hMule: an unified KAD-BitTorrent file-sharing application

Damian Vicino, Juan Pablo Timpanaro, Isabelle Chrisment, Olivier Festor

hMule: an unified KAD-BitTorrent file-sharing application

Damian Vicino, Juan Pablo Timpanaro, Isabelle Chrisment, Olivier Festor

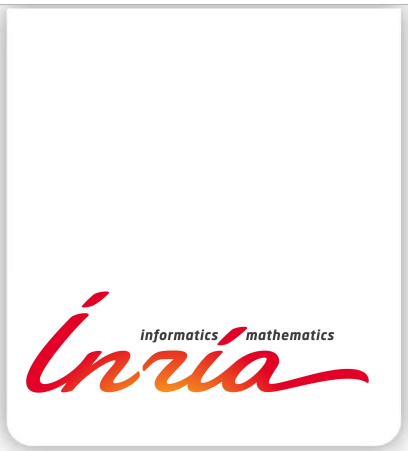
TECHNICAL REPORT

N° 7815

September 2011

Project-Team madynes

ISSN 0249-0803 ISRN INRIA/RT--7815--FR+ENG



hMule: an unified KAD-BitTorrent file-sharing application

Damian Vicino, Juan Pablo Timpanaro, Isabelle Chrisment, Olivier Festor

Project-Team madynes

Technical Report n° 7815 — September 2011 — 51 pages

Abstract: BitTorrent is a fast, popular, P2P filesharing application with no search engine mapping keywords to contents. The trackerless approach uses a DHT based on Kademlia to look for sources when the SHA1 hash of the metadata of the content to transfer is known. However, this DHT implementation is exposed to several identified security issues. On the other hand, the KAD network uses a solid DHT implementation based also on Kademlia and developed for the eMule/amule P2P clients. The KAD DHT provides an extra level of indexation to map keywords to file identifiers that is used as search engine. We produced a hybrid implementation compatible with both P2P file sharing networks to have the KAD advantages on indexation and the BitTorrent speed for transfer without losing backward compatibility.

Key-words: KAD, BitTorrent, DHT, P2P Architecture, Performance, Security

RESEARCH CENTRE
NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

hMule: an unified KAD-BitTorrent file-sharing application

Résumé : BitTorrent est un réseau rapide et populaire de partage de fichiers P2P sans moteur de recherche. Il utilise une DHT basée sur Kademlia pour chercher des sources lorsque l'empreinte SHA1 des métadonnées du contenu à transférer est connue l'avance. Cependant, cette implantation de la DHT présente de nombreuses failles de sécurité. Le réseau KAD repose, par contre, sur une DHT solide fondée aussi sur Kademlia et qui est développée pour les clients P2P emule/amule. La DHT de KAD fournit un niveau supplémentaire d'indexation qui est utilisé comme moteur de recherche. Dans ce rapport, nous proposons une implantation hybride, compatible avec les réseaux P2P de partage de fichiers, BitTorrent et KAD, afin d'obtenir les avantages KAD pour l'indexation et de BitTorrent pour la rapidité pour le transfert de données en assurant une compatibilité ascendante.

Mots-clés : KAD, BitTorrent, DHT, l'architecture P2P, performance, sécurité

1 Introduction

Since the beginning of Internet one of the most popular uses has been the file sharing between a set of users.

In early days this was achieved in a client-server model (i.e. a ftp server could have all the files and users could retrieve the files as they need it). After the big expansion of internet worldwide the resources needed to provide services as ftp servers became too expensive and a new set of technologies for sharing files needed to be developed, the solutions were found in P2P models where the users transfer the files between them.

Three different approaches were used: pure P2P[1] where users never need to interact with a centralized resource (i.e. gnutella, freenet), statically centralized hybrid P2P where users transfer the files between each other but coordination between peers is decided by a central known resource (i.e. eDonkey2000) and dynamically centralized P2P where a small subset of selected peers work as coordinators, usually called supernodes (i.e. Kazaa).

Since different countries got different policies about copyright, censorship and content distribution, there was controversy about the technology and the means it provided for people doing illegal distribution of content. The immediate response from some countries to the legal conflict was to block nodes that were doing some illegal activity. This decision impacted the network efficiency for every content (legal or not). As legislation is different according to the countries, users in a place where the content was legal got affected by other country legislation since users are a worldwide community. Last few years P2P file sharing communities were prosecuted in some countries because users indexed copyrighted material for distribution taking the local government actions to censor or block the central index, disabling the whole index in some cases by a single country action, (i.e. Kazaa [1] [2]). The networks had to evolve into pure P2P to avoid been compromised by conflicts of few nodes. One popular approach was to use DHTs[2] for indexing peers and sometimes files; the most widely deployed implantation is based on XOR metrics design and called Kademlia[5].

Recent research in the two most popular file-sharing networks[8] pointed that quality of searching is better in eMule networks thanks to the KAD indexing network, a variant of Kademlia which has two levels of indexation and is more solid against known attacks than BitTorrent DHTs which only has one indexing level and less security measures implemented. On the other hand, the BitTorrent network provides a fast transfer protocol in terms of overheads and propagation times.

The objective of this work is to build an integrated software that merges together the benefices of KAD indexation with BitTorrent transport protocol without loosing compatibility with previous eMule and BitTorrent clients.

The report is organized as follow. In section 2, we introduce what a file-sharing service should provide and presents the Kademlia DHT, the ed2k network and BitTorrent protocol. In section 3, we give a survey of the technologies available to be used in our project. In section 4, we review the requirements for the project and define some use cases. In section 5, we explain some approaches that were evaluated before implementation. In section 6, we show the flow of messages in a simple download using

[1] Also known as decentralized file sharing network in some bibliography.
[2] Distributed Hash Table.

original aMule client. In section 7, we define the changes that we proposed to achieve our goal and the expected flow of messages after our implementation. In section 8, we review the implementation details. In section 9, we explain a set of tests that were run in PlanetLab[3] with the new client and the results from those tests. In section 10, we enumerate some future work ideas. In the appendix, we provide a installation guide, a user guide, a troubleshooting for known issues and the documentation of the most relevant new classes implemented.

[3] http://www.planet-lab.org/

2 State of the art

2.1 File sharing service

File sharing service is composed of 3 steps: (1) a user searches for files that would suit his/her needs using keywords; (2) he/she selects which files from the search results he/she would like to obtain and ask for the list of sources; (3) at last he/she asks to the sources for the content of the files.

In client/server approach, the first step was done using some site like `www.yahoo.com`, `www.google.com`, `www.download.com`, etc. or links provided using email, irc or another channel. The second step was provided by the server who has the content itself or a secondary server providing a portal with the index of the content and resources location. The third step, the transfer, was done using ftp, http or some similar protocol.

There are different approaches in P2P with slightly variants. We will focus on just two technologies, eMule and BitTorrent.

For the first step, the eMule KAD network has a first level index that maps keywords to file identifiers while BitTorrent uses web sites that publish info-files containing the metadata needed for second step (pretty much like the client/server approach). The second step for eMule is handled by a second level index in KAD while BitTorrent uses some implementation of DHT that maps file metadata to sources. Both networks can also use centralized coordination servers[4] in some cases, these kind of use comes from backward compatibility before the use of DHTs for indexing. In the last step, each client uses it own protocol for transfer.

2.2 Kademlia for indexing

2.2.1 The theory

Kademlia is a general purpose P2P DHT based on XOR metrics[5]. In Kademlia every node has a 160 bit identifier randomly generated before joining the network. The distance between two nodes is defined as the XOR operation between two nodes' id. This operation in a fully populated tree of 160 bits is equivalent to the height of the smallest tree that contains both nodes.

For each i between 0 to 160, every <ip, port, id> triplets of nodes that are in distance 2^i and 2^{i+1} from itself are saved and called K-buckets. When a new node joins the network, it looks for itself and starts taking note of the closest nodes. It keeps record of the k closest known nodes(usually $k = 20$). These known nodes are used to route query messages to the other nodes.

To store information in Kademlia, a 160 bits key is created for this information and the closest nodes to the key will store data. To locate the stored data, the closest nodes to the key identifying the data should be found. To locate a node by its identifier you should ask the closest nodes you know, they will reply with the closest nodes they know and you iterate with the new closest nodes to the target until you reach the target node.

[4] Know in BitTorrent as trackers and in eMule as eDonkey servers

2.2.2 The eMule implementation - KAD

The eMule network uses a variation of Kademlia called KAD where the size of the keys is 128 bits (since eMule uses MD4 hashes as identifiers for files to keep compatibility with ed2k networks)[6]. KAD indexes has two levels to provide the whole file-sharing service process. The first level has an index that maps MD4 hashes of keywords to file identifiers (MD4 of the file contents) and the second level maps those file identifiers to the sources that share the file. In previous research, we have shown that KAD has a more reliable Kademlia implementation in terms of security than BitTorrent DHTs[7].

2.2.3 The BitTorrent implementations - Mainline and Vuze

There are two Kademlia implementations for BitTorrent's DHT: Mainline or Vuze, both incompatible between them. Both implementations uses 160 bits keys as indexes since BitTorrent applies uses as key the `sha1` hash of the info-file, which contains the metadata of the indexed content, also known as the info-hash. The BitTorrent DHT implementations have only one level of indexation that maps content identifiers to peers, that is why they need to access to the web or other channels to find the files identifiers to fulfill the first step in a file sharing service (as in section 2.1). The info-hash is used in place of file content's hash since BitTorrent transfers can handle several files as one single content.

2.3 Transfer protocols

2.3.1 ed2k

The ed2k protocol was developed originally for the eDonkey2000 network. It uses two queues for transfers: one to handle uploads and the other one to handle downloads. When a peer asks for a file to another peer, the peer providing the file replies saying how many users asked before and waits until finishing with those previous requests to start the transfer of the requested piece of data. Some improvements related to peer exchange and credit systems were implemented in some alternative clients (i.e. eMule[4]). The original eDonkey2000 client was shut down in 2006 because of legal actions [1] [2] and eMule took the central place in the scene when talking about standardization and protocol changes.

2.3.2 BitTorrent

BitTorrent users sharing the same content are grouped together into what is called "a swarm". The participants of a swarm follow a set of rules defined to maximize availability of content and redistribute the cost of upload and download between them. [3]

3 Survey of available technologies

3.1 eMule

The eMule client is an eDonkey2000 open source alternative for windows started in 2002. Since it release, eMule was expanding the original ed2k protocol with new non-official functionalities like peer exchange, credit systems and protocol obfuscation. [4] When legal prosecution started against the servers providing indexation of ed2k, eMule developed the KAD DHT based in Kademlia with 2 level index, one mapping file identifiers to peers, and one to map keywords to file identifiers so search for content can be done inside the network. [6] In 2006 legal actions shut down eDonkey2000 [1] [2] and eMule became the unofficial owner of ed2k protocol. The KAD guidelines for implementation are very strict and the network is not allowed to index anything but files shared. The source code is available under GPLv2+ in VC++ and can be obtained from `http://www.emule-project.com`.

3.2 aMule

The aMule client is a fork of xMule, an open source alternative multi-platform implementation compatible with eMule. It is the second most used client with active development these days. The architecture originally fully monolithic has been replaced with a modular one using ECP[5] to separate the core functionality from other components that provide different kinds of user interfaces. There is still the chance to get a monolithic build from same code.

The aMule components are:

- aMule Daemon: handles all the file-sharing work and can be connected to any user interface provided by aMule from a local or remote origin.
- aMule Remote GUI: similar to eMule GUI and implemented using wxWidgets library.
- aMule Web: a service that provides user interface using HTTP protocol, it's based on PHP code so it can be easily extended by webdevelopers.
- aMule Cmd: a simple user interface for command line.
- aMule: the monolithic version that resembles a Daemon + Remote Gui squashed together in a single binary.

The source code is available under GPLv2+ in C++ with wxWidgets library and can be obtained from `http://www.amule.org`.

3.3 BitTorrent

BitTorrent is the official client originally developed by Bram Cohen and the protocol specification is publicly available in BitTorrent site (`http://www.bittorrent.`

[5] External Connection Protocol

org). Changes to the BitTorrent protocol are proposed and decided by community. BitTorrent official client uses a DHT for index based on Kademlia called Mainline. The index maps SHA1 hashes of info-files to sources. The search for the file from keywords is handled outside of BitTorrent using the Web or other channels. In previous work BitTorrent transfer speed were compared with eMule in several scenarios showing that he download algorithm of BitTorrent performs better than KAD's algorithm[8]. BitTorrent was originally developed in Python in 2001 and licensed under MIT, since version 4 the code was closed by BitTorrent Inc, after several open-source alternatives were developed.

3.4 Vuze (formerly azureus)

Vuze is one of the most deployed alternatives to BitTorrent client, their developers had implemented a lot of extra features that are not in BitTorrent standard, also they provide a easy way to use interface for third party plug-ins. The Kademlia based DHT used by Vuze is not Mainline, neither it is compatible to it. The source code developed in Java is under a hybrid license having some components under proprietary and some under GPLv3, the open source pieces of code can be downloaded from http://dev.vuze.com/

3.5 rTorrent

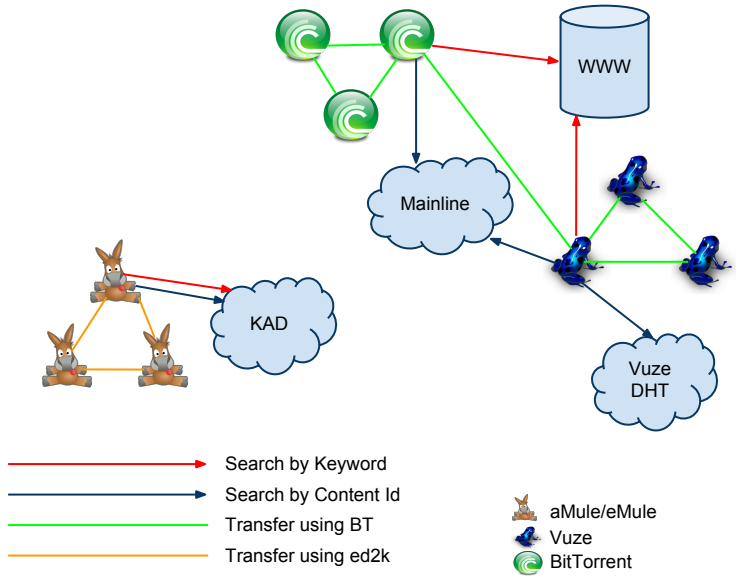
The rTorrent is a command line BitTorrent client whose core is also released as libtorrent (rakshasa) and used to implement other clients. The source code is available under GPLv2+ in C++ and can be obtained from http://libtorrent.rakshasa.no.

3.6 libtorrent-rasterbar

LibTorrent-rasterbar is a library implementing the BitTorrent protocol and a set of related tools needed to build a full client in few lines of code, including session management, plugins management, DHT access, metadata exchange and persistence of the content. The build from source code offers the option to build Python and Ruby bindings. The source code is available under BSD license in C++ and can be obtained from http://www.rasterbar.com/products/libtorrent/.

4 Requirements analysis

4.1 How different kind of clients interact today

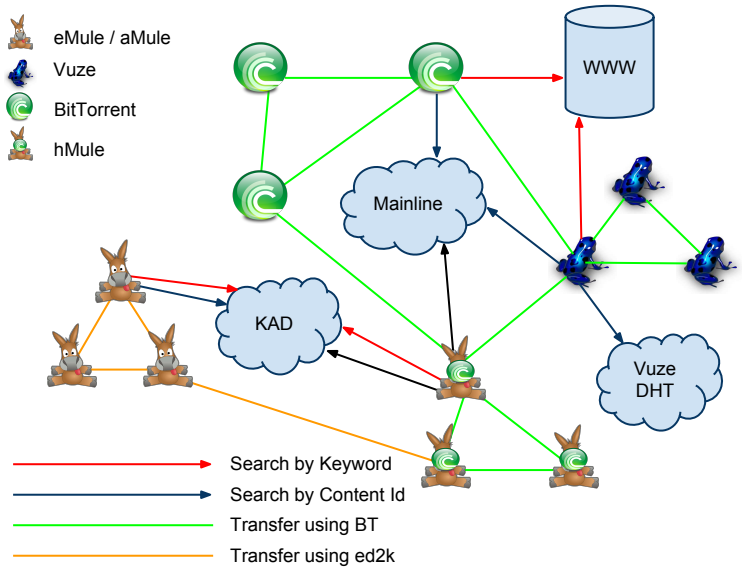


We can observe that Vuze and BitTorrent clients need to search by keyword in the web or another external resource to retrive the info-file or the info-hash. After they obtain the info-file or info-hash they lookup for a swarm to join in a tracker or a DHT. When looking in a DHT most the clients can use Mainline DHT, Vuze uses it own DHT implementation or Mainline up to configuration. When client find a swarm sharing the content needed, it will join and get the content. It's important to notice that the swarm can be joinned by any client using BitTorrent protocol, the only resource that is not shared between all of them is the DHT.

In the other side we got *Mule clients, they search by keyword in their DHT called KAD, then they ask for sources using again KAD. When a set of sources is obtained, *Mule client asks to every source for the metadata of the file and subscribes to the remote upload queue.

The difference in the order to obtain metadata is important, because it change assumptions in the protocols, i.e. BitTorrent info-file is unique metadata for a content, while the metadata for *Mule transfers can be as many as detected sources providing it.

4.2 How should different clients interact once hMule is fully implemented



The new client appears as a bridge between networks, it can access full content on both sides. After downloading a file from one of the networks it provides it to the other one.

4.3 List of requirements

- hMule should be able to join KAD to search and publish content.
- hMule should be able to transfer content to other hMule clients using BitTorrent transfer protocol.
- hMule should be retrocompatible:
  - It should be agnostic about transfer protocol when publishing content in KAD, meaning that a content should not have to be searched with different criterias depending of the transfer protocol to be used and KAD should return a valid result set for all kind of users.
  - It should share the published files to aMule/eMule clients using ed2k transfer protocol.
  - It should be able to join swarms with other BitTorrent clients.

4.4 Use cases

4.4.1 Request in hMule and hMule sources are provided by KAD

- Start a query in KAD with a keyword to get possible files.
- When KAD replies to the query, prompt the user to select the file to download.

- Start a query in KAD with the selected file to get sources.
- For each source
  - query source for metadata about the file.
  - if metadata includes an info-hash identifying the file as content in BitTorrent network, the download will be added to BitTorrent's transfer queues, a swarm will be joined using the peer providing the metadata as entry point.
  - add metadata to the ed2k queue.

4.4.2 Request in hMule and no hMule sources are provided by KAD

- Start a query in KAD with a keyword to get possible files.
- When KAD replies to the query, prompt the user to select the file to download.
- Start a query in KAD with the selected file to get sources.
- For each source
  - query source for metadata about the file.
  - add metadata to the ed2k queue.
- When transfer is complete, since no BitTorrent metadata was retrieved it should be generated and stored for future transfers.

4.4.3 Request in hMule from BitTorrent network having info-file or info-hash

- User introduces the info-file or info-hash.
- Download using torrent protocol, trackers and DHT if needed.
- When transfer is complete, compute KAD metadata and keywords.
- Publish in KAD.

4.4.4 Publish new content from hMule

- Compute BitTorrent info-file of the new content to be able to reply to BitTorrent requests for info-file from peer that only have the info-hash.
- Compute KAD metadata and keywords.
- Publish in KAD.

5 Possible approaches evaluated

To fulfill the requirements of the project several popular available technologies were studied and different approaches were evaluated to combine them into a solution for hMule implementation.

5.1 Expanding a BitTorrent client to use KAD

Using Vuze we could have implemented the indexing on KAD as a plugin for the search engine. When searching it should have to ask KAD first index level for files, then KAD should be asked for sources providing those files and finally transfer the file. The first problem with this approach is that KAD identifiers are smaller (128 bits) than BitTorrent's info-file which has no fixed length and can take several bytes. This issue happened to the developers of the BitTorrent DHTs too, that why they indexed using info-hashes of 160 bits. Knowing the metadata hash and sources providing the content, the sources can be asked for a copy of the full info-file to start the download. In the other side KAD reply has only 128 bits and there is no direct translation to compute an info-hash without lossing backward compatibility with KAD or BitTorrent.

To overcome these problems we thought to include the info-hash as a keyword when publishing the file, but some clients drop keywords or overrides them when other publication for the same file arrives (probably from an old client that doesn't provide this information). As we (at least until our client becomes popular enough) are a small minority in the network, there is a high probability that our info-hash gets eclipsed by other clients providing the same content.

The solution we came up for the issue matching KAD file identifiers to info-files was to implement a new protocol message that has a 128 bits identifier. The peers that share the file ask the info-file to clients identified as sources by KAD and if some client replies with the info-file, content transfer can be started using BitTorrent protocol.

This solution needs a new message and the implementation is not backward compatible to users using KAD right now since if we publish a content in KAD and we are the only client peer providing it, it will be available in index for all the *Mule users but will noone using an old client will be able to get the transfer. So using this approach only provides a shared index between networks that got garbage for both kind of users, those using BitTorrent can find transfers that only can be acquired using ed2k protocol and those using ed2k can find content that oly can be acquired using BtiTorrent protocol for transfers. This shows the need of ed2k protocol to get files when there is no BitTorrent sources, same to provide the content.

The implementation of ed2k protocol into the BitTorrent client is complex since there is no popular open source libraries providing it and should be stripped from some *Mule. We could try to filter results in our client so only those available for BitTorrent transfer appear, but filtering is complex because can not be done without asking to each source of each result what application version got installed. Asking to all sources of all results is overkill and if the file is provided or not by a hybrid client cannot be registered in KAD for same reasons than 160 bits hashes couldn't, also the use is unfair to other users of KAD who will index content they can not access.

Moving the KAD source code from eMule to a BitTorrent client (Vuze or any other) is

possible since other clients using KAD already had the need for that, but extracting the ed2k transfer protocol and session components is really difficult since eMule is tightly coupled between these functionalities and it's main loop.

5.2 Creating a hybrid client from the scratch using KAD, BitTorrent and ed2k libraries

Both libraries for BitTorrent were well designed and useful, but the libraries to provide KAD and ed2k functionalities were not available for production or needed too much work to be extracted from original implementation as libraries.

5.3 Expanding an eMule client to support BitTorrent as transfer protocol

Libraries providing BitTorrent were evaluated and both of them provide a good start for a BitTorrent client. The KAD protocol is working in any eMule client and the glue to exchange 128 bits for 160 identifier can be coded as a libtorrent extension or hooked to some eMule operation when bootstraping downloads.

The library extension approach major benefit is it will be used by several clients and they will provide the mapping service. However since KAD is not provided in the library if those clients don't connect to KAD network, they will not be suggested as sources so none of them will be asked for the translation.

We also know that eMule client asks every new source added for metadata, we can take advantage of this extending the eMule protocol so in the reply they also come up with the info-hash we need.

5.4 Choosen approach

We decided to expand an eMule-compatible client to support BitTorrent as an alternative transfer protocol.

5.4.1 T

he technologies we used

- We use aMule as eMule-compatible client since our experimentation infrastructure is based on Linux and also because the aMule project has a more modular architecture than eMule.
- We use libtorrent-rasterbar to provide the BitTorrent protocol and the tools, it has a complete high level interface to handle sessions, individual torrents and peers, and also it has access to low level interfaces to do adjustments when needed. The implementation is based in widely known Boost libraries that makes it easily portable to other operating systems and it provides interfaces to develop extensions. Extentions providing Mainline DHT, metadata exchange between peers, uPNP and IPFilters are already provided by the library.

6 aMule message flow study of a file transfer

As explained before, aMule uses EC Protocol to provide user interfaces and core functionality as separate applications.

The full list of messages defined in the EC protocol can be found in ECCodes.abstract file that renders the ECCodes.h file in the build process.

There were 84 messages defined at the time this project started. To simplify we will focus on a small subset of messages that are needed to start the download of a file after searching it in KAD from keywords.

- EC_OP_KAD_START: the daemon starts the KAD service and look for peers.
- EC_OP_SEARCH_START: the daemon starts a search of keywords in some network. In our case the param EC_SEARCH_KAD and at least one keyword should be used.
- EC_OP_SEARCH_PROGRESS: asks to the daemon for progress in the search since the search operation is asynchronous.
- EC_OP_SEARCH_RESULTS: After a search was started, this message asks for the list of results matching the criteria so far, asking more than one time can give different results until progress is 100%.
- EC_OP_DOWNLOAD_SEARCH_RESULT: when user decides what file to download from the results list, this message tells the daemon to start the download of the selected file.
- EC_OP_DWLOAD_QUEUE: asks to the daemon the status of the files in the download queue to provide the user feedback.

The aMule daemon when receiving the messages from the user interfaces replies the messages using the same EC protocol and interacts with the other peers in the network to fulfill the required actions. At the same time, the aMule daemon keeps track of the session, the transfer queues using the ed2k transfer protocol and the DHT using the KAD protocol.

To search for a file, the aMule daemon uses the CKademlia class defined in Kademlia.h and interacts with other peers using the following messages[6]:

- KADEMLIA_REQ message is sent to the known nodes whose identifier is closer to the hash of the keyword been looked up and iterates sending same message until no more closer to target nodes are found.
- SEARCH_REQ message is sent to those nodes that are known as the closer ones to the keyword's hash requested and they reply with the data they got about files that include the keyword in their metadata.

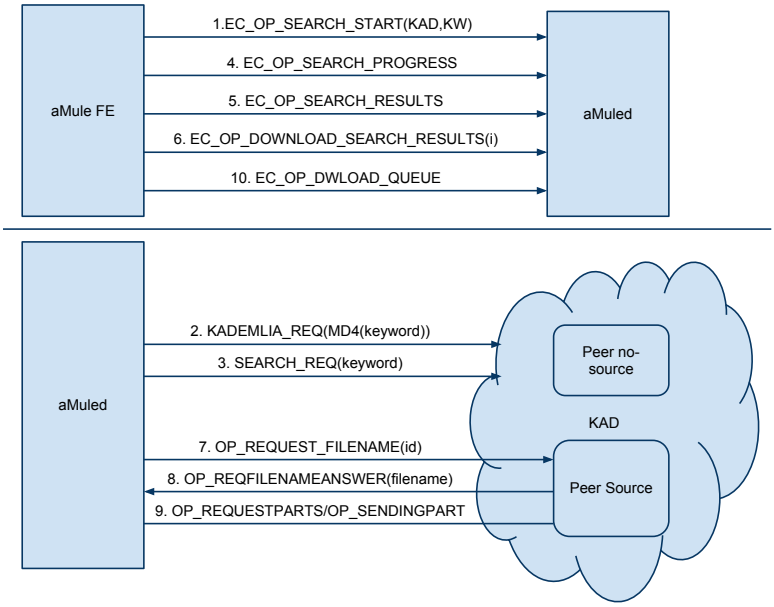
The aMule daemon caches the received results and passes them to the external interface using EC protocol when requested. When the file is selected by the user, the download is registered in the download queue.

Every time a new file is added to the queue, a SEARCH_REQ message is sent to find sources using the file id as search criteria. This search repeats periodically to update the source list of each file in the download queue. After sources are retrieved from KAD, aMule contacts each source asking for metadata about the file using the messages from ed2k protocol defined in TCP.h (i.e. OP_REQUESTFILENAME is sent to each source and is replied with a OP_REQFILENAMEANSWER message).

The sources register the client in their upload queue and notify it when it got the resources to transfer the requested data using the OP_SENDINGPART message.

After a file is downloaded (or if it is added to the sharing directory manually), it gets processed to extract metadata and all the extracted metadata is published, first in the nodes with the closer id to the hash of the keywords of the file to be published using the same procedure used for search, after the closest nodes to the target are found a STORE_KEYWORD message is send to those nodes and a STORE_FILE to the nodes that are closer to the file_id.

Next is a diagram of a simple flow of messages intervening in the download of a file in



aMule.

The numbers show a possible trace of the messages, some messages were suppressed to help the understanding, the order of them is not strict in the protocol and some times some of them can be swapped as it is the case for (8) and (9).

At top we got the interaction between the back end and the front end using the ECP: (1) we start a search; (4) we ask for search progress; (5) we obtain the search results; (6) we select what to dowload; (10) we monitor the download queue status.

At the bottom we got the interaction between the backend and other peers: (2,3) we send a requests to KAD; (7) we request metadata for the file to each source; (8) the sources reply with metadata about our requested file; (9) we request a subcription to the remote upload queue in a source.

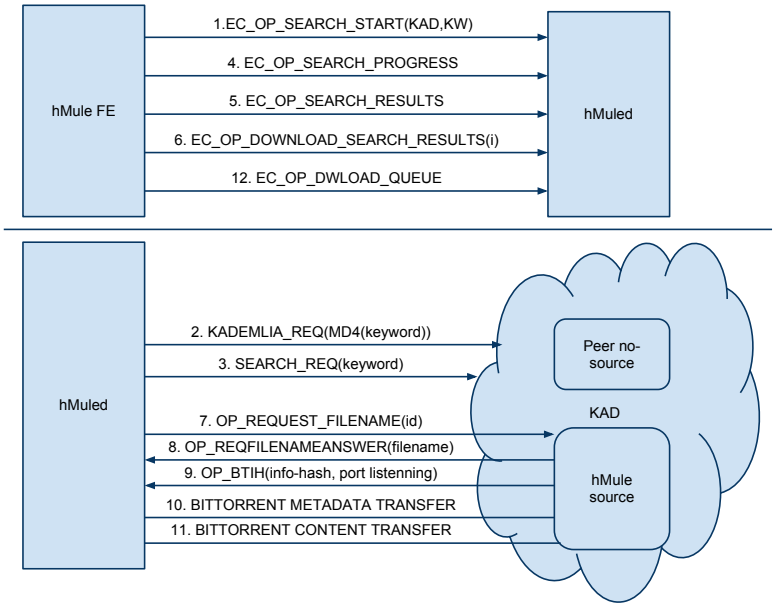
7 Changes proposed to the aMule client

We got to retrieve the BitTorrent metadata from peers that know it. This metadata can be retrieved after download is added to download queue when sources are consulted about the metadata of the file to be transferred. If one of the known nodes replies with the info-hash, that also means that it provides BitTorrent services in the client and the port where this service is running should be retrieved to start a metadata exchange request using the already implemented methods in the libtorrent-rasterbar. There is no need to implement a message to request the info-hash since OP_REQUIRE_FILENAME is sent to all known sources and expects several asynchronous replies with different kinds of metadata as user comments, media tags, etc.

7.1 Protocol changes

- The new message providing the info-hash and the port number where to listen for BitTorrent protocol messages will be implemented as part of eMule extended protocol and the message will be named OP_BTIH.
- Some extra messages should be defined in the EC protocol to check Torrent specific stats, add new download starting from info-files, info-hash or magnet links, disable or enable Mainline DHT support.

Next is a diagram of a simple flow of messages intervening in the download of a file in the modified client hMule.



This diagram is based in previous aMule diagram in section 6.

The changes are between messages (8) to (12): (9) the new implemented message providing info-hash and port which is listening for BitTorrent protocol, is sent from sources having hMule; (10) Info-file is obteined using BitTorrent metadata exchange extension; (11) The swarm is joinned and the content is transfered.

7.2 Application changes needed

- To reply with OP_BITH an internal dictionary that maps eMule file ids to BitTorrent metadata hashes should be maintained internally. This dictionary should persist between sessions since torrent metadata coming from different sources has different metadata hash for the same content and we don't want to lose the original swarms.
- To provide backward compatibility, when a file was fully transferred using ed2k protocol, the info-file is generated and it is added to the dictionary. So users with hMule client can retrieve info-file and info-hash to use BitTorrent protocol. An external torrent file should be able to be added to the session and after fully transferred using BitTorrent protocol the file should be published in KAD and the dictionary updated.
- A class should be defined to wrap the libtorrent-rasterbar uses and a coordinator between the transfer protocols should be defined to avoid overlap of efforts and competition. This coordinator should be able to decide which protocols to use and fix the decision if it was not the best. The switch of strategies should be easy so the client can be used for experiments.
- A translation between wxWidgets structures and Boost ones should be provided in some cases.
- Building scripts should be modified to provide a non hybrid build and to detect the needed libraries.

8 Current implementation details

The glue between aMule and libtorrent-rasterbar was implemented defining a namespace torrent that contains a class Torrent to handle the wrapping and a class TorrentMuleMapping that keeps track of the metadata relations.

The TorrentMuleMapping class is internally represented as 3 unordered maps to index the relations for search and a vector for iteration of the relations. The relations have 4 fields, a CMD4hash[6], a SHA1hash[7], a boost::filesystem::path that should point the metadata torrent file and a status of download enum. Several getters and update methods are defined to access the container, including constant iteration.

The Torrent class is implemented as a singleton and contains internally the libtorrent session, an instance of TorrentMuleMapping and methods to wrap libtorrent behavior as createMetadataForFile (which generates the torrent metadata for a known file). Torrent class also initializes and calls the TorrentStrategy used to select protocols for transfer, which is selected in the app's configuration file.

8.1 Initializing and shutting down the application

Main application class, called amuleApp, has an OnInit method and an OnExit method that are called at start and end of the application run. There is also specific methods for subapplication customization of those methods.

When application is launched and OnInit is called, aMule reads all the preferences of the user from the config file, initializes the Download and Upload queues, reads a set of metadata files called PartMet files to register the current downloads metadata in the download queue. There some extra activities handled in the OnInit method based in configuration, i.e. join edonkey servers and start running KAD, if any of them is enabled as soon as possible they start looking sources for all the downloads queued.

After all ports for aMule connections are open, the Torrent class instance is created setting the ports for BitTorrent connections (by default port 7000, but tries others if that one is taken). The Torrent object uses two configuration options to select where to persist the info-files and where to persist the dictionary betweeen sessions. The strategy to use is read from the config file only once, so if you want to change it, session should be restarted.

After the download queue are completly loaded, the method refreshMetadata in the Torrent class is called. This method iterates the TorrentMuleMap to load all the metadata known from previous sessions then checks the torrent directory defined in the config for info-files that are not mapped yet and load them, this works as a dropping directory for those not using the amulecmd UI.

Then aMule calls reloadFiles in SharedFilesList class to update the metadata of the shared files, this method was modified so it calls addFilesFromDirectory method in Torrent object to update the metadata of torrent files and update information in the TorrentMuleMap also.

When the application shutdown calls OnExit method all is persisted and shutdown, this

[6] Type of identifier used for files in aMule
[7] Type of identifier used for the BitTorrent content

method was modified to include info-files and dictionary in the list of persisted things between session and also shutdown the BitTorrent session.

8.2 Starting a download from KAD

When a user searches KAD and finds a file he/she wants to download, the file is added to the download queue, in KAD same file identifier was used for files that are transmitted using ed2k protocol for transfer than those transferred using BitTorrent since the content is the same and we want to keep backward compatibility. If a user having an old client finds the file in KAD and doesn't support BitTorrent transfers, the file should be transferred using ed2k protocol without the need of a new search.

After the file is added to DownloadQueue and KAD provides a list of sources a message OP_REQUEST_FILENAME is sent to each source to request for metadata of the file to be transferred.

When the source uses the extended eMule protocol, it will reply with several message with different information as idv3, comments, filename, etc., if using our implementation we will reply one more message, named OP_BTIH that contains the info-hash and the listening port for BitTorrent interactions.

When our client receives metadata from sources, it will process it using the processFileInfo method in DownloadClient class, this one was modified to check for OP_BTIH messages, if any was received the information contained is passed to the Torrent class using the addTorrentUsingSHA1AndPeer method, this adds the ip and port of the peer that provided the information and the info-hash of the file and with this data libtorrent-rasterbar starts a torrent connection to the peer and asks for a full metadata exchange.

The main loop of the application has a second thread to process asynchronous tasks, every time a loop is completed the download queue and upload queue gets a call to the process method, a call to the process method in Torrent class was added so asynchronous alerts from libtorrent-rasterbar can be attended and TorrentStrategy can do the switch of protocols if needed.

After file is fully downloaded in torrent, the process method calls for a recheck of the file in the aMule download queue this validates the AICH hashes and flag the file as completely downloaded, moves the file out of the temp directory and shares it.

8.3 Strategies for referee

Two basic strategies were implemented, one that saves metadata provided by others, provides the OP_BTIH replies to other peers requesting it and transfers content to other but doesn't use BitTorrent protocol to download content, this strategy was named NoTorrentStrategy and can be selected in config as strategy=0, the second was defined so at the moment metadata for BitTorrent is fully obtained the download is handled completely by BitTorrent transfer protocol, this one was named AllwaysFallToTorrentStrategy and can be selected in config as strategy=1.

8.4 Starting a download from .torrent

When a file .torrent with the metadata for a torrent file is provided from the web or other source, it needs to be added to the libtorrent-rasterbar session to be downloaded and after it is fully downloaded be published in KAD.

The file path can be provided in the add command of amulecmd or dropping the file in the torrent directory configured, also add command can receive a BitTorrent magenet link and download if mainline DHT is enabled.

Mainline DHT can be enabled using connect mainline command in aMulecmd.

8.5 Sharing a new file

When a new file is added in the sharing directory and reload button or reload command is executed and the reload method in SharedFilesList is called, this already calls the reload method in Torrent class and everything gets registered.

8.6 Other changes

Autoconf scripts have been modified to check for the new libraries needed in the software, a –enable-torrent option has been added to the ./configure script that separated the code been compiled for hybrid version than the one used to build a original aMule version with no hybrid code in it.

9 Experimentation

To test functionalities, we used in PlanetLab where several daemons were installed. We deployed 8 nodes with hMuled, 4 of them were configured to use NoTorrentStrategy and the other 4 were configured to use AlwaysFallToTorrentStrategy, also we deployed 4 nodes using aMuled, 2 built from aMule original code and 2 bould from hMule using building paramenter –disable-torrent.

After deploying all the nodes, random content was generated in each node and shared. Using aMulecmd or aMulegui nodes were accessed to do search in KAD of the other nodes published content and all files were transferred. We validated that, when possible, the content was transferred using BitTorrent protocol and over 95% of the time the metadata for BitTorrent was obtained before any content transfer using ed2k protocol, meaning no thrash was produced in the Temp directories.

After test that all our nodes can share the content between them, we did a second experiment where popular content was searched in KAD network and downloaded by all our peers. At first we were searching in each node a few minutes later than the previous one and when last one started to download it was verified that it was using only BitTorrent resources.

Another showed that when all the downloads start simultaneously the swarm doesn't get joined by all the peers who could since the join is done when a peer asks for metadata to others and at the time the download is started none of them can provide an info-file. Anyway, after at least one source finish the download, if a new peer asks for the same content afterwards it will gather all those peers together in a swarm, this happens because the new peer will trigger the exchange of bootstrap message in all of the sources, then it will get the info-file from the source that already completed the transfer and when others check this peer as a source will obtain the info-file from it.

Two more test were ran where peers running hMule downloaded content from BitTorrent networks using magnet links (with mainline DHT) or info-files (downloaded from the web) was succesful. After publish the content in KAD, other peers searched for it, they found the content and started the download, these new peers downloaded joined the original swarm with any other BitTorrent client keeping backward compatibility. Notice that files downloaded from BitTorrent networks doesn't get indexed in KAD until full download and to keep new downloaders joining the peers in the original swarm, the original info-file should be provided and not a new computed one (as it happens).

Tests also showed that when using AlwaysFallToTorrent, after a file started download using BitTorrent and all peers providing the content goes down the transfer stills, but after a new peer connects (or some of the previous reconnects) the transfer is resumed fine because the new peers are detected in the aMule network and notified to the BitTorrent session.

10 Future work

10.1 Definition of referee strategies based on experimentation

New experiments should be created with several nodes to check which are reasonably better parameters for the defined strategy. Also more interesting strategies should be implemented where offering the chance to switch back to ed2k network when the download could be improved.

10.2 Hybrid download using sparse files

One of the main problems to switch protocol in a download is to keep the downloaded data, aMule and libtorrent-rasterbar provide a format for the saving file called "sparse file" that reserves the whole file required space at the start of the download fill it all with 0s and starts writing the received content in the place it should go. Using this format it is reasonable to switch protocols on the same file without problems, it will only require a full recheck of the pieces hash at the start to know what the other protocol provided.

There are a few issues to solve for that, both libraries use different ways to open the file so the ways should be unified or the file should be split in 2 or more pieces so it can be opened separately, if split the point of split should be wisely selected, second issue will be to avoid overlap of work, this includes the creation of a map of pieces overlapping since pieces in torrent and ed2k have different size and will not match 1 to 1, neither 1 to an integer in most the cases.

10.3 Multifile multiprotocol content support and collections

Torrent metadata of a content can include several files and even directories, most of the popular torrent file downloaded from the web got more of 1 file, in the other hand eMule provides a kind of file name .emulecollection containing a list of MD4 hashes of a set of files that when downloaded is read and all the files identified into the files start download. It is difficult to match the 2 approaches (specially because the emule approach doesn't have concept of directory).

The proposed approach for this issue will be to read the shared files directory. If a directory is there, it will be shared in torrent as a directory structure in a single torrent file and in ed2k as a emulecollection, if the file is downloaded from torrent an emulecollection file will be defined when the full directory structure is downloaded and if downloaded from an emulecollection a directory will be created when download finishes with all the files and be shared as a torrent file that contains several files with one single directory in the structure.

10.4 Protection against Fake SHA1

Since there is no math conversion between the ed2k identifier and the info-hash of a file we trust user to provide the translation, those user can fake the sha1 so we start downloading something that doesn't match the original indexed content, so when a

download finishes in using BitTorrent protocol we validate that the content's MD4 hash matches the original ed2k identifier and throw an error, this error should be handled.

10.5 Fake SHA1 earlier detection

In *Mule networks there is a set of hashes to detect when a file is not going to match the expected content to early detect, it is called AICH validation and similar set of hashes is produced in BitTorrent when the Torrent is Merkled. We should be able to find when some validated against Merkle pieces pull together cover a full piece of ed2k content and this piece fails to validate against AICH, when this happens we are downloading content that matches the provided info-hash but doesn't match the ed2k identifier, enough to say that info-hash is fake and we don't have to keep downloading it.

10.6 Torrent Fast Resume support

LibTorrent-rasterbar provides some tools to skip rehashing and management overheads that should be used.

10.7 Thread safeness

Before creating more complex strategies the operations that would be used for them should be checked for needing of locks, specially iteration operations, because most the code only locks at start and finish of the method and that can be not enough in some cases.

References

[1] Metro-goldwyn-mayer v. grokster ltd., 2004.

[2] Metro-goldwyn-mayer studios inc. v. grokster, ltd., 2005.

[3] B. Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72. Citeseer, 2003.

[4] Y. Kulbak and D. Bickson. The emule protocol specification. *eMule project, http://sourceforge. net*, 2009.

[5] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. *Peer-to-Peer Systems*, pages 53–65, 2002.

[6] D. Mysicka and R. Wattenhofer. Reverse engineering of emule-an analysis of the implementation of kademlia in emule, 2006.

[7] J.P. Timpanaro, T. Cholez, I. Chrisment, and O. Festor. Bittorrent's mainline dht security assessment. In *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on*, pages 1–5. IEEE.

[8] J.P. Timpanaro, T. Cholez, I. Chrisment, and O. Festor. When kad meets bittorrent-building a stronger p2p network. 2011.

Appendices

Appendix A Install Guide

The procedure to install hMule is pretty straight foward in any Linux or BSD system[8].

Step by step install for Ubuntu 11.04 systems.

- Using synaptics, aptitude or apt-get install the following packages.
  - libgtk2.0-dev
  - libssl-dev
  - autoconf
  - libtool
  - libcrypto++-dev
  - autopoint
  - bison
- Download wxWidgets 2.9.2 from `http://www.wxwidgets.org` and unpack it.
- Open a terminal, step into the wxWidget's code directory and run the following commands.

```
./configure --enable-debug --enable-unicode \
        --disable-shared --prefix=/usr
make
make install
```

- Download Boost 1.47+ from `http://www.boost.org` and unpack it.
- Open a terminal, step into the Boost's code directory and run the following commands.

```
./bootstrap.sh --with-libraries=system,filesystem --prefix=/usr
./b2
./b2 install
```

- Download libtorrent-rasterbar 0.16 from `http://www.rasterbar.com/products/libtorrent/` and unpack it.
- Open a terminal, step into the libtorrent-raterbar's code directory and run the following commands, if you are trying to find a bug there is options in configure that provide logging of every operation done by the library that can be really useful.

```
./configure --enable-debug --prefix=/usr
make
make install
```

[8]The software is suppose to build an run in windows also, but was not tested by us in those platforms

- Download hmule and unpack it.
- Open a terminal, step into the hmule's code directory and run the following commands.

```
./autogen.sh
./configure --enable-torrent
make
make install
```

- There is interesting options that could be added to the configure command in hmule.
  - –enable-amule-daemon builds amuled, a deamon to run amule that can be controlled remotely with amulecmd or amule-gui (best option to use in planetlab).
  - –enable-amulecmd builds amulecmd, a command line interface to control an amuled.
  - –enable-amule-gui builds amule-gui, a gtk GUI for amuled.
  - –disable-monolithic prevents the building of the full monolithic client (useful to add when using any of the previous options).

Appendix B User Guide

This guide doesn't intend to fully cover hMule usage, but to introduce the user to the new functionallities provided by this implementation, this guide can be complemented by the original user guide of aMule provided in `http://wiki.amule.org`.

Appendix B.1 New configuration parameters

In aMule configuration file, usually in /.aMule/amule.conf, we defined a new category called Torrent which has 2 configuration options.

- TorrentDir that defines a directory where the info-files will be saved.
- Strategy that defines the strategy used todecide which protocol to use, so far we got NoTorrentStrategy (using Strategy=0) or AlwaysFallToTorrent (using Strategy=1).

Appendix B.2 Connection to Mainline DHT

To connect to or disconnect from Mainline DHT two new commands are provided in aMuleCmd.

- connect mainline
- disconnect mainline

Appendix B.3 New files

All files been downloaded using BitTorrent protocol will be stored in Temp directory with the name provided by the info-file until fully transfered, when transfer is complete it will be moved to Incomming directory. All metadata obtainned for BitTorrent transfers will be saved in as info-files into Torrent directory.

Two new files are recorded in base configuration directory, MTBT.dat contains the MuleToTorrentMap persistance in a plain text format and lt-state.dat contains the state of the session dump benconded as described in libtorrent-rasterbar documentation's section about persisting sessions (`http://www.rasterbar.com/products/libtorrent/manual.html#load-state-save-state`).

Appendix C Known Issues

- Localization fails and crash when openning amuled in some systems. To disable localization export LC_ALL=C before running amuled and you can skip the error at the price of using non localized version.
- Some times when trying to find for updates an http error appears and crashes the full app, restart the app, it tries once every few days and if didn't crash in the first few minutes it is safe since app will not try again until new instance is started.
- When shutting down the application some times it tries to remove some objects more than once creating a core dump, it happens after saving all data so doesn't affect next run.

More known bugs can be found in the `http://bugs.amule.org/`, but only these show up in the experimentation we have done.

Appendix D Most relevant new classes documentation

Appendix D.1 torrent::CTorrent Class Reference

`#include <Torrent.h>`

Public Member Functions

- void StartTorrentSession ()
- void EndTorrentSession ()
- const int GetPort () const
- void RefreshMetadata ()
- void CreateMetadataForFile (const CMD4Hash fileID, const CPath &filename, const CPath &storeDir)
- void CreateMetadataForFile (const CMD4Hash fileId, boost::filesystem::path filename, boost::filesystem::path storeDir)

- bool HasBTMetadata (const CMD4Hash fileId) const
- std::string GetBTIHAsString (const CMD4Hash fileId)
- void AddDownloadUsingSHA1AndPeer (const CMD4Hash fileId, std::string SHA1Hash, std::string peerIP, int port)
- bool AddDownloadUsingTorrentFile (boost::filesystem::path file)
- bool AddDownloadFromMagnet (std::string magnet)
- void Process ()
- uint64 GetCompletedSize (CMD4Hash fileId)
- void StartMainline ()
- void StopMainline ()
- bool IsMainlineConnected ()
- void LoadUnregisteredTorrents ()
- boost::filesystem::path SaveTorrent (libtorrent::create_torrent &t, boost::filesystem::path &filename)
- void GiveUp (CMD4Hash)
- virtual ~CTorrent ()

Static Public Member Functions

- static CTorrent & GetInstance ()

  constant identifying the SwitchToTheMostUsablePeersStrategy.

Static Public Attributes

- static const int **NO_BT** = 0
- static const int ALLWAYS_FALL_TO_BT = 1

  constant indetifying the NoBtStrategy.

- static const int SWITCH_TO_THE_MOST_USABLE_PEERS = 2

  constant identifying the AlwaysFallToBTStrategy.

Appendix D.1.1 Detailed Description

Wrap class for torrent functionalities.

It is implemented as a singleton, to get the instance use: Torrent::getInstance() method.

Appendix D.1.2 Constructor & Destructor Documentation

Appendix D.1.2.1 virtual torrent::CTorrent::~CTorrent ( ) `[virtual]`

Destructor

Appendix D.1.3 Member Function Documentation

Appendix D.1.3.1 bool torrent::CTorrent::AddDownloadFromMagnet (std::string *magnet*)

Add a download to Torrent queue having a torrent magnet link.

Parameters

magnet	A magnet link for torrent.

Returns

true if the file was registered successfully in tmm.

Appendix D.1.3.2 void torrent::CTorrent::AddDownloadUsingSHA1AndPeer (const CMD4Hash *fileId*, std::string *SHA1Hash*, std::string *peerIP*, int *port*)

Add a download to Torrent queue knowing SHA1 and a peer.

If the file is not in the queue, it is added and the peer is associated hoping to get a full metadata exchange from it, else the peer is added to the set of known peers for that file.

Parameters

fileId	An MD4 identifier for a file known in aMule.
SHA1Hash	The SHA1 provided by KAD that identifies the file's torrent metadata.

Appendix D.1.3.3 bool torrent::CTorrent::AddDownloadUsingTorrentFile (boost::filesystem::path *file*)

Add a download to Torrent queue having a .torrent file.

The file is copied into the torrent files directory and an unregistered torrents call is made.

Parameters

file	The complete filename for a info-data torrent file.

Returns

true if the file was registered successfully in tmm.

Appendix D.1.3.4 void torrent::CTorrent::CreateMetadataForFile (const CMD4Hash *fileID*, const CPath & *filename*, const CPath & *storeDir*)

Creates torrent Metadata for a file.

Parameters

fileId	The MD4 aMule identifier of the file.
filename	The name of the file.
storeDir	Path where the file is stored

Appendix D.1.3.5 void torrent::CTorrent::CreateMetadataForFile (const CMD4Hash *fileId*, boost::filesystem::path *filename*, boost::filesystem::path *storeDir*)

Creates torrent Metadata for a file.

Parameters

fileId	The MD4 aMule identifier of the file.
filename	The name of the file.
storeDir	Path where the file is stored

Appendix D.1.3.6 void torrent::CTorrent::EndTorrentSession ( )

Closes all torrent conections and destroy session object.

Appendix D.1.3.7 std::string torrent::CTorrent::GetBTIHAsString (const CMD4Hash *fileId*)

Obtain the SHA1 content identifier for torrent knowing its muleId.

Parameters

fileId	An MD4 identifier for a file known in aMule.

Returns

SHA1 torrent content identifier.

Appendix D.1.3.8 uint64 torrent::CTorrent::GetCompletedSize (CMD4Hash *fileId*)

Given a file get an estimation of how much of it was downloaded.

Parameters

fileId	The MD4 hash identification of a file in the aMule Download Queue.

Appendix D.1.3.9 static CTorrent& torrent::CTorrent::GetInstance ( ) [static]

constant identifying the SwitchToTheMostUsablePeersStrategy.

Get the instance.

Returns

the single CTorrent instance.

Appendix D.1.3.10 const int torrent::CTorrent::GetPort ( ) const

Port for the torrent incomming connections.

This method is used to bootstrap Torrent traffic with known KAD peers.

Appendix D.1.3.11 void torrent::CTorrent::GiveUp (CMD4Hash)

If downloading in bt this file, just give up.

Parameters

fileId	A MD4 aMule identifier of a file.

Appendix D.1.3.12 bool torrent::CTorrent::HasBTMetadata (const CMD4Hash *fileId*) const

Checks if BT Metadata is known for a file identified with a MD4 aMule identifier.

Parameters

fileId	A MD4 aMule identifier of a file.

Appendix D.1.3.13 bool torrent::CTorrent::IsMainlineConnected ( )

Check if Mainline DHT service is runnning.

Returns

true if the Mainline DHt is running.

Appendix D.1.3.14 void torrent::CTorrent::LoadUnregisteredTorrents ( )

Checks the ThePrefs::TorrentDir for info-files that are known yet.

This method iterates the dir, and queue all unknown torrents into actual session for download. All Torrents in the Torrent metadata directory are loaded, it works as a Drop box for Torrents.

Appendix D.1.3.15 void torrent::CTorrent::Process ( )

Process decides the way content should be downloaded.

This method is suppossed to be called in interval ticks as same as CDownloadQueue::Process It reads Torrent and aMule queues and decides what to start, pause, stop in each of them based in the selected TorrentStrategy.

Appendix D.1.3.16 void torrent::CTorrent::RefreshMetadata ( )

Iterates the TorrentMuleMap to load/save metadata.

Loads all the torrents info-files that are known but not loaded in session yet and saves all the torrent info-files of those that were received but not persisted yet.

Appendix D.1.3.17 boost::filesystem::path torrent::CTorrent::SaveTorrent (libtorrent::create_torrent & *t*, boost::filesystem::path & *filename*)

Saves torrent metadata into file.

Parameters

t	create_torrent instance of the file to be persisted.
filename	The filename of the content.

Returns

filename of the saved info-file torrent metadata.

Appendix D.1.3.18 void torrent::CTorrent::StartMainline ( )

Starts Mainline DHT service.

Appendix D.1.3.19 void torrent::CTorrent::StartTorrentSession ( )

Starts the torrent session.

Torrent sessions are a container for all the shared and downloading torrents and handle all the connections some ports are opened when starting the session and if prefered Mainline DHT is joined too.

Appendix D.1.3.20 void torrent::CTorrent::StopMainline ( )

Stops Mainline DHT service.

The documentation for this class was generated from the following file:

- Torrent.h

Appendix D.2 torrent::MD4ToHash Struct Reference

`#include <TorrentMuleMapping.h>`

Public Member Functions

- std::size_t **operator()** (CMD4Hash const &v) const

Appendix D.2.1 Detailed Description

Hash function for MuleIdToMetadataRelation indexing

The documentation for this struct was generated from the following file:

- TorrentMuleMapping.h

Appendix D.3 torrent::SHA1ToHash Struct Reference

`#include <TorrentMuleMapping.h>`

Public Member Functions

- std::size_t **operator()** (libtorrent::sha1_hash const &v) const

Appendix D.3.1 Detailed Description

Hash function for BTIdToMetadataRelation indexing

The documentation for this struct was generated from the following file:

- TorrentMuleMapping.h

Appendix D.4 torrent::filenameToHash Struct Reference

`#include <TorrentMuleMapping.h>`

Public Member Functions

- std::size_t **operator()** (boost::filesystem::path const &v) const

Appendix D.4.1 Detailed Description

Hash function for InfoFileToMetadataRelation indexing

The documentation for this struct was generated from the following file:

- TorrentMuleMapping.h

Appendix D.5 torrent::CTorrentMuleMapping Class Reference

`#include <TorrentMuleMapping.h>`

Public Types

- typedef std::vector< MetadataRelation * >::const_iterator const_iterator

Public Member Functions

- MetadataRelation * UpdateMetadata (CMD4Hash muleId, libtorrent::sha1_hash torrentId, boost::filesystem::path torrentFile)
- MetadataRelation * UpdateMetadata (CMD4Hash muleId, libtorrent::sha1_hash torrentId)
- void Erase (CMD4Hash muleId)
- MetadataRelation * UpdateMetadata (libtorrent::sha1_hash torrentId, boost::filesystem::path torrentFile)
- void SetDownloading (CMD4Hash muleId)
- void SetDownloading (libtorrent::sha1_hash torrentId)
- void SetSharing (CMD4Hash muleId)
- void SetSharing (libtorrent::sha1_hash torrentId)
- void SetRemoved (CMD4Hash muleId)
- void SetRemoved (libtorrent::sha1_hash torrentId)
- bool HasTorrentPath (CMD4Hash muleId) const
- bool HasTorrentPath (libtorrent::sha1_hash torrentId) const
- bool HasTorrentPath (boost::filesystem::path torrentFile) const
- bool HasBTIH (CMD4Hash muleId) const
- bool HasBTIH (libtorrent::sha1_hash torrentId) const
- bool HasBTIH (boost::filesystem::path torrentFile) const
- bool HasMuleIH (CMD4Hash muleId) const
- bool HasMuleIH (libtorrent::sha1_hash torrentId) const
- bool HasMuleIH (boost::filesystem::path torrentFile) const
- const boost::filesystem::path & GetTorrentPath (CMD4Hash muleId) const
- const boost::filesystem::path & GetTorrentPath (libtorrent::sha1_hash torrentId) const
- const libtorrent::sha1_hash & GetBTIH (CMD4Hash muleId) const
- const libtorrent::sha1_hash & GetBTIH (boost::filesystem::path torrentFile) const
- const CMD4Hash & GetMuleIH (libtorrent::sha1_hash torrentId) const
- const CMD4Hash & GetMuleIH (boost::filesystem::path torrentFile) const
- bool IsDownloading (CMD4Hash muleId)
- bool IsDownloading (libtorrent::sha1_hash torrentId)
- bool IsDownloading (boost::filesystem::path &)
- bool IsSharing (CMD4Hash muleId)
- bool IsSharing (libtorrent::sha1_hash torrentId)
- bool IsSharing (boost::filesystem::path &)
- bool WasRemoved (boost::filesystem::path &)

- const_iterator begin () const
- const_iterator end () const
- void Load (boost::filesystem::path filename)
- void Save (boost::filesystem::path filename)
- virtual ∼CTorrentMuleMapping ()

Appendix D.5.1 Detailed Description

CTorrentMuleMapping is a container for the MetadataRelations.

Appendix D.5.2 Member Typedef Documentation

Appendix D.5.2.1 typedef std::vector<MetadataRelation∗>::const_iterator torrent::CTorrentMuleMapping::const_iterator

Iterator type, only const iteration is allowed.

Appendix D.5.3 Constructor & Destructor Documentation

Appendix D.5.3.1 virtual torrent::CTorrentMuleMapping::∼CTorrentMuleMapping ( ) `[virtual]`

Destructor

Appendix D.5.4 Member Function Documentation

Appendix D.5.4.1 const_iterator torrent::CTorrentMuleMapping::begin ( ) const

Standard iterator begin

Appendix D.5.4.2 const_iterator torrent::CTorrentMuleMapping::end ( ) const

Standard iterator end

Appendix D.5.4.3 void torrent::CTorrentMuleMapping::Erase (CMD4Hash *muleId*)

Erases a MetadataRelation.

Parameters

muleId	A MD4 identifier of an aMule known file.

Appendix D.5.4.4 const libtorrent::sha1_hash& torrent::CTorrentMuleMapping::GetBTIH (CMD4Hash *muleId*) const

Obtain the SHA1 content identifier for torrent knowing its muleId.

Warning

This method assumes you asking for a valid tuple, check before use.

See also

HasBTIH

Parameters

muleId	An MD4 identifier for a file known in aMule.

Returns

SHA1 torrent content identifier.

Appendix D.5.4.5 const libtorrent::sha1_hash& torrent::CTorrentMuleMapping::GetBTIH (boost::filesystem::path *torrentFile*) const

Obtain the SHA1 content identifier for torrent knowing the filename of its info-file.

Warning

This method assumes you asking for a valid tuple, check before use.

See also

HasBTIH

Parameters

torrentFile	The name of a info-file containing torrent metadata.

Returns

SHA1 torrent content identifier.

Appendix D.5.4.6 const CMD4Hash& torrent::CTorrentMuleMapping::GetMuleIH (libtorrent::sha1_hash *torrentId*) const

Obtain the MD4 aMule file identifier for torrent knowing the torrent SHA1 content identifier.

Warning

This method assumes you asking for a valid tuple, check before use.

See also

HasMuleIH

Parameters

torrentId	An SHA1 identifier for a content known in Torrent.

Returns

MD4 identifier for a file known in aMule.

Appendix D.5.4.7 const CMD4Hash& torrent::CTorrentMuleMapping::GetMuleIH (boost::filesystem::path *torrentFile*) const

Obtain the MD4 aMule file identifier for torrent knowing the filename of its torrent info-file.

Warning

This method assumes you asking for a valid tuple, check before use.

See also

HasMuleIH

Parameters

torrentFile	The name of a info-file containing torrent metadata.

Returns

MD4 identifier for a file known in aMule.

Appendix D.5.4.8 const boost::filesystem::path& torrent::CTorrentMuleMapping::GetTorrentPath (CMD4Hash *muleId*) const

Obtain the filename of a torrent info-file knowing its muleId.

Warning

This method assumes you asking for a valid tuple, check before use.

See also

HasTorrentPath

Parameters

muleId	An MD4 identifier for a file known in aMule.

Returns

filename of the torrent info-file.

Appendix D.5.4.9 const boost::filesystem::path& torrent::CTorrentMuleMapping::GetTorrentPath (libtorrent::sha1_hash *torrentId*) const

Obtain the filename of a torrent info-file knowing its torrentId.

Warning

This method assumes you asking for a valid tuple, check before use.

See also

HasTorrentPath

Parameters

torrentId	An SHA1 identifier for a content known in Torrent.

Returns

filename of the torrent info-file.

Appendix D.5.4.10 bool torrent::CTorrentMuleMapping::HasBTIH (boost::filesystem::path *torrentFile*) const

Check if the info-file has some related torrent SHA1 identifier.

Parameters

torrentFile	A filename of where the info-file metadata is saved.

Returns

true if the info-file has some related torrent SHA1 identifier.

Appendix D.5.4.11 bool torrent::CTorrentMuleMapping::HasBTIH (CMD4Hash *muleId*) const

Check if the muleId has some related torrent SHA1 identifier.

Parameters

muleId	A MD4 identifier for a file known in aMule.

Returns

true if the muleId has some related torrent SHA1 identifier.

Appendix D.5.4.12 bool torrent::CTorrentMuleMapping::HasBTIH (libtorrent::sha1_hash *torrentId*) const

Check if the BT info-hash is declared

Parameters

torrentId	A SHA1 identifier for a content known in Torrent.

Returns

true if the BT info-hash is declared.

Appendix D.5.4.13 bool torrent::CTorrentMuleMapping::HasMuleIH (CMD4Hash *muleId*) const

Check if the muleId is declared.

Parameters

muleId	A MD4 identifier for a file known in aMule.

Returns

true if the muleId is declared.

Appendix D.5.4.14 bool torrent::CTorrentMuleMapping::HasMuleIH (libtorrent::sha1_hash *torrentId*) const

Check if the info-file has some related MD4 aMule identifier.

Parameters

torrentId	A SHA1 identifier for a content known in Torrent.

Returns

true if the info-file has some related MD4 aMule identifier.

Appendix D.5.4.15 bool torrent::CTorrentMuleMapping::HasMuleIH (boost::filesystem::path *torrentFile*) const

Check if the info-file has some related MD4 aMule identifier.

Parameters

torrentFile	A filename of where the info-file metadata is saved.

Returns

true if the info-file has some related MD4 aMule identifier.

Appendix D.5.4.16 bool torrent::CTorrentMuleMapping::HasTorrentPath (CMD4Hash *muleId*) const

Check if the muleId has some related torrent metadata info-file.

Parameters

muleId	A MD4 identifier for a file known in aMule.

Returns

true if the muleId has some related torrent metadata info-file.

Appendix D.5.4.17 bool torrent::CTorrentMuleMapping::HasTorrentPath (boost::filesystem::path *torrentFile*) const

Check if the torrentFile is declared

Parameters

torrentFile	A filename of where the info-file metadata is saved.

Returns

true if the torrentFile is declared.

Appendix D.5.4.18 bool torrent::CTorrentMuleMapping::HasTorrentPath (libtorrent::sha1_hash *torrentId*) const

Check if the torrentId has some related torrent metadata info-file.

Parameters

torrentId	A SHA1 identifier for a content known in Torrent.

Returns

true if the torrentId has some related torrent metadata info-file.

Appendix D.5.4.19 bool torrent::CTorrentMuleMapping::IsDownloading (CMD4Hash *muleId*)

Check if a file is in Downloading state knowing its MD4 aMule identifier.

Parameters

muleId	An MD4 identifier for a file known in aMule.

Returns

true if the file is in state Downloading.

Appendix D.5.4.20 bool torrent::CTorrentMuleMapping::IsDownloading (boost::filesystem::path &)

Check if a content is in Downloading state knowing its torrent filename.

Parameters

torrentFile	The name of a info-file containing torrent metadata.

Returns

true if the file is in state Downloading.

Appendix D.5.4.21 bool torrent::CTorrentMuleMapping::IsDownloading (libtorrent::sha1_hash *torrentId*)

Check if a content is in Downloading state knowing its SHA1 torrent identifier.

Parameters

torrentId	An SHA1 identifier for a content known in Torrent.

Returns

true if the file is in state Downloading.

Appendix D.5.4.22 bool torrent::CTorrentMuleMapping::IsSharing (CMD4Hash *muleId*)

Check if a file is in Sharing state knowing its MD4 aMule identifier.

Parameters

muleId	An MD4 identifier for a file known in aMule.

Returns

true if the file is in state Sharing.

Appendix D.5.4.23 bool torrent::CTorrentMuleMapping::IsSharing (libtorrent::sha1_hash *torrentId*)

Check if a content is in Sharing state knowing its SHA1 torrent identifier.

Parameters

torrentId	An SHA1 identifier for a content known in Torrent.

Returns

true if the file is in state Sharing.

Appendix D.5.4.24 bool torrent::CTorrentMuleMapping::IsSharing (boost::filesystem::path &)

Check if a content is in Sharing state knowing its torrent filename.

Parameters

torrentFile	The name of a info-file containing torrent metadata.

Returns

true if the file is in state Sharing.

Appendix D.5.4.25 void torrent::CTorrentMuleMapping::Load (boost::filesystem::path *filename*)

It loads a persisted instance of the class from filename

Appendix D.5.4.26 void torrent::CTorrentMuleMapping::Save (boost::filesystem::path *filename*)

It persists actual instance into filename.

Appendix D.5.4.27 void torrent::CTorrentMuleMapping::SetDownloading (CMD4Hash *muleId*)

Set state of a relation as Downloading.

Parameters

muleId	A MD4 identifier for a file known in aMule.

Appendix D.5.4.28 void torrent::CTorrentMuleMapping::SetDownloading (libtorrent::sha1_hash *torrentId*)

Set state of a relation as Downloading.

Parameters

torrentId	A SHA1 identifier for a content known in Torrent.

Appendix D.5.4.29 void torrent::CTorrentMuleMapping::SetRemoved (CMD4Hash *muleId*)

Set state of a relation as Removed.

Parameters

muleId	A MD4 identifier for a file known in aMule.

Appendix D.5.4.30 void torrent::CTorrentMuleMapping::SetRemoved (libtorrent::sha1_hash *torrentId*)

Set state of a relation as Removed.

Parameters

torrentId	A SHA1 identifier for a content known in Torrent.

Appendix D.5.4.31 void torrent::CTorrentMuleMapping::SetSharing (CMD4Hash *muleId*)

Set state of a relation as Sharing.

Parameters

muleId	A MD4 identifier for a file known in aMule.

Appendix D.5.4.32 void torrent::CTorrentMuleMapping::SetSharing (libtorrent::sha1_hash *torrentId*)

Set state of a relation as Sharing.

Parameters

torrentId	A SHA1 identifier for a content known in Torrent.

Appendix D.5.4.33 MetadataRelation* torrent::CTorrentMuleMapping::UpdateMetadata (CMD4Hash *muleId,* libtorrent::sha1_hash *torrentId,* boost::filesystem::path *torrentFile*)

Updates or creates a MetadataRelation.

It looks for MetadataRelations partially containing the data provided and replaces the null data so the Metadata Relation increases knowledge, it can't remove info or modify it, only increment it. Usually this Update is used when a file starts sharing or when a download started in KAD and some peer provided the info-file metadata for Torrent transfer. (both cases content metadata is completely known).

Parameters

muleId	A MD4 identifier of an aMule known file.
torrentId	A SHA1 identifier of a torrent known content.
torrentFile	A filename of where the info-file metadata is saved to load it again in next session.

Returns

Pointer to the created or updated Metadata Relation.

Appendix D.5.4.34 MetadataRelation* torrent::CTorrentMuleMapping::UpdateMetadata (CMD4Hash *muleId,* libtorrent::sha1_hash *torrentId*)

Creates a MetadataRelation.

It creates a MetadataRelation with the aMule Identifier and the BT info-hash, but keeps the torrentFile as NULL. Usually this Update is used when a download started in KAD and some peer provided BTIH, but the info-file metadata for Torrent transfer was not acquired yet.

Parameters

muleId	A MD4 identifier of an aMule known file.
torrentId	A SHA1 identifier of a torrent known content.

Returns

Pointer to the created or updated Metadata Relation.

Appendix D.5.4.35 MetadataRelation* torrent::CTorrentMuleMapping::UpdateMetadata (libtorrent::sha1_hash *torrentId,* boost::filesystem::path *torrentFile*)

Creates a MetadataRelation.

It creates a MetadataRelation with the BT info-hash and the filename of the info-file containing torrent metadata and keeps the torrentFile as NULL. Usually this Update is used when a download started in from mainline or .torrent file and the MD4 identifier for aMule is not known yet.

Parameters

torrentId	A SHA1 identifier of a torrent known content.
torrentFile	A filename of where the info-file metadata is saved to load it again in next session.

Returns

Pointer to the created or updated Metadata Relation.

Appendix D.5.4.36 bool torrent::CTorrentMuleMapping::WasRemoved (boost::filesystem::path &)

Check if a content Was Removed knowing its torrent filename.

Parameters

torrentFile	The name of a info-file containing torrent metadata.

Returns

true if the file is in state removed.

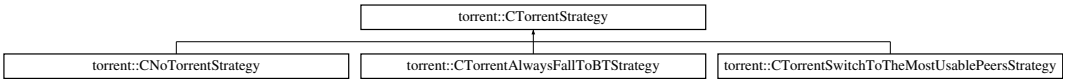
The documentation for this class was generated from the following file:

- TorrentMuleMapping.h

Appendix D.6 torrent::CTorrentStrategy Class Reference

`#include <TorrentStrategy.h>`

Inheritance diagram for torrent::CTorrentStrategy:



Public Member Functions

- CTorrentStrategy (libtorrent::session ∗torrentSession, CTorrentMuleMapping ∗mapping)
- virtual void Process ()=0
- uint64 GetCompletedSize (CMD4Hash fileId)
- virtual void GiveUp (CMD4Hash fileId)
- virtual ∼CTorrentStrategy ()

Protected Member Functions

- void ProcessAlert (std::auto_ptr< libtorrent::alert >)
- virtual void OnReceivedMetadata (libtorrent::metadata_received_alert ∗)
- virtual void OnTorrentResumed (libtorrent::torrent_resumed_alert ∗)
- virtual void OnFinishedDownload (libtorrent::torrent_finished_alert ∗)
- void ValidateDownloads ()
- CTorrentStrategy ()

  This Vector keeps record of those hashes awaiting validation and how many tries for validation were done.

Protected Attributes

- uint32 **m_lastTimeProcessWasRun**
- uint32 m_lastTimeAlertsWereProcessed

  last time the Process method was called for this strategy.

- uint32 m_lastTimeValidationQueueProcessed

  last time the Alerts for torrent asynchronous task were processed.

- libtorrent::session * m_ts

  last time the Validation Queue was processed.

- CTorrentMuleMapping * m_tmm

  pointer to the active torrent session.

- std::vector< std::pair< CMD4Hash, int > > m_validationQueue

  pointer to the Metadata Relation for torrent and amule.

Appendix D.6.1 Detailed Description

Abtract class for Transfer protocol selection strategies.

Different strategies can be created to handle the way it is decided to choose transfer protocol and when to switch to the other one.

Appendix D.6.2 Constructor & Destructor Documentation

Appendix D.6.2.1 torrent::CTorrentStrategy::CTorrentStrategy (libtorrent::session * *torrentSession*, CTorrentMuleMapping * *mapping*)

Constructor.

Parameters

torrentSession	A pointer to the running torrent session.
mapping	A pointer to the MetadataRelations between Torrent and aMule.

Appendix D.6.2.2 virtual torrent::CTorrentStrategy::~CTorrentStrategy ( ) `[virtual]`

Destructor

Appendix D.6.2.3 torrent::CTorrentStrategy::CTorrentStrategy ( ) [protected]

This Vector keeps record of those hashes awaiting validation and how many tries for validation were done.

Prevent default constructor

Appendix D.6.3 Member Function Documentation

Appendix D.6.3.1 uint64 torrent::CTorrentStrategy::GetCompletedSize (CMD4Hash *fileId*)

Given a file get an estimation of how much of it was downloaded.

Parameters

fileId	The MD4 hash identification of a file in the aMule Download Queue.

Appendix D.6.3.2 virtual void torrent::CTorrentStrategy::GiveUp (CMD4Hash *fileId*) [virtual]

Removes any internal representation of torrents that are no longer needed.

Parameters

fileId	The MD4 hash identification of a file in the aMule Download Queue.

Reimplemented in torrent::CTorrentSwitchToTheMostUsablePeersStrategy.

Appendix D.6.3.3 virtual void torrent::CTorrentStrategy::OnFinishedDownload (libtorrent::torrent_finished_alert *) [protected, virtual]

Process the received finished download alert

If not overrided by inheritance, it will just do nothing.

Appendix D.6.3.4 virtual void torrent::CTorrentStrategy::OnReceivedMetadata (libtorrent::metadata_received_alert *) [protected, virtual]

Process the received metadata alert

If not overrided by inheritance, it will save the received metadata in a torrent file and exit.

Appendix D.6.3.5 virtual void torrent::CTorrentStrategy::OnTorrentResumed (libtorrent::torrent_resumed_alert *) [protected, virtual]

Process the received torrent resume alert

If not overrided by inheritance, it will just do nothing.

Appendix D.6.3.6 virtual void torrent::CTorrentStrategy::Process ( ) [pure virtual]

Process decides the way content should be downloaded.

Abstract method that should read Torrent and aMule queues and decides what to start, pause, stop in each of them, also decides when alerts from asynchronous operations in torrent should be handled and how should they be handled.

Implemented in torrent::CTorrentAlwaysFallToBTStrategy, torrent::CNoTorrentStrategy, and torrent::CTorrentSwitchToTheMostUsablePeersStrategy.

Appendix D.6.3.7 void torrent::CTorrentStrategy::ProcessAlert (std::auto_ptr< libtorrent::alert >) [protected]

Processes the alerts coming from asynchronous torrent tasks.

Appendix D.6.3.8 void torrent::CTorrentStrategy::ValidateDownloads ( ) [protected]

Check the torrents that were finished if any of them failed the MD4 check.

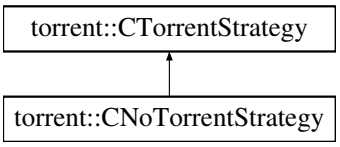
The documentation for this class was generated from the following file:

- TorrentStrategy.h

Appendix D.7 torrent::CNoTorrentStrategy Class Reference

`#include <TorrentStrategy.h>`

Inheritance diagram for torrent::CNoTorrentStrategy:



Public Member Functions

- CNoTorrentStrategy (libtorrent::session ∗torrentSession, CTorrentMuleMapping ∗mapping)
- void Process ()
- virtual ∼CNoTorrentStrategy ()

Appendix D.7.1 Detailed Description

A strategy that doesn't use Torrent Protocol for transfers, just noop and back.

Appendix D.7.2 Constructor & Destructor Documentation

Appendix D.7.2.1 torrent::CNoTorrentStrategy::CNoTorrentStrategy (libtorrent::session ∗ *torrentSession,* CTorrentMuleMapping ∗ *mapping*)

Constructor Overrides CTorrentStrategy so nothing is done.

Parameters

torrentSession	A pointer to the running torrent session.
mapping	A pointer to the MetadataRelations between Torrent and aMule.

Appendix D.7.2.2 virtual torrent::CNoTorrentStrategy::∼CNoTorrentStrategy ( ) `[virtual]`

Destructor

Appendix D.7.3 Member Function Documentation

Appendix D.7.3.1 void torrent::CNoTorrentStrategy::Process ( ) `[virtual]`

Does nothing.

See also

CTorrentStrategy::Process and CTorrentStrategy::ProcessAlerts

Implements torrent::CTorrentStrategy.

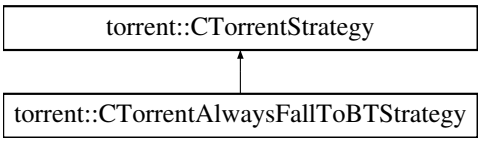
The documentation for this class was generated from the following file:

- TorrentStrategy.h

Appendix D.8 torrent::CTorrentAlwaysFallToBTStrategy Class Reference

`#include <TorrentStrategy.h>`

Inheritance diagram for torrent::CTorrentAlwaysFallToBTStrategy:



Public Member Functions

- CTorrentAlwaysFallToBTStrategy (libtorrent::session *torrentSession, CTorrentMuleMapping *mapping)
- void Process ()
- virtual ~CTorrentAlwaysFallToBTStrategy ()

Appendix D.8.1 Detailed Description

A strategy that Falls to Torrent transfer protocol as soon as BTIH is known

This strategy checks if any peer provided a Info Hash for Torrent transfer and as soon as it is transfered pauses the download in the aMule download queue and fully transfer the content using BitTorrent transfer protocol. It is not a very smart strategy but is great for debugging. Check CTorrentStrategy comments.

Appendix D.8.2 Constructor & Destructor Documentation

Appendix D.8.2.1 torrent::CTorrentAlwaysFallToBTStrategy::CTorrentAlwaysFallToBTStrategy (libtorrent::session * *torrentSession*, CTorrentMuleMapping * *mapping*)

Constructor Same as CTorrentStrategy constructor so far

Parameters

torrentSession	A pointer to the running torrent session.
mapping	A pointer to the MetadataRelations between Torrent and aMule.

Appendix D.8.2.2 virtual torrent::CTorrentAlwaysFallToBTStrategy::~CTorrentAlwaysFallToBTStrategy ( ) [virtual]

Destructor

Appendix D.8.3 Member Function Documentation

Appendix D.8.3.1 void torrent::CTorrentAlwaysFallToBTStrategy::Process ( ) [virtual]

When new BT info has was received moves the download to Torrent Protocol and start ProcessAlerts

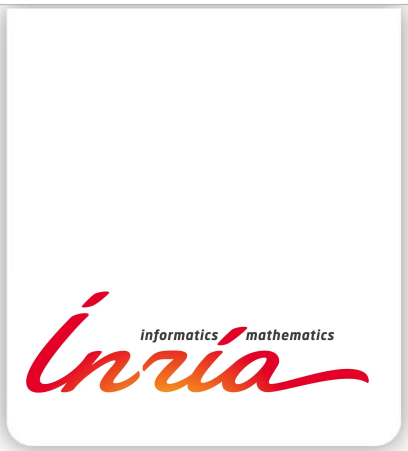
See also

CTorrentStrategy::Process and CTorrentStrategy::ProcessAlerts

Implements torrent::CTorrentStrategy.

The documentation for this class was generated from the following file:

- TorrentStrategy.h



RESEARCH CENTRE
NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-0803