of peers has been possible. However, with a list of peers without a fix size it would have generated a huge outgoing traffic. Anyway, if a rate is reasonable or not depends on the context where it is used. In this case we have supposed the usage of Internet connections for domestic users, perhaps for a mobile device those rates are not reasonable.

The growth of the number of messages seems to be linear looking at Figure 7.12, but this experiment is not large enough to prove this. The average number of messages is almost a straight line, enough samples would probably stabilize it. A large set of info hashes is necessary to study the growth. It would be useful in order to estimate the traffic in info hashes with a non existing number of peers nowadays.

We performed another experiment with an info hash having about 30000 peers as well without limiting the list of peers. The node enabled for the experiment sent about 9.8 GB in 20 hours just for the traffic of the DHT. We decided not to carry out again this experiment because we received a warning from Planetlab telling us that we had to reduce the traffic rate.

Our results prove that the problem of scalability represents a threat for the DHT. In the future info hashes with millions of peers may exist. Projects like P2P-Next [34] (which intends to broadcast live transmissions using P2P) can originate swarms with millions of peers in the broadcast of international events.

In the case of redirecting the incoming traffic as a DDoS attack (by using a fake list of peers as a response for the get_peers queries), the victim node would receive the incoming traffic corresponding to the get_peers queries. In the case of the info hash with 33239 peers it would be 4.5 KB/s which is not a dangerous rate. An info hash with many more peers or many nodes doing this at the same time would be necessary to make it serious.

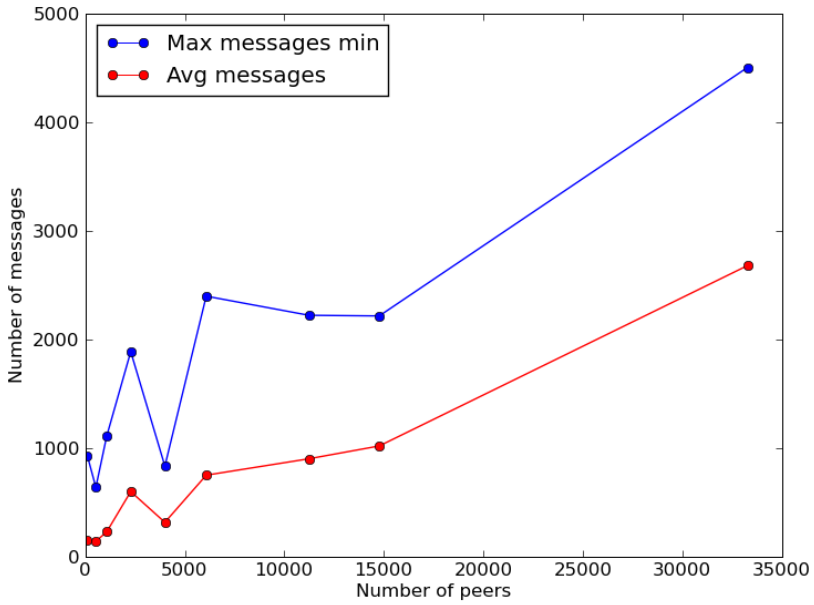

Figure 7.12. Number of messages according to the number of peers.

Data	IH 1	IH 2	IH 3	IH 4	IH 5
Estimation of peers	107	536	1083	2287	4001
Experiment duration	10h 29m	10h 30m	10h 28m	10h 32m	10h 33m
Total messages	97056	88424	143447	377758	199085
Total pings	1344	1372	1810	2656	2164
Total find nodes	48182	31824	64217	73128	78157
Total get_peers	37691	46610	64660	236569	105727
Total announcements	9839	8618	12760	65405	13037
Announcements for the info hash	7452	7097	9682	61457	9373
get_peers for the info hash	31620	42979	57071	226960	97173
Announcements other hashes	2387	1521	3078	3948	3664
get_peers other hashes	6071	3631	7589	9609	8554
Maximum of messages in one minute	929	637	1110	1886	833
Pings in that minute	3	1	2	8	4
Find nodes in that minute	106	71	121	168	144
get_peers in that minute	495	431	500	1112	478
Announcements in that minute	325	134	487	598	207
Average messages per minute	154	140	228	598	315

Table 7.6. Incoming messages for different info hashes (part1).

Data	IH 6	IH 7	IH 8	IH 9
Estimation of peers	6076	11228	14760	33239
Experiment duration	10h 31m	10h 36m	10h 34m	10h 39m
Total messages	472562	572521	645336	1711189
Total pings	2164	5298	3961	5984
Total find nodes	67398	115781	100675	136317
Total get_peers	323594	381844	446312	1228111
Total announcements	79406	69598	94388	340777
Announcements for the info hash	75980	62384	88409	330880
get_peers for the info hash	313825	360642	431533	1201341
Announcements other hashes	3426	7214	5979	9897
get_peers other hashes	9769	21202	14779	26770
Maximum of messages in one minute	2397	2221	2215	4502
Pings in that minute	2	8	8	8
Find nodes in that minute	134	254	223	250
get_peers in that minute	1367	1746	1297	2924
Announcements in that minute	894	213	687	1320
Average messages per minute	749	900	1018	2678

Table 7.7. Incoming messages for different info hashes (part2).

7.9 Experiment 9 - Distance between info hashes and their closest node

This experiment has measured the distance between info hashes and their closest node. This data is is useful to detect suspicious situations where nodes are too close to an info hash. It may provide a parameter to exclude suspicious nodes.

7.9.1 Expected results

As the identifier space is huge and its occupation is supposed to be very small (as we deducted in Chapter 5). We do not expect to find node identifiers closer than 130 to the info hash. According to the probabilities we present in Appendix F, the probability for a random node identifier to be closer to the info hash than 130 is 2^{-30}.

This experiment can not be affected by clients which do not respect the rules for the announcements because, in all the cases, the closest node to an info hash will contain list of peers (as show the experiment 2 and the experiment 5). Independently of which nodes are chosen for the announcements, this experiment just focuses on the closest.

7.9.2 Experiment definition

- Number of active nodes: 1.
- Number of info hashes to lookup: 1200.
- Number of lookups over every info hash: 10 (1 every 5 seconds).
- Data to measure: distance between info hashes and their closest node.

The tools complete_lookup and display_distance_closest have been used for this experiment.

7.9.3 Results

The result of the experiment is summarized in Table 7.8 and represented graphically in Figure 7.13. The distribution seems a normal distribution but our data is not large enough to prove it. The most typical distance is 138. As there is not any case of nodes with distance less than 131, this data can be used as a parameter to detect suspicious situations. If we observe a node containing a list of peers which is closer than 131 to the info hash, this node can be ignored (exists the possibility that a random identifier is closer than 131 to an info hash, but it is very improbable. It

does not matter if the node is ignored in this case), excluding by this way suspicious nodes (only nodes in relation with that info hash will exclude it. The rest of the DHT will treat it as a normal node, so it will not be excluded from the DHT).

Distance	Number of times
131	1 (0.11%)
132	6 (0.67%)
133	14 (1.55%)
134	26 (2.88%)
135	63 (6.98%)
136	124 (13.75%)
137	188 (20.84%)
138	227 (25.17%)
139	146 (16.19%)
140	69 (7.65%)
141	28 (3.10%)
142	7 (0.78%)
143	2 (0.22%)
144	1 (0.11%)

Table 7.8. Distance of the closest node to the info hash to the info hash.

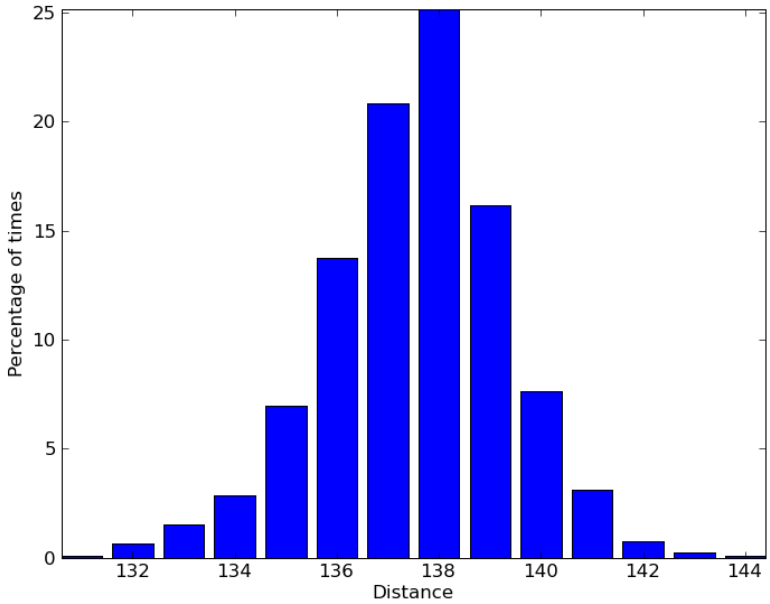

Figure 7.13. Distance of the closest node to the info hash to the info hash.

Chapter 8

Analysis of results

This chapters provides a summary and an analysis about all the results we obtained in the experiments presented in Chapter 7.

8.1 Profiling work

The experiments 1, 2 and 3 study differents aspects in the behavior and distribution of nodes containing a list of peers. They reach their goals since they have obtained a profiling of the part of the DHT they intended to study. The experiment 5 is another experiment of profiling which shows the reason of several unexpected found behaviors.

The frequency of announcements observed in the experiment 3 was as we expected. However, a strange behavior was observed, many nodes sent several announcements in a small interval of time. We do not know the reason of this behavior but we have an hypothesis: these nodes may have restrictions in the incoming network traffic (which can be caused, for example, by a firewall). Firewalls or routers using a NAT (Network Address Translation) have some behavior which limit the incoming traffic in the computer. In this case, nodes sending several announcements may not receive the responses and that would be the reason why they send the query several times. But we ca not be sure this is the reason, it should be investigated more in depth.

The experiment 5 found a reason for several unexpected results we obtained in some other experiments. It shows that some BitTorrent clients are not following the design rules for the announcements. BitSpirit clients are announcing in more than three nodes in all the cases. KTorrent clients are choosing nodes to announce itself which is almost impossible they are the closest to the info hash. Although we only analyzed three clients, the results obtained in this experiment are enough to find the reason of some unexpected results in other experiments.

The experiment 1 obtained a result which was not expected. It showed that the number of nodes tracking a given swarm is higher than expected. Also, the standard deviation of this value is too high to take into account the average. We discovered that this result is probably due to the results of the experiment 5 which shows that some clients send announcements to nodes which are not the closest to the info hash.

The experiment 2 showed that the nodes receiving the largest amount of announcements are the closest to the info hash. However, nodes which are part of the closest are receiving too many announcements. This is probably due to the results of experiment 5 which shows that some clients send announcements to more than three nodes.

This experiments are a first contact with the actual behavior of the DHT. With them, we have found interesting behaviors to study but they also provide an approach to study more specific topics like the vulnerabilities.

8.2 Censorship

The experiment 4 was an attempt of censorship. The results we obtained in some of the previous experiments provided the data to design this experiment. According to the experiment 1, more than 12 nodes should be enough. According to the experiment 2, their distance to the info hash did not matter (as long as they were the closest). According to the experiment 3, in about one hour they should control the list of peers.

Censorship was not possible in any case. Even though in one attempt there was a small period of time when it was, there were always new nodes containing list of peers. The reason of this result is again the result obtained in the experiment 5. Although there are clients which announce themselves in more than three nodes, it should still be possible to censor the info hash. But, as there are clients which announce not only in the closest nodes, it is almost impossible to censor a list of peers by positioning nodes as the closest to the info hash. Clients behaving like this are those making impossible censorship. But, if, for example, an info hash with only UTorrent peers were found, it would probably be possible to censor it. The heterogeneity of different clients and their behaviors is making censorship impossible.

The implicit mechanism against censorship offers an implicit security in the DHT. Although it is due to BitTorrent clients which do not follow the specifications of the DHT (what may be harmful for the DHT), as long as the peers of an info hash are heterogeneous clients of BitTorrent, a protection against censorship is almost guaranteed.

8.3 Impact of nodes which do not follow the specifications of the DHT

As in the experiment 5 we discovered that some nodes do not follow the design rules, the experiments 6 and 7 studied the impact in the DHT of these nodes.

The experiment 6 shows that the number of nodes containing list of peers too far from the info hash are too many. The distribution in the distance to the info hash of these nodes seems to be a normal distribution, the closest nodes seem to follow it, but the farthest nodes are many more than the closest. The high number of far nodes is probably due to the result of the experiment 5. However, the distance of the closest nodes can not be altered by a behaviors not following the design since nodes seem to announce always in them. This is the motivation for the experiment 9.

The experiment 7 analyzes the distribution of the list of peers among the nodes containing it. The result shows that the three closest nodes are those containing the most of the list of peers. However, there are nodes containing list of peers which are not part of the closest. The percentage of the list of peers they contain seems to decrease exponentially according to the distance order to the info hash. An hypothesis for this result is that some clients announce themselves in the nodes they find in the lookup. In [2] it is demonstrated that if the DHT has n nodes, the number of hops to find a value is $O(log(n))$. This could explain the exponential decrease.

8.4 Scalability and DDoS attacks

The experiment 8 checks the problems of scalability and DDoS attacks. It shows that, in very popular info hashes, the impact of the incoming traffic in the closest node to the info hash is not too critial with the info hashes we found (having up to 33000 peers), it is a reasonable rate for a usual domestic connection, but still too much just for the DHT. The problem of the outgoing traffic is more serious. The traffic rates are not reasonable. As the list of peers for a popular info hash becomes very large, the outgoing messages become huge.

In order to reduce the impact of this problem, we propose an easy solution. It consists on limiting the size of the list of peers in the get_peers responses. As studied along this thesis, a list of more than 50 peers does not fit in a UDP packet. The option of limiting it to 50 peers raises the limit of the scalability problem. With this solution, the traffic of the most popular info hash we found has been borne. Even though this solution reduces the problem, it still exists and generates too much traffic just for the DHT.

In the future info hashes with millions of peers may exist. Projects like P2P-

Next [34] (which intends to broadcast live transmissions using P2P networks) can originate swarms with millions of peers in the broadcast of international events. Also, BitTorrent has nowadays millions of users, if some day there is an extremely popular info hash it could have millions of peers as well. The traffic in the nodes containing the list of peers of this kind of info hashes would make the problem much more serious. We provide some data to estimate the traffic they would generate in the results of the experiment 8.

8.5 Detection of suspicious nodes

The experiment 9 arises with the results of the experiment 6. The experiment 6 shows the distribution of distances of nodes containing list of peers, however, the number of nodes too far from the info hash is higher than expected. The distribution seems a normal distribution for nodes which are not too far, this data can help to detect suspicious nodes.

As the distance of the closest nodes to an info hash is not affected by behaviors which do not follow the design, it can provide a non altered measure. Measuring the distance between an info hash and its closest node can give a parameter to detect nodes suspiciously close to an info hash. The data obtained seems to have a normal distribution in these distances. Also, the distribution can help to check if a set of the closest nodes to an info hash fits in the distribution. It can help to detect suspicious nodes. We have not found nodes closer to the info hash than 131 and the most typical distance is 138.

Chapter 9

Future work

The work of this thesis studies an unexplored part of Mainline DHT. There is not much previous work related with the investigations we made. In this chapter we present all the things that it would be interesting to study according to the experience of this thesis.

9.1 Improvement of the tools and larger experiments

The beginning of the work was the development of a set of tools. Some experiments have not been too exhaustive. Some results are significant but they can not be taken as an assertion since they should check a larger part of the DHT (for the experiments like experiment 1, the experiment 6 or the experiment 9). So, part of the immediate future work is doing big enough experiments to give a global and a complete view of Mainline DHT. These experiments would probably require an improvement of the tools to make them more efficient due to the huge quantity of information they should manage.

The tools have not been tested with such a huge quantity of information like that which could be obtained monitoring the whole DHT. Probably, the first improvement needed would be related with the log files. These files have a human readable format, which is very useful to check the results before processing them. But the storage could be much more reduced enabling different levels of log files. It could be added a level of logs which stored information to make it easy to read by a script and using less storage.

9.2 Study of anomalous behaviors

One of the important facts discovered in this thesis has been that some BitTorrent clients do not follow the specifications of the DHT (as shown in the experiment 5). The coexistence of many different implementations and their behaviors makes wrong some theoretical assumptions deducted from the design, influencing in the behavior of the DHT. Some of these different behaviors are beneficial (for example, the experiment 5 shows that, as there are clients which do not choose only the closest nodes to an info hash for the announcements, it is almost impossible to censor the info hash) but on the other hand they generate extra traffic.

There are anomalous behaviors that seem to be harmful for the DHT (like those clients found in the experiment 3 which send several announcement messages at the same time when only one is necessary). These unexpected behaviors should be investigated more in depth or the developers should be contacted. They can be a bug or perhaps there is any reason and it could be possible to document it and to propose it as an improvement.

Some of the anomalous behaviors discovered are the following:

- The number of nodes where a node chooses to announce itself is not the same in all the BitTorrent clients (as shown in the experiment 5).
- Some BitTorrent clients choose nodes which are not part of the closest to the info hash to announce themselves (as shown in the experiment 5).
- Clients announcing themselves send several announcements in a few seconds (as shown in the experiment 3).
- Some clients responding to a get_peers query provide a list of peers with repeated peers (as observed in several experiments).

9.3 Management of the list of peers

A very important fact (which at the beginning did not seem so important) is the management of the list of peers. At the beginning we discovered that lists of peers with more than 50 peers do not fit in a UDP packet. There are two alternatives to solve this problem: limiting the size of the list of peers or fragmenting the IP packets to be reassembled in the destination. The experiment 8 was shown that the scalability problem is highly reduced using a limited list of peers. But this solution may have some implications which were not discovered. It should be analyzed if this limitation in the list of peers is influencing the behavior of the DHT somehow.

Some policies to manage a limited list of peers should be studied. In the case of this thesis, the list consisted of the last 50 announcements, but perhaps some

better policies can be defined (like choosing random peers, or fragmenting packets even though the whole list is not sent, or checking if the peers are alive). An exhaustive study of different policies should be done because we have proved that the management of the list of peers can be a critical point in the DHT. This study should also take into account the BitTorrent protocol where peers who have joined the data exchange can exchange peers in order to get more peers (as explained in [30]).

9.4 Deeper study of the DHT

In this thesis we have studied a specific part of the DHT and, with it, we have discovered situations which are interesting to study. Before this thesis we did not have too much data about the acual behavior of the DHT. By looking at its actual behavior we have understood what is really happening on it.

We have not studied the whole DHT but a specific part of it. It would be very useful to obtain a global profile of more aspects. We have demonstrated that there are clients which do not follow the specifications, this fact implies an actual DHT which does not match with the theoretical model. Working on a DHT whose behavior is unknown can invalidate theoretical works. For example, if someone did a simulation of the DHT using only UTorrent clients, the data obtained might not be representative since it would have an unreal view of the DHT. For the same reason, it is also very important to study the proportion in the usage of BitTorrent clients in the DHT (in the experiment 2 we provide some data about this).

In deeper studies we would probably find more unexpected behaviors caused by different BitTorrent clients. It would be very useful to obtain a classification of clients and their behaviors (for example, classifying for every client: number of nodes where it chooses to announce, maximum size of the list of peers, frequency of announcements, etc). This data would be very useful for other investigations. For example, if we had had this data when we did the analysis, we would have known that censorship was almost impossible.

We propose two kind of works for studying the DHT: looking at its actual state and looking at the behavior of the individual clients. A complete profile of both the sides would provide a very complete source of information. This information could be used for proposing improvements, identifying clients abusing the network, finding more vulnerabilities, etc.

9.5 Further study of the vulnerabilities

Although the experiment 8 proved that the scalability problem is not too critical, the problem still exists. Maybe in an info hash with a million of peers, nodes

containing the list of peers could not bear the traffic generated. We provide data in the results of the experiment 8 to estimate the traffic of non existing info hashes nowadays.

9.5.1 Sketching a solution for the problem of scalability

A mechanism to solve the problem of scalability should distribute tasks of the storage of the list of peers in more or less nodes according to the size of the swarm. It should be a mechanism which intended that all the nodes in the DHT had a similar traffic due to the DHT.

A complete solution could be performed from those nodes providing the closest nodes to the info hash in the lookup. They could detect that too many nodes are looking for the closest nodes and limit somehow the access to them. It would be a mechanism to make dynamic the set of nodes containing list of peers. But this kind of solution could carry to a denial of access to those nodes. A complete solution should be defined and studied.

Something which could help solving the problem is caching the values in the DHT. In the design of Kademlia [2] there is an explanation of how values in Kademlia can be cached. They explain that, as all lookups converge along the same path, caching alleviates hot spots. This is not carried out in Mainline DHT. This solution would be easy to implement if values in the DHT were static, however, in Mainline DHT they are dynamic so implementing it would not be immediate.

9.5.2 Preventing DDoS attacks

Another point of view of the scalability problem, as explained in the experiment 8, is the possibility of providing fake lists of peers to perform a DDoS attack. This vulnerability still exists. It would have the same impact on the victim node than the impact of the incoming traffic in the scalability problem (since the incoming traffic would be somehow redirected to the victim node).

The experiment 8 showed that the incoming traffic is not a too critial problem in the scalability problem, so it would not be that serious in the DDoS attacks as well. However, if many users agree to position nodes close to several popular info hashes it could be a serious problem. We provide some parameters in the experiments of this thesis to find suspicious nodes. These parameters should be refined with bigger experiments. However, nodes still can be part of the closest without being suspiciously close to the info hash. A complete solution (which would be the same for the scalability problem) should be proposed.

9.6 Long term modifications

Something that should be fixed is the possibility of choosing the identifier of a node when it is supposed to be random that we studied in Chapter 3. A mechatnism to detect if an identifier is random or not would solve several problems.

One possible solution is making compulsory to calculate the identifier as a hash of the IP address of the node (like in Azureus [24]). It would permit checking if a node has chosen its identifier by the right way. If a node does not follow it, it should be ignored and no communication should be established with it.

The problem of adapting this solution is that all the nodes in the DHT should start using it at the same time, this is hard due to the coexistence of many different clients. It would suppose a global change but a transition could be established. In the transition, nodes could try to keep on their routing tables nodes with a right identifier. After some time using this solution, nodes with a right identifier could start ignoring those with a wrong identifier.

Chapter 10

Conclusion

This chapter presents the conclusion extracted from the results obtained in the experiments and, in general, from the experience of the development of this thesis. It also compares the results with the goals of the thesis and explains the contribution of this work.

10.1 Goal achievement

In Chapter 1 we proposed a set of goals. This section compares the results with the goals in order to show that they have been achieved.

In Chapter 3 we presented a theoretical analysis about the generation, distribution and obtaining of lists of peers in the DHT. The experiments presented in Chapter 7 explore this part of the DHT and compare the results with the analysis. The combination of all the experiments presents a global view of this part of the DHT.

We have developed a set of tools to interact with the DHT, they are presented in Chapter 6. These tools have served for testing the theoretical analysis presented in Chapter 3. The tools are Open Source so anyone can use them for further investigations.

The experiment 4 explores the problem of censorship. It shows that censorship was not possible in any case and we show the reasons in the experiment 5, this experiment shows that there is an implicit mechanism against censorship. The experiment 8 quantifies the problems of scalability and DDoS attacks. It shows that the problem of scalability is serious for the outgoing traffic and proposes a solution to reduce its impact by limiting the size of the list of peers. It also shows that the problem of DDoS attack is not too critical if only one user intends to do it with the info hashes we found.

We have reached our goals but the exploration of Mainline DHT has arisen a lot of work interesting to do (they are presented in Chapter 9).

10.2 Contribution

This section presents all the contribution arisen with the work of this thesis.

10.2.1 Results

Providing a set of tools to monitor Mainline DHT. The tools we have developed (which are presented in Chapter 3) can be used or adapted to collect different data from the DHT. Their usage can be useful for future investigations like the future work proposed in Chapter 9 or new ones related with Mainline DHT. These tools are Open Source so anyone can access them.

Proving that censorship is almost impossible by adding a set of nodes as the closest to an info hash. The experiment 4 shows that censorship hasn't been possible in any case. Documenting why this vulnerability is not possible is an important contribution to the community since the analysis presented in Chapter 3 pointed out it was possible. The experiment 5 documents that it has not been possible because there are nodes which do not announce in the three closest nodes to the info hash.

Demonstrating that a node close to a very popular info hash may suffer a huge unexpected traffic rate. We documented in the experiment 8 that some nodes in the DHT may suffer a traffic rate much higher than the rest due to the DHT. Even though this traffic may not be very critical for some users nowadays, detecting that some nodes have to bear a much higher traffic than others is an important contribution since it breaks the equality of the DHT.

10.2.2 Partial results needing more work

Providing a mechanism to reduce the problem of scalability. In the experiment 8 we propose a mechanism to reduce the impact of the problem of scalability by limiting the size of the list of peers. This mechanism is useful for developers so they adapt it to their Mainline DHT clients. As it does not need to be implemented in all the clients at the same time, it is an easy way to improve the software.

Providing some guidelines to fix the problems of scalability and DDoS attacks. In Subsection 9.4.1 we provide some baselines for solutions of these problems. Currently they are not too critical with the improvement proposed, but in the future they should be taken into account. In the experiment 8 we also provide data to estimate the traffic in non existing info hashes nowadays.

10.2.3 Observations requiring further studies

Finding clients which do not follow the specifications of the DHT. We found out some clients which do not follow the speficitations as we show in the experiment 3 (which shows that some clients do not respect the frequency for the announcements) and the experiment 5 (which shows that some clients do not announce in the three closest nodes to the info hash). These inconsistencies suppose a lack of fairness in the DHT since some clients are using it in a different way than others.

Demonstrating that a list of peers can be used to perform a DDoS attack and providing data to detect suspicious nodes. In the experiment 8 we showed how nodes containing list of peers could perform DDoS attacks. This problem is not too critical but it can become more serious in the future. In the experiment 6 and the experiment 9 we provide some data in order to find malicious nodes and by looking at their distance to the info hash.

10.3 General conclusion

This section presents some general conclusion extracted from all the work done in this thesis.

The correct working of BitTorrent is based on the equal cooperation of all the participants. Identifying and avoiding situations which do not respect this principle helps to ensure the equality of all the nodes. We have analyzed, documented and proposed solutions for some vulnerabilities in Mainline DHT. Our study helps developers to improve the security and correct behavior of their BitTorrent clients. There are millions of users of BitTorrent clients nowadays, an improvement in such a popular software is a contribution to all of them. This work also helps to find and exclude malicious users. The result of this thesis helps to keep the equality in BitTorrent.

Proposing individual changes for clients is easy, but global changes are hard in a DHT like Mainline DHT. If a global change has to be adapted in the whole DHT at the same time it is almost impossible to carry it out. The existence of many different clients and versions makes this coordination very hard. But a transition for global changes may be reasonable. If a global change is proposed, a transition period can be established by using an intermediate solution temporarily.

Every solution or improvement should be documented. Although a solution is not implemented, it can help to warn people of problems and to take it into account in new designs of DHTs. All the documentation of problems in Mainline DHT can be taken further to improve any other DHT or new designs or versions of DHTs.

The influence of UTorrent in Mainline DHT is very important. The

experiment 2 showed that UTorrent is probably the most popular client with an important difference of usage with the rest. UTorrent seems to follow all the specifitations for the announcements. Probably, global changes should start in this client because UTorrent nodes are a very important part of Mainline DHT.

Some of the problems presented in this thesis exist due to the possibility of choosing the identifier for a node instead of a random one. This property has been a very important point in this thesis. In the future, mechanisms to ignore those nodes which choose their identifier should exist.

Studying actual aspects of Mainline DHT can be a help to understand the theory and to make hypothesis. We read some papers presenting experimental perspectives of Mainline DHT at the beginning of the thesis and they were very useful to understand better the protocol. This thesis can also help to understand in depth how Mainline DHT works. Also, some hypothesis done in this thesis deducted from the design were wrong. Better hypothesis can be made taking into account both the theory and the actual behavior of the DHT.

Appendix A

Glossary of terms

- DDoS attack: a denial-of-service attack (DoS attack) or distributed denial-of-service attack (DDoS attack) is an attempt to make a computer resource unavailable to its intended users.
- DHT: a structured overlay that uses key-based routing for put and get index operations and in which each peer is assigned to maintain a portion of the index.
- Hash: a hash function is any well-defined procedure or mathematical function that converts a large, possibly variable-sized amount of data into a small datum.
- Info hash: 160-bit SHA-1 hash of an object.
- IP address: an Internet Protocol (IP) address is a numerical label that is assigned to devices participating in a computer network that uses the Internet Protocol for communication between its nodes.
- Key: sequence of bits used to index a value, usually much smaller than the value.
- Leecher: a peer or any client that does not have 100% of the object.
- Node: computer connected to the Internet running a BitTorrent client using the DHT.
- Object: chunk of data.
- P2P: Peer-to-Peer systems are distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes

while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server or authority.

- Peer: an end system, node, or host that is a member of a P2P system.
- Seeder: a seeder is a peer that has a complete copy of the torrent and still offers it for upload.
- SHA-1: is a cryptographic hash function designed by the National Security Agency (NSA) and published by the NIST as a U.S. Federal Information Processing Standard. SHA stands for Secure Hash Algorithm.
- Swarm: together, all peers (including seeders) sharing a torrent. For example, six ordinary peers and two seeders make a swarm of eight.
- Torrent: a small meta-data file which contains information about the object to download, not the object itself. It is downloaded from a web site (BitTorrent file extension is .torrent). Part of the meta-data is the info hash of the object.
- Tracker: server on the Internet that coordinates the action of BitTorrent Clients. Upon opening a torrent, the tracker is contacted and a list of peers to connect to is received. Throughout the transfer, the client will query the tracker, telling it how much it has downloaded and uploaded and how much before finishing.
- Value: chunk of data indexed by a key.

Appendix B

Acronyms

- DDoS: Distributed Denial of Service.
- DHT: Distributed Hash Table.
- IP: Internet Protocol.
- P2P: Peer-to-Peer.
- SHA: Secure Hash Algorithm.
- TCP: Transmission Control Protocol.
- UDP: User Datagram Protocol.
- URL: Uniform Resource Locator.

Appendix C

Mainline packet format

The packet format defined here is extracted from [20].

PING query:

```
"t": token
"y": "q"
"q": "ping"
"a": "id": sender_id
```

"t": parameter is a token.
token: transaction ID.
"y": the query contains two additional keys: "q" and "a".
"q": the message is a query.
"ping": the query is a ping query.
"a": there are arguments.
"id": parameter is an identifier.
sender_id: node identifier of the querier.

PING response:

```
"t": token
"y": "r"
"r": "id": responder_id
```

"t": parameter is a token.
token: transaction ID (must be the same sent in the query).
"y": the response contains an additional key: "r".
"r": the message is a response.
"r": there are returned values.

"id": parameter is an identifier.
responder_id: node identifier of the responder.

FIND_NODE query:

```
"t": token
"y": "q"
"q": "find_node"
"a": "id": sender_id
     "target": target_node
```

"t": parameter is a token.
token: transaction ID.
"y": the query contains two additional keys: "q" and "a".
"q": message is a query.
"find_node": the query is a find_node query.
"a": there are arguments.
"id": parameter is an identifier.
sender_id: node identifier of the querier.
"target": parameter is the node to find.
target: identifier of the node to find.

FIND_NODE response:

```
"t": token
"y": "r"
"r": "id": responder_id
     "nodes": info_node_1
              info_node_2
              ...
              info_node_8
```

"t": parameter is a token.
token: transaction ID (must be the same sent in the query).
"y": the response contains an additional key: "r".
"r": the message is a response.
"r": returned values.
"id": parameter is an identifier.
responder_id: node identifier of the responder.
"nodes": parameter is a list of nodes.
info_node_i: information about node i (identifier, IP address and UDP port) compacted.

GET_PEERS query:

```
"t": token
"y": "q"
"q": "get_peers"
"a": "id": sender_ID
     "info_hash": target_info_hash
```

"t": parameter is a token.
token: transaction ID .
"y": the query contains two additional keys: "q" and "a".
"q": message is a query.
"get_peers": the query is a get_peers query.
"a": there are arguments.
"id": parameter is an identifier.
sender_id: node identifier of the querier.
"info_hash": parameter is an info hash.
target_info_hash: info hash to get_peers.

GET_PEERS response if the queried node has no peers (in that case will provide closer nodes):

```
"t": token
"y": "r"
"r": "id": responder_id
     "token": token_ann
     "nodes": info_node_1
              info_node_2
              ...
              info_node_8
```

"t": parameter is a token.
token: transaction ID (must be the same sent in the query).
"y": the response contains an additional key: "r".
"r": the message is a response (must be the same sent in the query).
"r": there are returned values.
"id": parameter is an identifier.
responder_id: node identifier of the responder.
"token": parameter is a token to send in case the querier wants to announce.
token_ann: value of the token.
"nodes": parameter is a list of nodes.
info_node_i: information about node i (identifier, IP address and UDP port) com-

pacted.

GET_PEERS response if the queried node has peers:

```
"t": token
"y": "r"
"r": "id": responder_id
     "token": token_ann
     "values": peer_1
               peer_2
               ...
               peer_n
```

"t": parameter is a token.
token: transaction ID (must be the same sent in the query).
"y": the message contains an additional key: "r".
"r": message is a response.
"r": there are returned values.
"id": parameter is an identifier.
responder_id: node identifier of the responder.
"token": parameter is a token to send in case the querier wants to announce.
token_ann: value of the token.
"values": parameter is a list of peers.
peer_i: information about peer i (IP address and TCP port) compacted.

ANNOUNCE_PEER query:

```
"t": token
"y": "q"
"q": "announce_peer"
"a": "id": sender_ID
     "info_hash": info_hash_ann
     "port": TCP_port
     "token": token_gp
```

"t": parameter is a token.
token: transaction ID.
"y": the message contains two additional keys: "q" and "a".
"q": message is a query.
"announce_peer": the query is an announce_peer query.
"a": there are arguments.
"id": parameter is an identifier.

sender_id: node identifier of the querier.
"info_hash": parameter is the info hash of the announcement.
info_hash_ann: value of the info hash.
"port": parameter is TCP port of the BitTorrent protocol.
TCP_port: value of the TCP port.
"token": parameter is the token provided in the get_peers query.
token_gp: value of the token.

ANNOUNCE_PEER response:

```
"t": token
"y": "r"
"r": "id": responder_id
```

"t": parameter is a token.
token: transaction ID (must be the same sent in the query).
"y": the message contains an additional key: "r".
"r": message is a response.
"r": there are returned values.
"id": parameter is an identifier.
responder_id: node identifier of the responder.

Appendix D

Obtaining of a large set of info hashes

Some of the experiments had to use a large set of info hashes in order to get reliable results, for this kind of experiments we have used a set of 1200 info hashes obtained from The Pirate Bay [8]. The way to get them has been using the following Unix command:

```
wget http://thepiratebay.org/browse/100/ -r -x --no-parent
```

This command retrieves all the files contained in the folder browse/100 of the webpage. In the webpage, it is possible to see that all the torrents are contained in folders with the prefix /browse/100, /browse/200, etc. So, repeating this command for five or six times, a large set of html files containing info hashes is obtained.
Once all this html files have been downloaded, it is necessary to parse them and get the info hash from them (it was previously checked that it was contained there). In all of them it is part of the line 155 of the file. To get it, it is been used the following Unix command:

```
sed '155q;d;' file.html | sed -n 's/.*\([a-f0-9]\{40\}\).*/\1/p'
```

The first part (before the vertical bar) gets the line number 155 of the file and uses it as input for the second command. The second command parses that line searching a sequence of 40 hexadecimal digits and prints it.
With this, we have developed a Python script to parse all the downloaded files and print all the info hashes in a single file.

A summary of the number of peers for this set of info hashes is summarized in D.1. The number of peers is all those peers observed in all the lookups.

Number of peers	Number of info hashes
0	298
1-49	658
50-99	61
100-149	47
150-199	41
200-249	30
250-299	20
300-349	15
350-399	6
400-449	9
450-499	6
500-549	1
550-599	2
600-649	3
650-699	1
700+	2

Table D.1. Big set of info hashes and their number of peers.

Appendix E

Experiments with a set of nodes

Some of the experiments require a set of nodes. In order to perform this experiments, some of the nodes of the Planetlab network have been used. Planetlab provides access to hundreds of computers all around the world and with different IP addresses. For some experiments up to 30 nodes have been used at the same time.
Nodes have been accessed using SSH [18]. We have used one node to coordinate the others, by using SSH, this node sends commands included in the Unix SSH command to the other nodes.

Appendix F

Probabilities of identifiers

- XOR distance is the result of the bitwise XOR of two sequences. Log distance performs the same operation, however, the result is not the number itself but the position (counting from the right and starting from zero) of the first bit which is 1 (counting from the left).
- In an identifier space of a bits, log distances belong to the interval $[-1, a)$.
- Two sequences have a log distance of -1 between them in an identifier space of a bits if they are equal (i.e. they have a bits equal).
- Two sequences have a log distance of $a-1$ between them in an identifier space of a bits if their first left bit is different.
- Two sequences have a log distance of d between them where $0 \leq d < a-1$ in an identifier space of a bits if their first $a-d-1$ left bits are equal and the next one is different.
- The probability that two random sequences of n bits are equal is $\frac{1}{2^n}$.
- The probability of two random sequences to have a log distance equal to -1 between them in an identifier space of a bits is $\frac{1}{2^a}$.
- The probability of two random sequences to have a log distance equal to $a-1$ between them in an identifier space of a bits is $\frac{1}{2}$.
- The probability of two random sequences to have a log distance equal to d between them where $0 \leq d < a-1$ in an identifier space of a bits is the probability that their first $a-d-1$ left bits are equal and the next one is different.
- The probability of two random sequences to have a log distance equal to d between them where $0 \leq d < a-1$ in an identifier space of a bits is $\frac{1}{2^{a-d-1}} * \frac{1}{2} = \frac{1}{2^{a-d}}$

- The probability of two random sequences to have a log distance equal or less to d between them in an identifier space of a bits is the probability that they have the $a - d - 1$ or more left bits in common and one different.
- The probability of two random sequences to have a log distance equal or less to d between them where $0 \leq d < a - 1$ in an identifier space of a bits is $\frac{1}{2^a} + \sum_{i=0}^{d} \frac{1}{2^{a-i}} = 2^{d-a+1}$.

Appendix G

Identifier space occupation

Positioning new nodes closer to an info hash than the closest one should be easy. According to the estimation of nodes made in [7], the number of nodes in the DHT is 1.3 million. It is a quite old estimation, supposing that nowadays it has grown a lot and it is about $17,000,000 \approx 2^{24}$, and knowing that the identifier space is 2^{160}, the identifier space occupation would be approximately $\frac{2^{24}}{2^{160}} \approx 2^{-136}$. Supposing that nodes are uniformly distributed, the average space between two consecutive nodes would be $\frac{2^{160}}{2^{24}} \approx 2^{136}$.

Even though since that estimation the number of nodes had grown much more or the distribution of nodes were not uniform, this average distance would still be huge. In conclusion, distance between an info hash and its closest node is supposed to be big enough to add there millions of nodes.

Appendix H

Classification of tools

Table H.1 presents a classification of the pairs of tools with passive loggers and Table H.2 the same with active loggers. A summary of the purpose of every pair of tools is given in both the tables.

Logger	Displayer	Result
censor	-	Positions a node with a given distance to an info hash in the DHT. It is the base to develop the other passive loggers.
announcements	display_announcements_distance	Displays graphically the number of announcements received in a node according to its distance to the info hash. It requires a log for every distance to display.
announcements	display_freq_announcements	Calculates the average frequency of the announcements for every node announcing and displays the statistics of these frequencies.
incoming_traffic	display_incoming_traffic_statistics	Summarizes the received traffic in a node showing the number of received messages for every type of message.

Table H.1. Tools with passive loggers and their purpose.

Logger	Displayer	Result
complete_lookup	display_list_distribution	Displays how the list of peers is distributed among all the peers containing it for the given info hashes.
complete_lookup	display_distance_closest	Displays the average distance between the given info hashes and their closest node.
complete_lookup	display_average_number_nodes	Displays the average number of nodes containing list of peers for the given info hashes.
distribution	display_distribution	Displays the distance between the given info hashes and all nodes containing list of peers for them.
evolution	display_evolution	Displays how a list of peers grows for every node containing list of peers for a given info hash.

Table H.2. Tools with active loggers and their purpose.

Bibliography

[1] B. Cohen. Incentives build robustness in BitTorrent. In Workshop on Economic of Peer-to-Peer Systems, Berkeley, CA, June 2003.

[2] P. Maymounkov and D Mazières. Kademlia: A peer-to-peer Information System Based on the XOR Metric. In proceedings of IPTPS, Cambridge, MA, IEEE, March 2002.

[3] Planetlab. http://www.planet-lab.org/ (last visited May 2010).

[4] R. Schollmeier. A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. In proceedings of the First International Conference on Peer-to-Peer Computing, IEEE, 2002.

[5] Stoica and Ion et al. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In proceedings of SIGCOMM (ACM Press New York, NY, USA), 2001.

[6] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, November 2001.

[7] S. Crosby and D. Wallach. An Analysis of BitTorrent's Two Kademlia-Based DHTs. In department of Computer Science, Rice University, Houston, Texas, USA, 2007.

[8] The Pirate Bay. http://thepiratebay.org/ (last visited May 2010).

[9] Mininova. http://www.mininova.org/ (last visited May 2010).

[10] A. Legout, G. Urvoy-Kellerand and P. Michiardi. Understanding bittorrent: An experimental perspective. Technical Report (inria-00000156, version 3 - 9 November 2005), INRIA. Sophia Antipolis, France, November 2005.

[11] K. El Defrawy, M. Gjoka and A. Markopoulou. BotTorrent: misusing BitTorrent to launch DDoS attacks. In proceedings of the 3rd USENIX workshop on

Steps to reducing unwanted traffic on the internet, p.1-6, Santa Clara, CA, June 18, 2007.

[12] K. Cheung Sia. DDoS Vulnerability Analysis of Bit-Torrent Protocol. In UCLA Tech. Report, Spring 2006.

[13] N. Liogkas, R. Nelson, E. Kohler and L. Zhang. Exploiting BitTorrent for fun (but not profit). In Proc. of IPTPS, 2006.

[14] M. Sirivianos, J. H. Park, R. Chen and X. Yang. Free-riding in BitTorrent Networks with the Large View Exploit. In Proc. of IPTPS, Bellevue, WA, February 2007.

[15] J. Falkner, M. Piatek, J. P. John, A. Krishnamurthy and T. Anderson. Profiling a million user dht. In proceedings of the 7th ACM SIGCOMM conference on Internet measurement, San Diego, California, USA, October 24-26, 2007.

[16] J. Yu, C. Fang, J. Xu, E. C. Chang and Z. Li. ID repetition in KAD. In Citeseer, 2010.

[17] eMule. http://www.emule-project.net/ (last visited May 2010).

[18] RFC 4251. The Secure Shell (SSH) Protocol Architecture. http://www.ietf.org/rfc/rfc4251.txt.

[19] Wireshark. http://www.wireshark.org/ (last visited May 2010).

[20] BEP0005: DHT Protocol - http://www.bittorrent.org/beps/bep_0005.html (last visited May 2010).

[21] BEP0020: Peer ID convention. http://www.bittorrent.org/beps/bep_0020.html (last visited June 2010).

[22] Mainline - http://www.bittorrent.com/ (last visited June 2010).

[23] Azureus - http://azureus.sourceforge.net/ (last visited June 2010).

[24] M. Steiner and E. W. Biersack. Crawling Azureus. In Technical Report RR-08-233, 2008.

[25] RFC3174. US Secure Hash Algorithm 1 (SHA1). http://www.ietf.org/rfc/rfc3174.txt.

[26] UTorrent. http://www.utorrent.com/ (last visited June 2010).

[27] KTorrent. http://ktorrent.org/ (last visited June 2010).

[28] BitSpirit - http://bitspirit.uptodown.com (last visited June 2010).

[29] BEP0003: The BitTorrent Protocol Specification.
http://www.bittorrent.org/beps/bep_0003.html (last visited June 2010).

[30] D. Wu, P.Dhungel, X. Hei, C. Zhang, K. W. RossUnderstanding Peer Exchange in BitTorrent Systems. In Proc. of IEEE International Conference on Peer-to-Peer Computing (IEEE P2P), Delft, Netherlands, Aug 2010.

[31] I. Kelényi and J. K. Nurminen. Energy aspects of peer cooperation - Measurements with a mobile DHT system. In Proc. Cognitive and Cooperative Wireless Networks Workshop in the IEEE International Conference on Communications Beijing, China, 2008, pp. 164 - 168, 2008.

[32] Ares. http://aresgalaxy.sourceforge.net/ (last visited June 2010).

[33] Python. http://www.python.org/ (last visited June 2010).

[34] P2P-Next. http://www.p2p-next.org/ (last visited June 2010).