

A Scalable Web 2.0 Platform

Fabian van der Werf




TU Delft

Delft University of Technology

A Scalable Web 2.0 Platform

Master's Thesis in Computer Science

Parallel and Distributed Systems group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Fabian van der Werf

27th November 2007

Author

Fabian van der Werf

Title

A Scalable Web 2.0 Platform

MSc presentation

TODO GRADUATION DATE

Graduation Committee

prof. dr. ir. H. J. Sips (chair) Delft University of Technology

dr. ir. J.A. Pouwelse Delft University of Technology

ir. dr. D. H. J. Epema Delft University of Technology

Abstract

Since a few years, there has been a social revolution on the Internet referred to as Web 2.0. Socialized web services have become very popular as shown by examples like YouTube, Flickr, and Wikipedia. The drawback of popularity is the increased operating costs. Especially public sharing sites like YouTube and Flickr receive many visitors a day and require much storage and bandwidth to host all the videos and photos of every user. Simultaneously, peer-to-peer technology has also gained much popularity.

This thesis describes the conducted research in providing Web 2.0 functionality with scalable peer-to-peer technology. We extend Tribler, an existing peer-to-peer client with a social community, with capabilities to leverage the wide collection of items that is already offered by existing Web 2.0 sites. Furthermore, we ease the process of publishing with peer-to-peer technology for the end user.

Preface

This is the report of my thesis project on combining Web 2.0 functionality with the peer-to-peer client Tribler. This research has been carried out at the Section Parallel and Distributed Systems of the Faculty Electrical Engineering, Mathematics, and Computer Science of Delft University of Technology.

First of all I would like to thank my supervisor Johan Pouwelse for his guidance and support during the research. I also thank Ivaylo Haratcherev for creating a Windows version of the Web 2.0 Browser, Maarten ten Brinke for helping me with the GUI of the Web 2.0 Browser, Arno Bakker for merging my work into the main development branch, and Jelle Roozenburg for explaining parts of the Tribler internals.

Fabian van der Werf

Delft, The Netherlands
27th November 2007

Contents

Preface	v
1 Introduction	1
1.1 Web 2.0	2
1.2 Peer-to-peer networks	4
1.2.1 BitTorrent	5
1.2.2 Tribler	7
1.3 Outline	8
2 Problem Description	9
2.1 Scalability and Robustness	9
2.2 Our Aim	11
2.2.1 YouTube	11
2.2.2 Tribler	12
2.2.3 Our Vision	13
2.3 Related Work	14
2.3.1 Vuze	14
2.3.2 Joost	15
2.3.3 Babelgum	17
3 Design And Implementation	19
3.1 Software Development Process and Functionality	20
3.1.1 Single-threaded prototype	20
3.1.2 Multithreaded, multi-site prototype – Web 2.0 Browser	20
3.1.3 Integration of Video Browsing with P2P	22
3.2 Web 2.0 Browser	23
3.2.1 Architecture	23
3.2.2 GUI design	26
3.2.3 Web 2.0 Interfaces	28
3.2.4 Ratings	31
3.2.5 Download Manager	31
3.3 Tribler Integration	32
3.3.1 Integration with Tribler	32

3.3.2	Decentralized Tracking	34
4	Experiments And Evaluation	37
4.1	Web 2.0 Interfaces	37
4.2	Real world usage	41
4.2.1	Datasets	41
4.2.2	Average Rating Requests	42
4.2.3	Ratings	45
4.3	Khashmir	46
4.3.1	Khashmir behaviour	47
4.3.2	Khashmir Timeout	48
4.4	End-to-End Video	51
5	Conclusions and Future Work	53
5.1	Conclusions	53
5.2	Future Work	53
5.2.1	Performance	53
5.2.2	Functionality	54

Chapter 1

Introduction

Traditionally, the World Wide Web (WWW) was a collection of a static web pages which an Internet surfer could look up. Over time this has changed a lot. The last few years, data and services offered via the WWW have been *socialized*. The social functionality is referred to as Web 2.0[6]. There is no fixed checklist according which a web service can be qualified as Web 2.0 or not, and there is still much debate to what Web 2.0 exactly comprises. Web 2.0 is not a new technology, though it is often associated with new technologies, e.g., Ajax and RSS, that enables web developers to create rich user experience web applications. Web 2.0 is a new approach to creating web applications in which the role of users is more important. Users are no longer just consumers of information but are participants in so-called *online communities* in which two-way communication occurs.

Web 2.0 applications have gained a significant share in top ranking web sites. Table 1.1 shows the global top 10 of popular sites as measured by Alexa. The Alexa ranking is not a true reflection of the popularity of the sites, however, it gives a good indication of the significance and success of Web 2.0 applications. Six out of ten sites in the top 10 are Web 2.0 sites, i.e., Google, YouTube, MySpace, Orkut, Wikipedia, and Tencent QQ.

Most of the Web 2.0 applications are being served by a single server or several computer centers, and consequently lack scalability and robustness. Our aim is to create a scalable Web 2.0 platform without relying on any central components. The users must be able to upload videos, photos, etc., and navigate easily through the uploaded content as Web 2.0 applications allow. To take advantage of the wealth of content already available on sites like YouTube and Flickr, integration of such sites is necessary.

This section gives an introduction into Web 2.0 and peer-to-peer technology. Finally, an outline of the remainder of this thesis is given.

Rank	Web Site	URL
1.	Yahoo!	www.yahoo.com
2.	Microsoft Network (MSN)	www.msn.com
3.	Google	www.google.com
4.	YouTube	www.youtube.com
5.	Windows Live	www.live.com
6.	MySpace	www.myspace.com
7.	Baidu.com	www.baidu.com
8.	Orkut	www.orkut.com
9.	Wikipedia	www.wikipedia.org
10.	Tencent QQ	www.qq.com

Table 1.1: Global top 10 ranking sites assessed by Alexa on July 5th, 2007.

1.1 Web 2.0

The term Web 2.0 was first used by O'Reilly Media. Figure 1.1 shows a meme map that was developed at a brainstorm session. The major concept get the users more involved in the web applications. The meme map contains six core competencies which are briefly discussed below.

These six competencies do not form a checklist to check whether an application is Web 2.0 or not, but are characteristics of Web 2.0 applications.

Services, Not Packaged Software Services are delivered to the end user through an ordinary web browser which is available on most platforms. Consequently, services are accessible from any platform without the need to port it each specific platform as is required with ordinary software. Furthermore, services do not have release schedules but are improved continuously. Services follow the “release early, release often” principle.

Architecture of Participation The WWW used is to be one way traffic of information, i.e., users only consumed information and did not produce anything. Web 2.0 applications let users participate and let users add value to the Web 2.0 applications.

Cost-effective scaling Most Web 2.0 applications are still not cost-effective scalable. These applications are being served by a client-server architecture just as the world's first web page. This non-scaling architecture causes video sites to have to degrade the quality of their videos to keep the necessary bandwidth needed limited. There exist a few scaling Web 2.0 applications, and they are mostly peer-to-peer file sharing. However, most of these file sharing applications only harness the bandwidth of the users and have a total lack of social interaction.

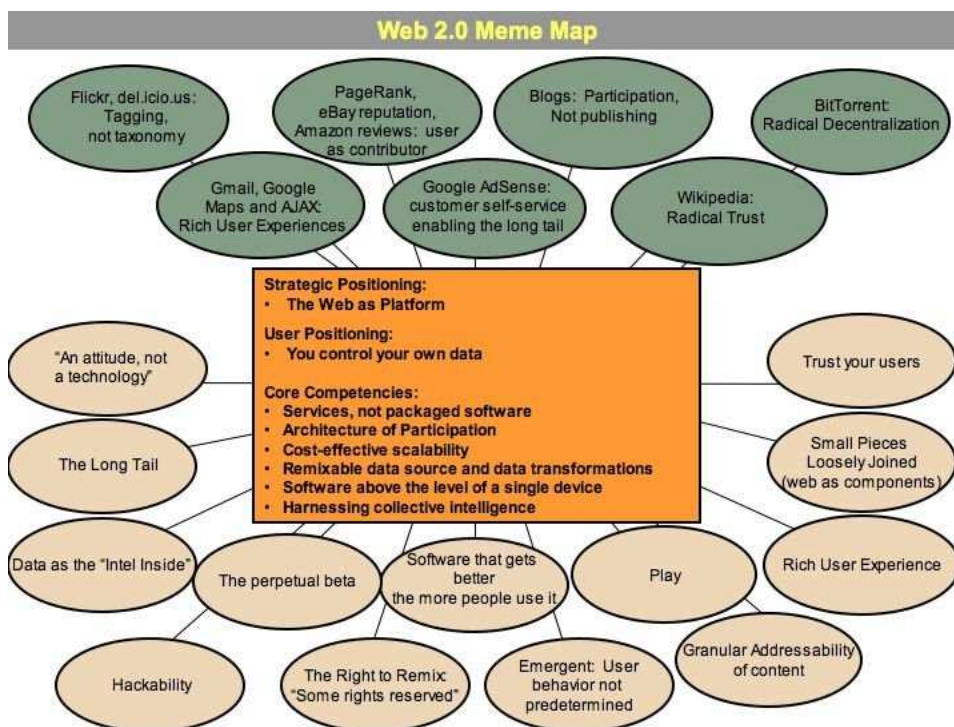


Figure 1.1: Web 2.0 meme map by O'Reilly Media

Remixable data source and data transformations Remixable data source and data transformations means that data and services provided by a Web 2.0 application should be easy to integrate with other web applications. In practice, this results in Web 2.0 applications having a documented API through which their data and services are exposed.

Software above the level of a single device Software is no longer limited to the PC platform; the Internet is the new platform and it comes with new possibilities. For example, Microsoft Office is restraint to a single PC. The Web 2.0 counterpart could be Google Docs & Spreadsheets. Documents are stored online, and accessible and editable from any PC equipped with an Internet browser. Google Docs & Spreadsheets also enables document sharing in order to allow multiple persons working on the same document.

Harnessing collective intelligence The classic example of the web application that harnesses the collective intelligence is Wikipedia. The articles of which Wikipedia constitutes are written by volunteers around the world, and can be written by anyone with Internet access. The accuracy of Wikipedia may not be as high as of an encyclopedia that is composed by experts, though, in most cases it is good enough.

1.2 Peer-to-peer networks

Peer-to-Peer (P2P) systems are networks in which work is performed by all participating nodes instead of a few powerful machines. Furthermore, nodes participating in a P2P system are autonomous and each node has its own objectives (e.g., downloading or publishing a certain file). The desire to accomplish these objectives, is the only reason to participate in a P2P system.

P2P networks have gained a lot of popularity. Research shows that Internet traffic is dominated by P2P traffic[4]. In 2007, 30% (daytime) to 70% (nighttime) of all Internet traffic was P2P traffic. P2P networks are mostly associated with sharing large files such as movies, TV shows, music, and software. However, other P2P applications exist such as telephony and TV.

P2P systems compared to centralized systems have two major advantages: scalability and robustness. P2P systems without any central components scale automatically as the user base grows. Each participating node donates bandwidth, computing power, and storage to the system. Hence, as a new node enters the system, the total capacity of systems increases. A P2P system will always have enough resources as long as each user donates at least as much as it consumes. Some P2P systems, like BitTorrent, have mechanisms in place to enforce this.

Second, P2P systems have no central components and are therefore much more robust than centralized systems. Centralized systems contain one or more single points of failure like a web server, an application server, a database, etc. If one of these components fail, the entire system is down and thereby affecting all its users.

In P2P systems the main components are the peers, and if a peer fails then only the resources contributed by that peer are no longer available.

This section gives a brief overview of the P2P systems that are relevant to this thesis, BitTorrent and Tribler.

1.2.1 BitTorrent

BitTorrent is a popular P2P file sharing system designed and implemented by Bram Cohen. There are tens of different BitTorrent clients.

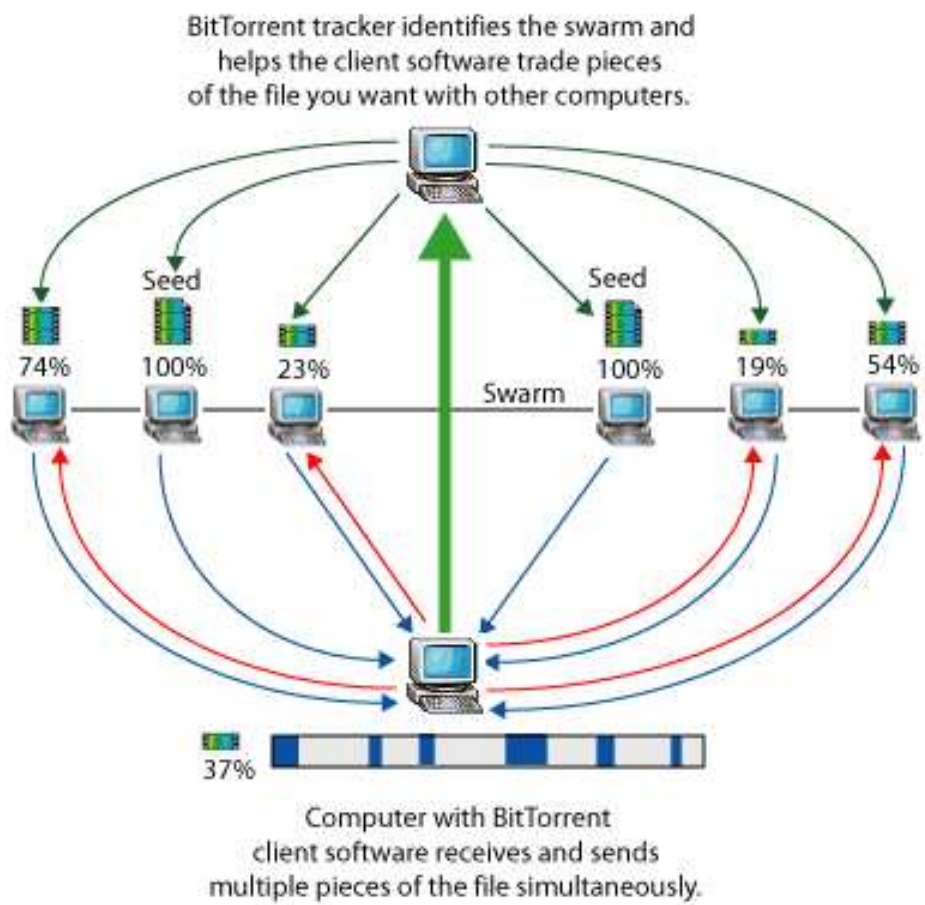
The BitTorrent system consists of leechers, seeders, and trackers. Leechers are peers that want to download the file, and may already have downloaded the file partially. Seeders are peers who already have a complete copy of the file and sharing it with leechers to help the distribution. Each torrent has one or more trackers which serve as a meeting point for seeders and leechers.

When a peer decides to distribute a file with BitTorrent, it has to create a so-called .torrent file. This .torrent file stores metadata and provides enough information for other BitTorrent peers to download the published file, like filename, length, and tracker URL. The .torrent BitTorrent logically splits files into pieces, typically 250KB each, and computes for each piece the SHA1 hash. These hashes are also stored in .torrent file and provide peers a way to verify the integrity of the data received from other peers. In addition to a .torrent file, a tracker and an initial seeder are necessary. At first, the publisher is the only peer with a copy of the file, and should therefore register itself as an initial seeder.

To download a file, a peer contacts the tracker stored in the .torrent file. The tracker will then return a set of other peers that are also downloading the same file. The peer then contacts the other peers to start exchanging pieces. Peers can verify the integrity of the pieces they receive with the SHA1 hashes stored in the .torrent file. Figure 1.2.1 shows a small and simplified BitTorrent network.

Peers exchange data in a tit-for-tat manner. If a peer wants to increase its download rate, it will have to increase its upload rate and check whether the other peer is also increasing its upload rate. If not, the upload rate is set at the original level, and if so the peers maintain current upload rates and both will have a higher download rate. The peer's objective is to download the file as fast as possible, and will continue searching for other peers to increase its download rate during the entire download. A peer that does not upload at all will not receive any data from other peers; thus the tit-for-tat policy prevents free riding which is a common problem for many P2P systems.

BitTorrent has been extended to be able to use a *Distributed Hash Table* (DHT) as an alternative for the centralized tracker. The DHT spreads the task of storing the peers for each torrent across all peers, and each peer becomes a tracker. All torrent swarms are stored by the same DHT. For each swarm, the DHT stores the peers. Thus a single DHT can replace all centralized trackers. A DHT removes the need for trackers. Besides as the primary swarm discovery method for trackerless torrents, the DHT is also used as backup for traditional tracker.



©2005 HowStuffWorks

Figure 1.2: A simplified BitTorrent network (source: <http://www.kevinwolf.com/?m=20060316>)

The BitTorrent system does not address the dissemination of .torrent files. The usual solution to this issue is to deploy a web server that hosts the .torrent files. Examples of such sites are `mininova.org`, `thepiratebay.org`, and `btjunkie.org`. These sites often also support RSS feeds of new torrents in a certain category or that match a certain search query.

1.2.2 Tribler

Tribler is a social-based P2P file sharing system based on BitTorrent. Tribler is a joint research effort by The Delft University of Technology and De Vrije Universiteit Amsterdam and is funded by the I-Share project. The I-Share project conducts research in area of resource sharing in virtual communities. Tribler builds upon the BitTorrent protocol, and has added multiple features mainly focused on social interaction and P2P video streaming. The features added by Tribler include .torrent gossiping, recommendations, friends-aided download, and Video-on-Demand.

Users in Tribler are not just an IP address and port number as is with most BitTorrent clients. Users have nickname and are accompanied by an avatar. Users are connected with each other by an overlay network and are continuously exchanging information such as .torrent files and person information. Gossiping .torrent file takes away the need for the user to search the web for .torrent files because they are provided by other Tribler clients. Furthermore, gossiping is used to exchange data about persons including nickname, avatar, and completed downloads.

A Tribler client tries to find other users with a similar taste, and using this information it provides recommendations for other files. So-called *taste buddies* are found by comparing the set of completed downloads. If the similarity for two users is high enough, then these users are considered taste buddies. Every download that a user has completed is regarded by Tribler as a recommendation to his taste buddies for that file. Thus Tribler gives the user recommendations aimed at his taste. The more downloads a person has completed the better will the recommendations be.

Tribler also allows the user to create friends. A user can request his friends to help him with a download. When helping a friend, file pieces are no longer uploaded in a tit-for-tat manner as usual, but as a friend, uploads are done altruistically at full upload rate. It is required the friendship is mutual, i.e., both users have to add each other as a friend. To ease the process of friendship creation, users can send invites by mail from Tribler.

Users no longer have to wait for their movie download to be completed before watching them. BitTorrent splits files into pieces. The usual policy of BitTorrent clients is to download rarest pieces first as this will lead to a uniform distribution of the availability of the pieces. However, this policy blocks the possibility to playback movies while they are being downloaded, because that requires the file to be downloaded in order. Tribler has adapted BitTorrent to be able to playback movies during the download provided the download rate is sufficient. This feature provides Tribler with scalable Video-on-Demand.

1.3 Outline

The main issue that this thesis addresses is the poor scalability of current Web 2.0 applications. The remainder of this thesis is organized as follows. Chapter 2 discusses the problem of unscalable Web 2.0 applications and the importance of this problem. Subsequently, functionality and features of existing Web 2.0 sites are discussed to clearly define the criteria which a potential solution must satisfy. Finally, a few current approaches are discussed. In Chapter 3, the architecture of the Web 2.0 Browser is an application, which provides an alternative way to navigating through videos, photos, and articles from multiple web sites. The integration with Tribler and the extensions made to Tribler are also described. Chapter 4 presents experiments and evaluation. Performance measurements as well as community measurements of the *Web 2.0 Browser* are presented. Finally, in Chapter 5 we give our conclusions and future work.

Chapter 2

Problem Description

Web 2.0 applications are delivered through an Internet browser by a single server or several computer centers. Consequently, the performance of these applications decreases as the number of users increases because the resources of the servers have to be shared with all users. Peer-to-peer networks, on the contrary, perform better as the user base grows.

In this chapter, we start by examining the problem of unscalable Web 2.0 applications and its magnitude. Our goal is to create a scalable Web 2.0 platform that offers Web 2.0 functionality and is hosted by a scalable peer-to-peer network. Before looking for a solution to this problem, it is necessary to formulate the criteria which a solution must satisfy. To formulate the criteria of a scalable Web 2.0 platform, we first analyze the functionality offered by Web 2.0 applications and peer-to-peer networks, and we define the criteria as best of both worlds. Finally, we describe current solutions to the problem and indicate why these attempts are not satisfying. Thereby justifying the need for a new scalable Web 2.0 platform.

2.1 Scalability and Robustness

The content of most Web 2.0 applications is delivered through the web browser and is served by a collection of powerful web servers. This traditional approach has two major drawbacks, poor scalability and poor robustness.

The client/server architecture scales poorly because the web site is driven by a collection of dedicated servers. The poor scalability is caused by the limited availability of bandwidth, storage, and computational power used to deliver the service to clients. The amount of resources needed to handle client requests depends mainly on the number of client requests. As the popularity of a web site increases, more requests will be issued and have to be handled. Handling the increased number of requests requires additional resources. These additional resources are usually provided by expanding the collection of servers with more servers.

The client/server model provides no robustness, because the servers are a single point of failure. Once the web servers fail, content hosted by these servers is no

longer available to anyone. To ameliorate robustness, large web sites often are equipped with backup servers which are activated when the main servers fail. However, these measures do not tackle the issue at the root of the problem, which is the centralized architecture, and therefore, site outages remain inevitable. Moreover, server failures may even lead to loss of vast amounts of data.

So to partially improve robustness and deal with poor scalability, web sites have to expand their server farm so that enough resources are available to handle client requests. However, expanding server farms comes with costs, and so this raises the question to what extent will organizations be able to continue to expand server farms before the operating costs of the service become too high for it to be profitable.

WikiMedia

A good example is the WikiMedia Foundation. WikiMedia's mission is to empower and engage people around the world to collect and develop educational content under a free license or in the public domain, and to disseminate it effectively and globally[7]. One of their most well-known projects is Wikipedia (Wikipedia.org), which is a free online encyclopedia that is editable by the public. Wikipedia is in the top 10 of the globally most visited sites (see Table 1.1). The total expenses for WikiMedia have increased tremendously in the last few years. For the years 2004 to 2006, the total expenses were respectively \$ 23,463; \$ 177,670; and \$ 791,907[8]. WikiMedia is a foundation, and therefore, these expenses have to be covered by donations and gifts. Considering the growing popularity of Wikipedia, these costs will continue to grow. Whether WikiMedia will in the future be able to cover the costs with just donations and gifts is questionable.

YouTube

YouTube is another good example. YouTube is a video sharing web site that allows users to share their videos with the rest of the world. Because video is one of the media types that requires the most storage space, YouTube has very high operational costs.

In April 2006, the total amount of bandwidth usage by YouTube was estimated at 200 terabytes per day[2][3]. And each day, over 100 million videos are served and 65 000 new videos are uploaded. Their monthly Internet bill was estimated at nearly a million dollars. Besides bandwidth, YouTube also needs a immense amount of storage space, for their video collection has been estimated at 45 terabytes.

The operating costs for YouTube are immense, and according to that there is much speculation on YouTube collapsing under its own weight.

2.2 Our Aim

Our aim is to provide the functionality offered by Web 2.0 sites with the scalability and robustness of peer-to-peer networks. Scalability is especially an issue for sharing sites that allow users to upload and share their content with the rest of the world. The total amount of data and traffic may become very high as pointed out in the previous section. Therefore, we restrict ourselves to Web 2.0 sharing sites.

Most Web 2.0 sharing applications focus on a single type of media, e.g., YouTube focuses on video, Flickr on photos, and Wikipedia on text. However, our intent is to be able to support all types of media including video, pictures, and text. Such that any content, regardless of its type, can be offered with the scalability and robustness of peer-to-peer networks.

As model Web 2.0 site, we use the popular video site YouTube. YouTube is by far the most popular Web 2.0 sharing site, and we therefore consider YouTube's approach to Web 2.0 sharing is good. In addition, most Web 2.0 sharing sites do not differ much in the functionality they provide.

In this section we analyze the functionality offered by YouTube and Tribler. Next, we identify the desired features for our Web 2.0 platform in order to formulate the requirements.

2.2.1 YouTube

YouTube is the most popular site for sharing video clips with the rest of the world. It was founded in 2005 and was acquired by Google Inc. in November 2006. In this section, we analyze the functionality of YouTube.

Ease Of Use YouTube is a video site aimed at the mass, and accordingly the site is easy to use. The main page of YouTube presents the user immediately with videos available on YouTube including videos being watched at the moment by other users, most viewed videos, most discussed videos, and top favorite videos. Videos are accompanied with rich metadata including a description, tags, category, date of creation, and a thumbnail of the video. Additionally, a user can search for specific videos using the keyword search. With each video, buttons are available to rate the video, to add the video to favorites, and to share the video with others. All this is presented in a "point-and-click" interface.

Publishing a video is also very easy. First, a user only enters some metadata and selects the video file from the local hard drive to upload. Alternatively, a user can also choose to create a video directly via his webcam. The collection of videos uploaded by a single user is considered to be a channel to which fellow users can subscribe.

Video-on-Demand YouTube videos can be watched instantly; it is not required to download the entire video before watching the video. This is an important aspect of the usability, because users do not like to wait.

Peer Review The primary tool provided by YouTube to let users review videos is its rating system. Each user can rate every video on a scale of 1 to 5. YouTube displays the average rating with each video, and the total number of ratings. In addition to these ratings, the number of views and the number of times the video is favourited gives also an indication of the popularity of the video.

Wealth Of Content The number of videos hosted by YouTube is immense. The total amount of video is estimated at 45 terabytes[3]. This immense amount of videos contributes to the popularity and success of YouTube, because a wide collection attracts many users.

2.2.2 Tribler

In this section, we analyze the functionality of Tribler.

Decentralization And Scalability Tribler is a decentralized system, and does not have the scalability issues from which centralized architectures suffer. This is the most important quality of Tribler, because it solves our initial problem, the poor scalability of Web 2.0 applications. A fully decentralized system does not have any maintenance costs. There are no central components that need to be maintained, and each user maintains its own client software like upgrading to the latest version. In addition to scalability, decentralization also removes any central authority which has to ability to delete any content that it finds undesirable. With a centralized architecture, the content can be managed by the supplier of the web service.

High-Definition Videos Tribler supports High-Definition (HD) videos. Because, Tribler is in its basic form a file sharing system, and it does not alter the files in any way. In fact, it is the scalability of Tribler that allows it to handle the large size of HD videos.

Video-on-Demand Tribler has two different download modes: normal mode and *play ASAP* mode. With these modes the user can select between two possible policies which decide in which order the pieces of the file are downloaded. In the normal mode, the usual piece picking policy of BitTorrent is used, i.e., rare pieces are preferred over more common pieces.

The *play ASAP* mode is intended for video and audio downloads. This mode will download the video and audio files in order, which allows to stream them. For multi-file torrents, the *play ASAP* mode lets the user pick a file from the torrent which will be downloaded using this policy.

Community The peer review functionality of Tribler constitutes of popularity and recommendations. Tribler determines the popularity of a download by looking at the number of peers in the swarm of the download. The more peers in a swarm

	Web 2.0	Tribler	Our Vision
Ease Of Use	×		×
HD Videos		×	×
Community	×	×	×
Scalability		×	×
Video-on-Demand	×	×	×
Wealth Of Content	×		×

Table 2.1: A summary of the functionality of Web 2.0 applications, Tribler, and our vision of a scalable Web 2.0 platform.

of an item, the more popular the item is. The total number of peers in a swarm can be retrieved from a tracker with the so-called *scrape extension*. Items can be sorted based on the popularity to easily find the most popular items.

The recommendations of Tribler provide users with peer reviews from other like-minded users, so-called *taste buddies*. Due to recommendations, users do not have to search for content they like, for it is pushed to them.

2.2.3 Our Vision

In this section we look at the functionality offered by Web 2.0 applications and Tribler to formulate the criteria of the scalable Web 2.0 platform. Table 2.1 summarizes the functionality of Web 2.0 application and Tribler, and it defines our vision of a scalable Web 2.0 platform the as the best of both worlds.

Tribler has an easy to use graphical user interface. Items are accompanied with rich metadata, and searching and downloading an item is straightforward. However, publishing with Tribler is not easy. To publish an item, the user needs to take the same steps as publishing with BitTorrent. A publisher has to set up a tracker and create a .torrent file which gets disseminated with Tribler to other users. Ideally, a user does not have to know anything about trackers and .torrent files, and it only has to select which files to publish with the graphical user interface.

It is technically not impossible for a centralized architecture to provide HD quality videos, yet in practice, this is much harder to achieve because of the higher costs that comes with a higher quality. The videos of YouTube have a resolution of 320×240 pixels while HD videos of full quality of a resolution of 1920×1080 . Thus, a full HD quality videos contains 27 times more information compared to YouTube videos. Offering HD videos would increase the required resources approximately with a factor 27. Considering that the costs of YouTube are already immense, HD would make it infeasible.

To leverage existing content, our platform must integrate with popular Web 2.0 applications. Once an item is retrieved from an external Web 2.0 site, it should be injected into our own scalable network. Further distribution of that item benefits of the scalability of our platform. To increase usability, our platform integrates with these sites seamlessly, and users should not be able to tell whether an item comes

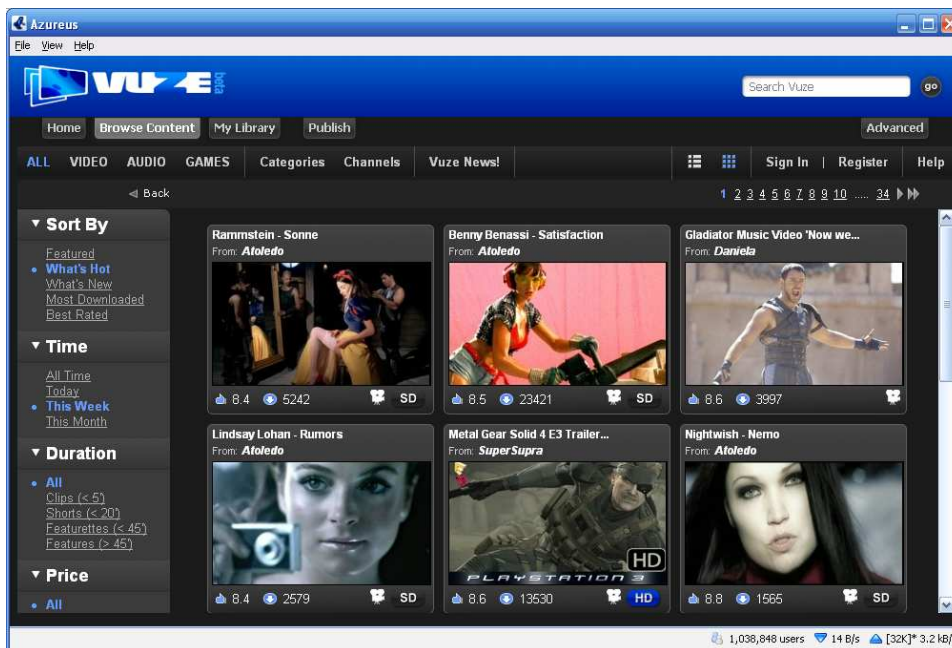


Figure 2.1: Vuze showing an overview of the hottest videos.

from an external site or from our own network. Although our main focus lies been on video, we do not restrict to this media type. Photos from Flickr and texts from Wikipedia are must also be supported.

2.3 Related Work

There is already a number of Web 2.0 applications that leverages the scalability of P2P systems. In this section we describe three of these applications, Vuze, Joost and Babelgum. Furthermore, we discuss why these applications do not meet our criteria.

2.3.1 Vuze

Azureus is one of the most popular BitTorrent clients. Version 3 of the client is named Vuze, and it is a platform for publishing video, audio, and games. Every user can publish and distribute their videos, music, and games. Users browse the content through a stunning graphical user interface (see Figure 2.1). Content is divided into several categories. In addition to the categories, Vuze offers a few fixed channels, such as *HD Trailers*, *BBC*, and *Anime!*. Users can search for content using the keyword search and by making a selection using criteria such as popularity, duration, price, and date of publication of the content.

Downloading of items occurs through the BitTorrent protocol, and every item in

Vuze is a torrent download. The downloaded copy of the file is identical to the original version that was uploaded. Consequently, Vuze also supports High-Definition content unlike YouTube which rescales every video to a resolution of 320×240 pixels.

Vuze supports a few business models. Not all Content on Vuze is free as publishers can decide to make their content available for purchase, rental, or make it ad-supported. These restrictions are enforced using DRM.

Vuze fails on two of our criteria: Scalability and Video-on-Demand. The scalability of Vuze is already much better than YouTube, because data is distributed through BitTorrent. But Vuze still relies on two centralized components in the system: the trackers and the indexing servers. Although Vuze supports distributed tracker by means of a DHT, the primary method for swarm discovery is by trackers. Vuze has a few centralized trackers maintained which have to be expanded as the user base grows.

The second central component are the indexing servers. As discussed in Section 1.2.1, BitTorrent does not provide a solution for the dissemination of the .torrent-files. Vuze simply uses central indexing servers to retrieve items with or without specific properties. Like trackers, the indexing server capacity has to be expanded as the user base grows.

Watching videos downloaded from Vuze require that they are completely downloaded before they can be watched. The size of a High-Definition movie varies from 10 to 30 gigabytes which must all be downloaded before the movie can be watched.

2.3.2 Joost

Joost is a P2P TV application created by the founders of Skype. A user can view the channels in its own channel list, which is composed by making a selection of all the nearly 200 channels offered by Joost. Among these channels are MTV, Reuters, and Comedy Central. For easy navigation, channels are divided into categories and channels can be searched by keywords. Each channel offers a number of programs which user can watch whenever they desire. Programs are delivered on demand, and when tuning into a program, the program starts in a few seconds.

The users can choose to have one or more widgets on the foreground while watching TV (see Figure 2.2). Functionality provided by current available widgets include instant messaging, ratings, and RSS feeds. Currently, Joost provides two instant messaging widgets. First, the channel chat is a chat room shared by the viewers of the same channel. However, because programs are played back on demand, viewers on the same channel do not need to be at the same position of a program or even be watching the same program. This limits a potential discussion on the actual content being played back but lets users with similar taste (they tuned in to the same channel) interact. Second, Joost provides integration with GMail chat and Jabber. Users can use these widgets to send instant messages while watching Joost TV.

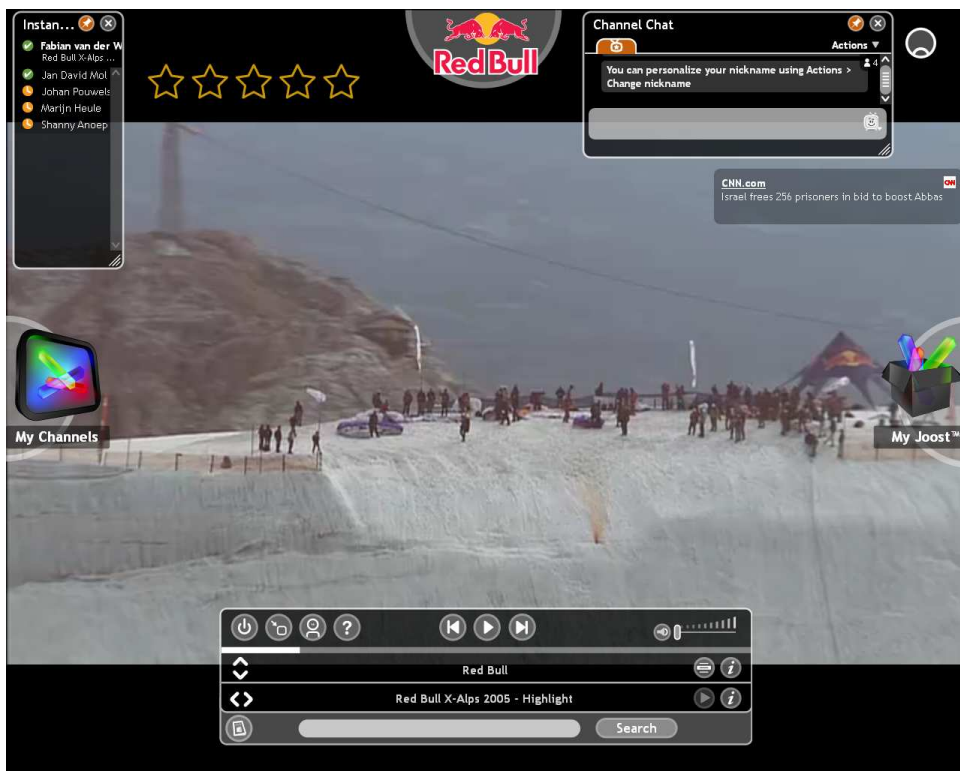


Figure 2.2: A screenshot of Joost displaying a program from the Red Bull channel on Joost with widgets for instant messaging, ratings, and RSS feeds.

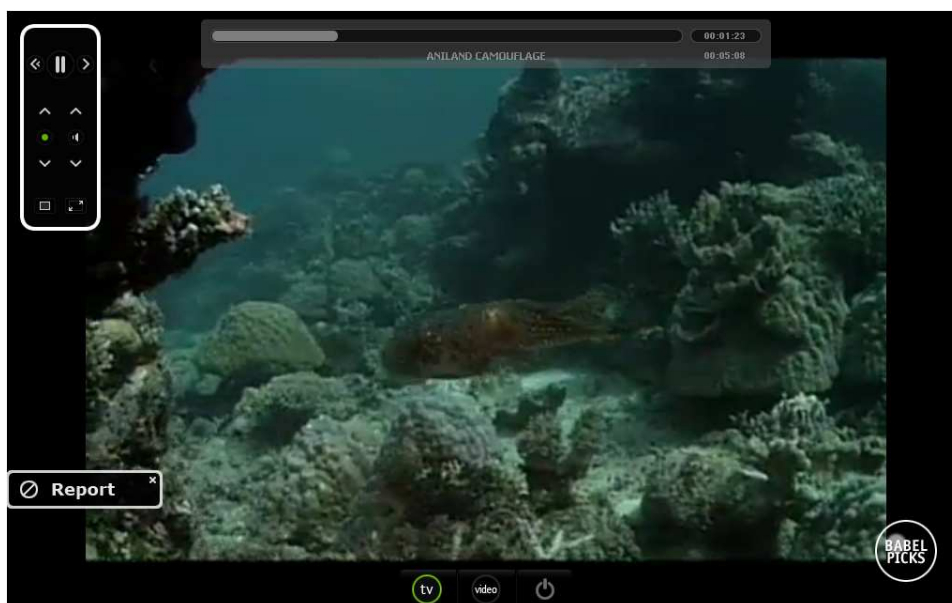


Figure 2.3: Babelgum playing a TV program.

Using the rate widget, users can view the average ratings for the current program and give a rating for a program. Joost also offers a “What’s popular” channel featuring the most viewed programs.

Joost is planning to expose and document the API for creating new widgets to allow external developers to write plug-ins. The widget system becomes then a flexible plug-in system which allows add extra functionality to Joost.

Joost is not a free publishing platform like Vuze is. Publishing is only limited to parties that have a business deal with Joost.

2.3.3 Babelgum

Compared to Vuze and Joost, the functionality offered by Babelgum (see Figure 2.3) is somewhat simple. Babelgum features a number of channels, currently nine, and each channel serves multiple programs. Users can view programs on demand. A unique feature that Babelgum distinguishes from Joost and Vuze is the ability for users to create so-called *smart channels*. A smart channel is created by entering a few tags, and the smart channel will be composed of programs that are relevant to the entered tags. Like Vuze and Joost, Babelgum offers users to rate programs. Furthermore, Babelgum allows users to report inappropriate content.

Babelgum does not satisfy our criteria, because it does not provide High-Definition video quality and publishing content is not straightforward. The video quality provided by Babelgum is mediocre and far from High-Definition quality.

Babelgum is not a free publishing application. Publishing is reserved for professional and semi-professional content owners.

Chapter 3

Design And Implementation

This chapter presents the design of our Web 2.0 platform system which is not hampered by scalability issues. Our Web 2.0 platform is built upon Tribler and extends it with Web 2.0 functionality. To satisfy our criteria as defined in the previous chapter, Tribler needs to be extended in two ways (see Table 2.1).

First, the ease of use of Tribler must be improved, and in particular, the ease of publishing. Publishing with BitTorrent requires the publisher to set up a tracker, create a .torrent file, and distribute the .torrent file to users that would like to download the published file(s). Tribler already takes care of the .torrent distribution with its .torrent file gossiping functionality. Ideally, the user does not have to know anything about trackers and .torrent files, and it only needs to select the file that it wants to publish in the graphical user interface.

Second, to take advantage of the content already available, Tribler must integrate seamlessly with existing Web 2.0 sites. Tribler must be able extract videos, photos, and articles from multiple sites with the corresponding metadata such as title, description, and tags. Items from different sites must be presented to the user uniformly as to the user cannot tell whether a video comes from YouTube or LiveLeak.

We have developed a stand-alone application, the *Web 2.0 Browser*. This application interfaces multiple Web 2.0 sites, and presents the items from various sites to the user in a uniform fashion. Then, the functionality of the Web 2.0 Browser has been partially integrated with Tribler, and Tribler has been extended with easy publishing capabilities.

In this chapter, we first discuss the functionality of the Web 2.0 Browser and Tribler integration. Next, we describe the architecture of the Web 2.0 Browser, and discuss the various parts of the architecture in more detail. Subsequently, integration of the Web 2.0 Browser with Tribler is described. Finally, we discuss how the publishing process is made more easy for end users.

3.1 Software Development Process and Functionality

The primary objective of the Web 2.0 Browser is to allow users to find interesting content that is available from the many Web 2.0 sharing sites without having to navigate to and through all these sites with an ordinary web browser. The content collections of all sites is presented to the user as a single large collection. The user can navigate through this collection and retrieve items by means of keyword searches.

Next is the integration of the Web 2.0 Browser with Tribler. To achieve scalable hosting, once an item is viewed the user becomes a seed for that item in the Bit-Torrent distribution system.

Due to the experimental nature of development of the proof-of-concept, we took an exploratory approach to the implementation. The implementation is evaluated at regular intervals to identify strong and weak points.

The development and functionality of the prototype can be divided into three stages:

1. Single-threaded prototype
2. Multithreaded, multi-site prototype – Web 2.0 Browser
3. Integration of Video Browsing with P2P

3.1.1 Single-threaded prototype

Figure 3.1 shows a screenshot of the prototype in the first stage. By means of a keyword search, users can search the supported web sites for content. When a search is executed, the prototype contacts the web site from which it will retrieve search results. For each item in the search results, metadata is extracted and is presented to the user. The user can download and view the items in the search results. To demonstrate the generality with respect to the type of media, the prototype supports videos, photos, and text.

The search operations are executed by means of a single thread, and therefore the pace at which search results become available is not high. A new item of a search result was available roughly each 1.5 second. See Section 4.1 for more response time measurements of search operations.

3.1.2 Multithreaded, multi-site prototype – Web 2.0 Browser

The prototype in this stage was named the Web 2.0 Browser and was released to the public on April 10th, 2007. Figures 3.3 and 3.4 are screenshots of the Web 2.0 Browser.

The changes and improvements over the first prototype are the following:

- improved search performance,
- combining multiple sites,

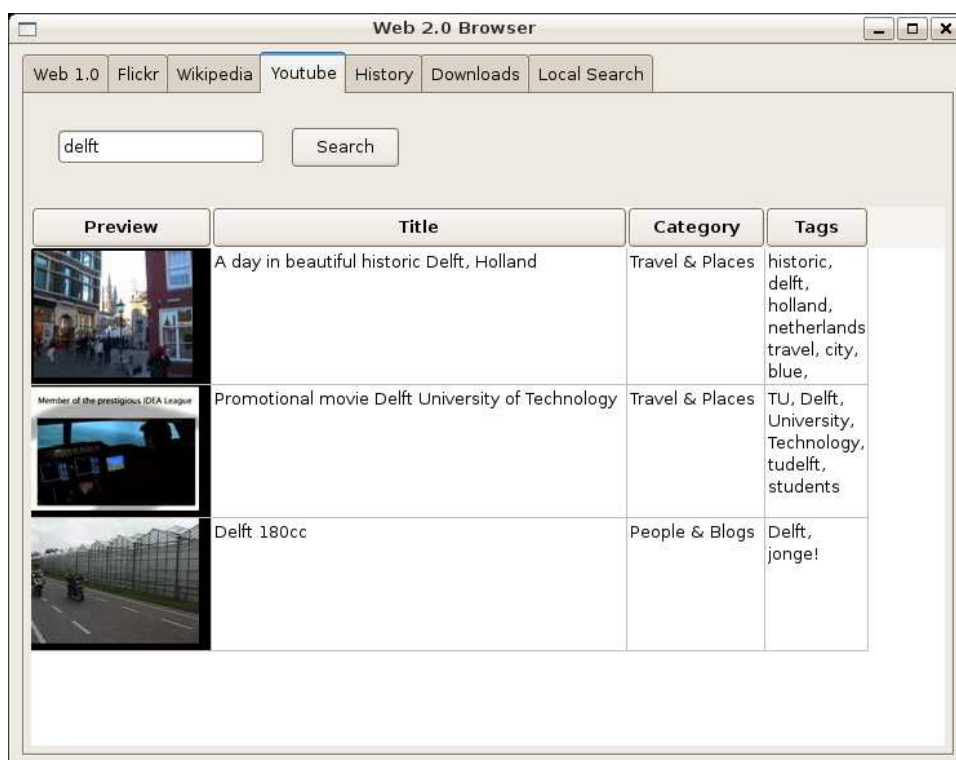


Figure 3.1: First Prototype

- improved GUI with integrated viewer,
- cross Web 2.0 sites rating, and
- iPod video encoding for videos.

In the first prototype, the user explicitly selected on which site the search operation had to be performed. In the Web 2.0 Browser the user only selects the media type it wants to search for, video, photo, or text. The Web 2.0 Browser contacts the appropriate web sites according to the selected media type. For a single media type, the Web 2.0 Browser can use multiple sites. For example, if the user searches for videos then YouTube, LiveLeak, Revver are used to find videos matching the search query. The items collected from the various sites are transformed into a uniform representation. For the user items from different sites look the same and it is not clear from which site an item was retrieved. The union of the video collection of YouTube, LiveLeak, and Revver appears as a single large collection to the user. Search operations are multithreaded with up to four threads per web site. The response times of search operation have improved vastly, see Section 4.1 for more details. The user experience of the search operations has additionally been improved through *precaching*. While the user watches a page of the search results, the next page of the search results is being loaded.

The improved GUI provides easy navigation through the search result set. The furthermore, the GUI is equipped with an integrated viewer, which is capable of handling all downloaded items.

As mentioned in Chapter 2, Web 2.0 sites usually provide a rating system to let users review items. Instead of using the rating mechanisms of the multiple sites, the Web 2.0 Browser is equipped with its own rating system. The Web 2.0 Browser provides cross Web 2.0 site ratings; a single rating system is used for all sites.

There is a growing demand for watching video clips on portable devices such as mobile phones and the iPod Video. To meet this demand, the Web 2.0 Browser is capable of transcoding any downloaded video into a format that the iPod Video is able to play.

3.1.3 Integration of Video Browsing with P2P

The functionality of the Web 2.0 Browser to search and view videos is integrated with Tribler, see Figure 3.6 for a screenshot. The search for torrents and Web 2.0 sites are combined in a single search operation. Searching for torrents does not require any network communication and therefore are available immediately. The search results of Web 2.0 sites are appended to the torrent search results.

There is no cache, thus search operations are not speed up by a cache. Downloaded items, however, are stored on-disk. When a user has downloaded an item, it automatically becomes a seeder for that item in the Tribler network using peer-to-peer technology. Furthermore, publishing items with Tribler only requires the user to select the file it wants to publish.

3.2 Web 2.0 Browser

In this section, we describe the architecture and design of the Web 2.0 Browser. First, the architecture is given, and in the following sections, the different parts of the architecture are discussed.

3.2.1 Architecture

In this section, we explain the architectural decisions that have been made, which are the choice between a stand-alone application or a Firefox extension and between multithreaded communication or asynchronous communication. This section also gives an overview of the components of the Web 2.0 Browser.

Stand-alone vs. Firefox extension

The Web 2.0 Browser is a stand-alone application, however, we have also considered to develop an extension for the Firefox Internet browser. Creating a Firefox extensions improves the browsing experience, because users can still use their browser. However, Usage is limited to Firefox users, users with different browser, like Internet Explorer, will not be able to use the application. In addition, Firefox extensions are written in JavaScript, while the Web 2.0 Browser has to be written in Python for easy integration with Tribler, which is already written in Python. Therefore, the Web 2.0 Browser is a stand-alone application.

Multithreaded I/O vs. Asynchronous I/O

When used actively, the Web 2.0 Browser has simultaneously multiple open network connections. For example, while there may be multiple active downloads, a user may also execute a search operations which requires one or more network connections. There are basically two ways to handle multiple network connections, multiple threads and asynchronous communication.

We chose a multithreaded I/O approach to reduce development time. The Python standard library provides a module to retrieve web pages. This module uses blocking I/O operations. Using asynchronous I/O requires to use a third-party library, like Twisted, or to have to write our own asynchronous HTTP module.

Web 2.0 Browser Overview

Figure 3.2 shows the architecture of the Web 2.0 Browser. For each media type there is a separate database. The Web 2.0 Browser currently supports video, photo, and text, and accordingly there are currently three databases. A database retrieves its items from one or more Internet sites. For example, the current Video Database retrieves its items from YouTube.com, LiveLeak.com, and Revver.com. All videos from these sites are considered to be items contained by the Video Database. The

main functionality a database provides is a keyword search on the items in the database.

Databases retrieve items and their metadata from multiple sites. To do so, a database has for each site that it uses a so-called Web 2.0 interface at its disposal. A Web 2.0 interface knows about the structure of site, how to execute search queries, how to parse the search results, and hides these site-specific details for the database by providing a generic interface. The strict separation of site-specific details and the database provide a flexible and easily extendable system. E.g., adding support for a new video sharing site requires only to specify the site-specific details in a Web 2.0 interface.

When the database is requested to perform a keyword search operation, the database requests each of its Web 2.0 interfaces to perform the same search operation. The database aggregates the results of each interface into a single result set and returns this set as the result of the keyword search operation. The result set only contains metadata of items and not the item itself. Retrieving each item in the result set brings about a lot of extra communication, because the size of the metadata is several magnitudes smaller than the size of the item. Furthermore, the user may be only interested in a few items of the entire result set. In such cases, the extra communication is mostly wastage and the response times of the search operations would be needlessly much higher. From the result set the user selects the items that it wants to download in order to view them.

To further improve the response times of search operations, each database is equipped with a cache. The cache is used to store the metadata of items that is retrieved as a result of a search operation. Instead of having to retrieve metadata of an item and parse it, the cache can be looked up to check whether it stores the metadata. If so, the metadata is fetched from the cache, and any communication to retrieve metadata is saved. Besides metadata, the cache also stores any downloaded items. The size of the cache is unlimited, and metadata is never evicted from the cache. Downloaded items may be removed by the user.

The rating server stores all ratings made by each user for each item, and provides average ratings of each item. Whenever a user rates an item, the Web 2.0 Browser send a message to the rating server containing the rating and IDs uniquely identifying the item, the Web 2.0 Browser installation. A new rating for a specific combination of item and installation IDs replaces any old rating for that combination. Using these records, the rating server keeps track of the average rating of each item. Average ratings are considered to be part of the metadata of an item. However, due to the temporal validity of average ratings, they are not stored in the cache and have to be requested from the rating server every time they are needed. The GUI displays the search results and the downloaded items in a user-friendly interface to the user. If the user decides to view an item, a new download for that item is started and is registered with the Download Manager. The Download Manager notifies the GUI updated of any started, finished, cancelled, and failed downloads.

To playback videos, VLC media player is used. Using the VLC library, video

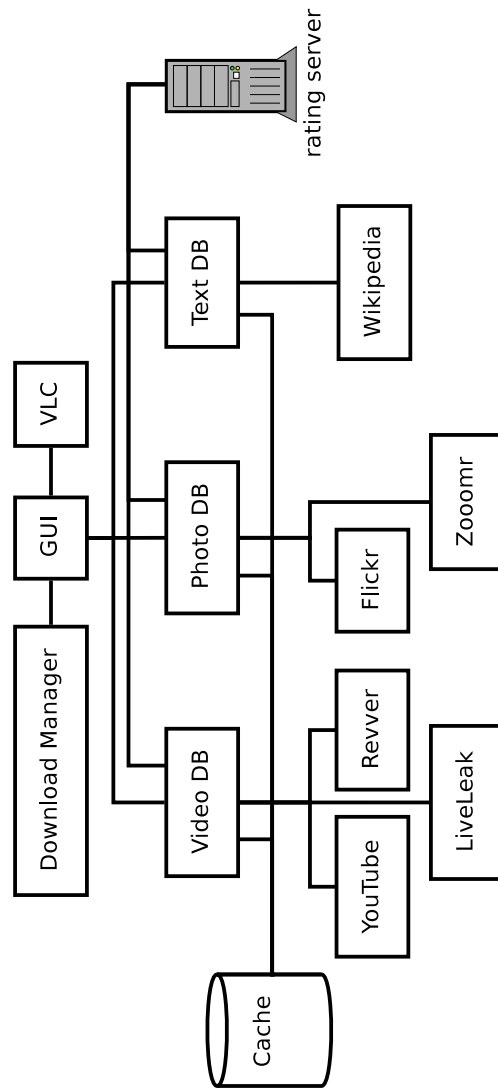


Figure 3.2: The architecture of the Web 2.0 Browser.

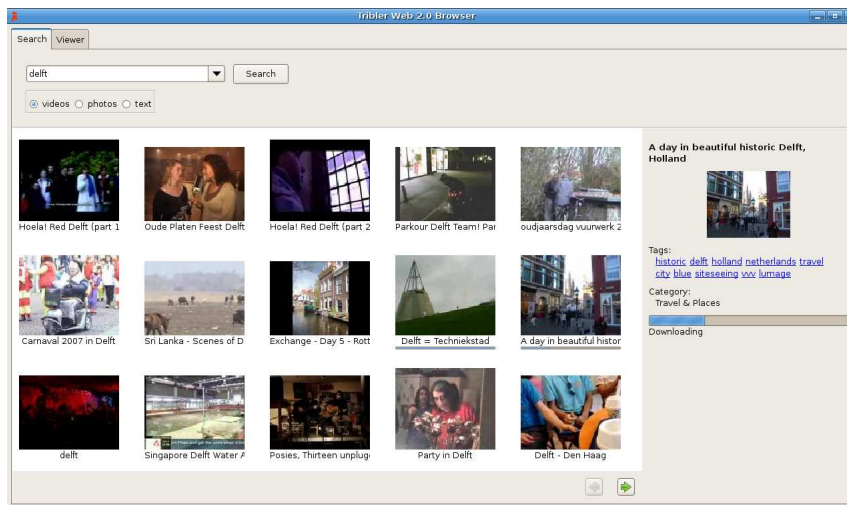


Figure 3.3: Screenshot of Web 2.0 Browser after a search for videos on *delft* has been fired.

playback can be integrated into the GUI. The VLC library is cross-platform and is available for Windows and Unix/Linux. VLC supports a wide variety of multimedia formats including Flash Video, which is used by many video sharing sites. The GUI toolkit is able to display other types of media.

3.2.2 GUI design

The graphical user interface of the Web 2.0 Browser consists of two tabs, see Figure 3.3 and Figure 3.3. The first tab, the *search tab*, allows users to enter search queries and shows the search results. The user enters a search query, selects the media type, and clicks the search button to start a new search.

The second tab is the *viewer tab* (see Figure 3.4). This tab shows the user a list of all files that have been downloaded or are being downloaded, and it allows the user the view these items.

Precaching

The search results are divided over multiple pages like most web sites also do. Switching to a new page of the search results requires retrieve search results over the Internet, and may take therefore some time. To improve responsiveness, the Web 2.0 Browser does *precaching*. The Web 2.0 Browser retrieves not only search results for the current page of search results, but it also retrieves search results for a few pages ahead. While the user is viewing a page of search results, the next two pages are being loaded. The time that a user waits for the next page to be loaded is reduced or even eliminated.

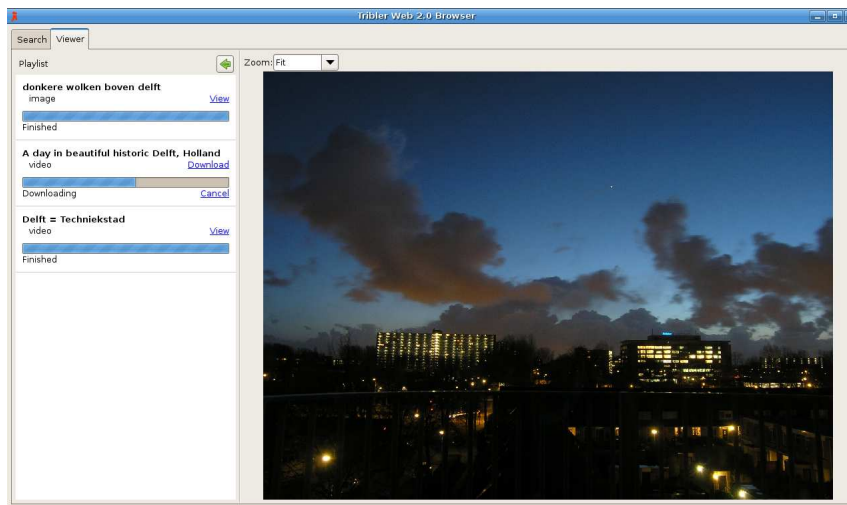


Figure 3.4: Viewing a photo of *Delft* that has been downloaded.

GUI updates

To keep the information presented in the GUI up-to-date we use a push model. As push model we used the Observer-Subject design pattern wherein the GUI observes the different components of the Web 2.0 Browser including search operations and download progress. This push model ensures that all information at all the locations in the GUI is always up-to-date and consistent.

For example, the progress of a download is shown in three different places in the GUI, in the search result grid, in the details panel, and on the viewer tab. The update model allows the various places to be always up-to-date consistently by letting the three places observe the download progress. As soon as a new progress percentage is pushed to the GUI, all three places are updated immediately and simultaneously.

Extensibility

The GUI has been designed to be easily extendable for new media types. Adding a new media type to the GUI involves extending three parts of the GUI. First, the subpanel for the search results grid has to be defined. Each item in the search result is a separate subpanel in the search results grid. What this subpanel looks like depends entirely on the media type. The search results grid positions the subpanels correctly and determine the number of subpanels per a page depending on the subpanel size and grid size.

Second, the details panel has to be defined. Similar to the search grid subpanel, the content of this panel is totally determined by the media type except for the rating control, which is added automatically to the panel. When an item is selected in the search results grid, the item is passed on to the details panel defined for the corresponding media type.

Third, the integrated viewer has to be extended to be able to handle the new media type, so that users can view the items.

3.2.3 Web 2.0 Interfaces

One of the essential components of the Web 2.0 Browser are the Web 2.0 interfaces, because these components are the bridge between the Web 2.0 Browser and items published on sharing sites for videos, photos, etc. This section describes the design of the Web 2.0 interfaces.

There are a number of bases class that a Web 2.0 interface can built upon. The base classes handle requests for items and delivery of items to other components of the application. A Web 2.0 interface only needs to implement the communication with the Web 2.0 site. The available base classes are DBSearch, ThreadedDBSearch, and CompoundDBSearch. These base classes form a flexible and extensible framework for Web 2.0 Browser. We estimate that support for a new Web 2.0 for videos, photos, and text can be added within a few hours.

In this section, we describe the interface which provides the abstraction of Web 2.0 sites. Then the three base classes and their use are explained. And finally, we explain how communication with Web 2.0 interfaces is implemented.

Web 2.0 Search Abstraction

A Web 2.0 interface must expose a few operations in order to let the database use the interface and retrieve items from the site that the interface connects to. For each keyword search a new search object is instantiated which expose a few operations to retrieve the search results. These are *start*, *getMore*, *enough*, and *quit*.

start The *start* operations initializes any necessary resources to start web scraping. However, there is no communication with the web site yet, and no web scraping is performed yet.

getMore The *getMore* operation is the most important operation. With this operations, this operations submit requests for search results. To fulfill the requests, the Web 2.0 interface uses the keyword search of the site and parses the search results. Retrieving and parsing search results requires network communication over the Internet, so to maintain responsiveness of the application, results are returned asynchronously.

enough The *enough* operation is used to order to discard any outstanding request. After *enough*, new request can be issued with *getMore*.

quit Finally, when the search object is no longer needed, the *quit* operation must be used to release resources. It is the counterpart of the *start* operations.

Simple web scraping

DBSearch is the simplest base class to implement a Web 2.0 interface. The Web 2.0 interface is required to only provide a single method, a *getItem* method. Each call to *getItem* must return the next item of the search results. The DBSearch is simple and uses only a single thread, and therefore its performance is not optimal. A performance gain can be achieved if multiple threads are used, which is exactly what ThreadedDBSearch does.

Threaded web scraping

DBSearch does not assume any structure of the web site, however, nearly all web sites share a similar structure for searching the site. This structure can be exploited for parallelism. The structure is as follows. The site provides the user with a text box to enter a search query, and the search button takes the user to an overview of the search results. The search results page shows the results with limited meta information, such as the title, the preview, and the beginning of the description. A results page shows only a limited number of results, typically between 10 and 20. More results, are on different pages which are accessible via *Next* and *Previous* links. For our purposes, the metadata on the results search page is not sufficient, and it is necessary to retrieve the web page of each item, which contains extended metadata.

ThreadedDBSearch exploits this structure to perform web scraping in parallel to increase the performance. A search results page typically shows 10 to 20 items. For each item on the search results page, the Search has to retrieve an additional web page and extract metadata from it. This step can be done in parallel for all items on the results page, and this is exactly what ThreadedDBSearch does by means of multithreading.

A Web 2.0 interface using the ThreadedDBSearch as base must implement two operations, *parseItempage* and *parseItem*. The *parseItempage* operation is expected to fetch, parse the next search results page, and return a list. Each item in the list presents an item of the search results, and must provide enough information to allow the *parseItem* operation extract the necessary metadata. ThreadedDBSearch regards the items in the list as opaque values, and implementations may put any type of data in this list. The *parseItem* method receives a single item from the list returned by *parseItempage*, and it must download the web page of the item and extract metadata. Before actually downloading the web page, *parseItem* should check the cache to reduce latency.

ThreadedDBSearch achieves parallelism through multiple worker threads. The list returned by *parseItempage* is regarded by ThreadedDBSearch as a work queue from which the worker threads pop an item and use *parseItem* to retrieve the full item. When the work queue gets depleted, the ThreadedDBSearch lets one worker thread call *parseItempage* to refill the work queue. This continues until no more items are requested or the search is exhausted.

Multiple web sites

The third base for a Web 2.0 Interface is CompoundDBSearch. The purpose of CompoundDBSearch is to aggregate multiple Web 2.0 Interface objects and make them appear as a single interface. CompoundDBSearch merges the result of the Search objects it is controlling on a first come first serve basis. When CompoundDBSearch receives a request for x items, it requests x from each of the interfaces it controls. With multiple interfaces, this will in fact request too many items, so as soon as the CompoundDBSearch has received enough items from its interfaces it calls the *enough* operation on all of them.

CompoundDBSearch improves the extensibility vastly. Other components in Web 2.0 Browser do not need any knowledge whether they are using just a single web site or if its using many sites simultaneously. CompoundDBSearch makes combining a new Web 2.0 interface with the existing interfaces very simple.

Regular expressions

There are basically two ways for a Web 2.0 Interface to retrieve data from a web site. First, a site may provide an API for external applications. Such an API provides methods methods for navigating through the content of the site. Second, the data is also available via the web pages that are viewed by the Internet surfer. This approach requires that web pages are parsed to extract the actual data and to discard all formatting and unnecessary data.

Parsing web pages can also be done in two ways. First, the HTML structure can interpreted, and data can be identified within this structure. For example, the title of an item can be defined as the text enclosed by the `<h1>` element in the first paragraph (i.e., `<p>`) of the web page. Second, the web page can be treated as plain text from which data can be extracted using regular expression. Regular expression specify which web pages to fetch to perform a search, and how data can be extracted.

Each Web 2.0 Interface can use a different method for retrieving data. However, all the current interfaces fetch web pages and extract metadata using regular expression. The drawback of APIs is that each site has a different API and not all sites provide such a API. On the other hand, the approach of fetching web pages and parsing the results works universally for all web sites.

The few regular expressions shown in Figure 3.5 provide enough information to perform and parse keyword searches on YouTube. The few number of regular expressions also also reflect the flexibility and extensibility of the Web 2.0 Interfaces. Adding new support for a new Web 2.0 site basically consists of finding the correct regular expressions and applying them. Finding the correct regular expressions is trial-and-error process. They need to be restrictive enough that only the desired data is extracted and they need not to be too strict to correctly for all items. From our experience, this takes only a few hours of programming.



```
youtube.py (~/.web2browser/web_2.0_browser/video) - VIM
File Edit View Terminal Tabs Help
youtube.py
14
15 ENCODING = "utf-8"
16
17 site = "youtube.com"
18
19
20 RE_SEARCHITEM = r"<a href=\"/watch?v=(.*?)\".*?<img src=\"(.*?)\".*?</a>"
21 RE_TAG = r'<meta name="keywords" content="(.*?)>'
22 RE_TAG2 = r'([^\s,]+)'
23 RE_CAT = r'<a href="/browse?s=.*?Video\+Category\+Link.*?(.*?)</a>'
24 RE_NAME = r'<title>YouTube - (.*?)</title>'
25
26 URL_WATCH = "http://www.youtube.com/watch?v=%s"
27 URL_DL_VIDEO = 'http://www.youtube.com/get_video?video_id=%s&t=%s'
28 RE_VIDEOURL = r'player2\.swf?videoid=([^\&]+?)&.*?&t=([^\&]+?)\?(?:&|")'
29
30 URL_SEARCH = "http://www.youtube.com/results?search_type=videos&search_quer
y=%s&search_sort=relevance&search_category=0&page=%d"
31
32 RE_RESULTS_HASNEXT = r'class="pagerNotCurrent">Next</a>'
33
34
```

Figure 3.5: The structure of YouTube.com captured by a few regular expressions.

3.2.4 Ratings

We keep track of ratings by with a central rating server. Ratings can be retrieved and submitted with simple HTTP GET and POST operations. When the user selects an item from its search results, detailed information is showed including the rating that was given by other users. If the user has not submitted its own rating for this specific item, then the average rating is retrieved from the rating server.

If a user selects and deselects an item multiple times in a short time frame, the Web 2.0 Browser will retrieve the rating for that item as many times as the item is selected. It is unlikely that a rating changes in a short period of time. So to reduce communication, any ratings retrieved from the rating server are cached and remain valid for a short period of time. After this period, the rating has to be retrieved from the rating server again.

Storing and retrieving ratings are not vital to the primary functionality of the Web 2.0 Browser. Therefore, retrieving and submitting ratings are done on a best-effort basis. This prevents any failure in the communication with the rating server from blocking or crashing the Web 2.0 Browser.

3.2.5 Download Manager

New downloads are registered with the Download Manager. The Download Manager ensures that there is at most one active download for each item. The Download

Manager also notifies the GUI of any newly started downloads and the progress of downloads.

3.3 Tribler Integration

As mentioned in the introduction of this chapter, Tribler has to be modified in two ways. First, Tribler has to be able to take advantage from the large collection of content offered by various sites. This is exactly what the Web 2.0 Browser does. So the Web 2.0 Browser has to be integrated with Tribler. In particular, the Web 2.0 interfaces and the Download Manager have been integrated. Tribler users can search for videos offered by YouTube and LiveLeak, download them, and view them. Any downloaded videos are automatically published using the scalable distribution system of Tribler. This requires that the process of publishing of an item can be carried out without any user involvement.

This is what the second modification of Tribler is about, which is the ease of publishing of Tribler has been improved. Ideally, the user only selects the file it wants to publish and Tribler takes care of the rest. The main problem in the BitTorrent process is the swarm discovery. The traditional swarm discovery requires the publisher to set up a tracker, which will serve as a venue point for peers. Most normal users do not have access to a server that is running continuously, which can be deployed up as tracker. Ideally, users do not require any knowledge about trackers or .torrent files, and just simply chooses the file to publish and Tribler takes care of the rest. This can also be used for publishing videos from YouTube and LiveLeak because no user intervention will be necessary.

To remove the need for trackers, we use a different swarm discovery method that is called *decentralized tracking*. With decentralized tracking, each peer becomes a potential tracker for the torrent, so no dedicated trackers are necessary. This swarm discovery method requires no action from the user neither does creating a torrent with decentralized tracking.

In this section, we first present the integration of the Web 2.0 interfaces with Tribler, and subsequently, the decentralized tracking is explained.

3.3.1 Integration with Tribler

We have only integrated the YouTube and LiveLeak interfaces with Tribler, however other Web 2.0 interfaces can also be integrated. To browse content the Web 2.0 interfaces provide a keyword search functionality. Tribler also provides keyword search functionality to search for torrents. We have merged the Web 2.0 search functionality with the Tribler search functionality to create a single search functionality to search in all content that is available through torrents and Web 2.0 interfaces.

The Web 2.0 Browser closely resembles the Tribler GUI. The search results are displayed in a grid that contains a thumbnail and the title of the item. On the right

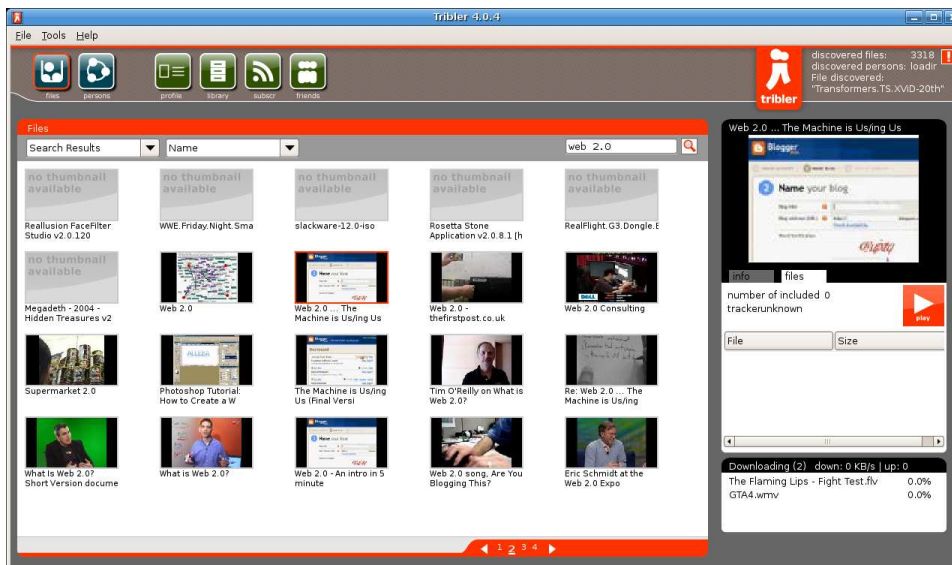


Figure 3.6: Keyword search on “web 2.0” in Tribler.

side of the window there is a panel that shows the details of the currently selected item. This single grid contains the search results of both the Tribler torrent search and the Web 2.0 search.

Figure 3.6 shows Tribler when a keyword search is performed. When a keyword search is performed Tribler searches in its local torrent database, but it also uses the Web 2.0 interfaces to retrieve more search results.

The Tribler torrent search searches only the local torrent database, therefore, the results of this search are “immediately” available because no Internet communication is required. The results of the torrent search are displayed first to the user and search results of the Web 2.0 search are appended to the torrent search results. The Tribler item grid has been modified to request items from the Web 2.0 Browser when it is needed. And just like the Web 2.0 Browser, two pages are loaded in advance to improve responsiveness.

If a user decides to download an item, the item is downloaded by the Download Manager which is the same from the Web 2.0 Browser. The downloads of the Download Manager are displayed together with the downloads of Tribler itself in the Tribler Library. When the item is completely downloaded it is treated the same as if the item was published by the user. A .torrent file is created which will cause the Tribler Library to view it as a torrent instead of a Web 2.0 item. The .torrent file is spread through BuddyCast to other Tribler users, so each item that is downloaded through a Web 2.0 interface is immediately injected in the Tribler network and, thereby, benefit from the scalability of Tribler. Thus, each item from a Web 2.0 interface needs only to be downloaded once through that interface, and then Tribler no longer depends on the Web 2.0 interface for that specific item and handles the distribution of that item.

3.3.2 Decentralized Tracking

Besides the usual trackers, some BitTorrent clients support a distributed alternative for swarm discovery. This method is often referred to as *decentralized tracking*, and is achieved by using a *Distributed Hash Table* (DHT). A DHT is a hash table that spreads the storage of key/value pairs across the peers that are participating in the DHT.

There are two DHTs in use for BitTorrent swarm discovery, Mainline DHT and the Azureus DHT. The Azureus DHT is part of the Azureus BitTorrent client. The Mainline DHT is developed as part of the Mainline BitTorrent client and has been adopted by several other BitTorrent clients including BitComet and utorrent. Unfortunately, both DHTs are not compatible with each other. Thus values stored in one DHT cannot be retrieved by the other DHT, and likewise, peers registered in the Mainline DHT cannot be discovered by peers using the Azureus DHT, and vice versa. Thus for each torrent there are two separate swarms in both DHTs.

We have chosen to integrate the DHT from the Mainline BitTorrent client into Tribler mainly reduce development time. The Mainline DHT is, like Tribler, written in Python and can therefore easily be integrated. The Azureus DHT is, on the other hand, written in Java. And there is no documentation available of the Azureus DHT thus porting it to Python also requires to elicit its exact working from the Azureus source code. For a more in-depth analysis of both DHTs we refer to [1].

We have integrated the Mainline DHT with Tribler. The Mainline DHT is called Khashmir, and is based on Kademlia[5]. This section first explains Kademlia, and then the modifications for BitTorrent.

Khashmir

Kademlia is a DHT that support the usual set and get operations. In Kademlia, keys are 160-bit values and participating nodes have a nodeID in the same 160-bit key space. In Khashmir, infohashes of torrents are used as keys and the values are a list of peers that participate in the swarm of the corresponding torrent.

Key/value pairs are stored at nodes with the nodeIDs closest to the key. The XOR operator is used to calculate the distance between keys and nodeIDs. Thus the distance between node A and B with IDs respectively A_{id} and B_{id} equals to $A_{id} \oplus B_{id}$, which should be interpreted as an unsigned integer. Thus a node stores the swarms for torrents of which the infohashes is close to its nodeID.

Routing Table The routing table is organized in so-called k -buckets. A k -bucket covers a range of the key space, and can contain up to k nodes which are ordered from least-recently seen to most-recently seen. The k -buckets are organized such that the routing table can contain many nodes that are close and few nodes that are far. k -buckets get filled with nodes as a side effect of lookup and store operations and by a regular lookup for the node with distance 1.

When new nodes are added to the routing table there is a preference for old live nodes over new nodes. This preference has two advantages. First, the longer a node is up the more likely it is that it will stay up for another hour. Consequently, the nodes in k -buckets have a higher probability of being live. Second, it provides robustness in a hostile environment. It is not possible to flush the routing table of nodes by flooding the system with new nodes. New nodes will only be added to the routing table if the old nodes have left the system.

Operations Instead of the usual get and set operations, Khashmir provides a *AnnouncePeer* and a *GetPeers* operation. For both operations, the first step is to find the closest nodes to the target (i.e., the infohash). This is done by querying nodes for the closest nodes to a target that they know of. This continues until no nodes can be found that are closer to the target.

Subsequently, *AnnouncePeer* or *GetPeers* requests are sent to the closest nodes to a target. If a node receives an *AnnouncePeer* request, it adds the contact information of the requester to the swarm for torrent for which the request was made. On a *GetPeers* request, the node returns the list of peers in the swarm.

A Tribler peer learns from other Tribler peers whether it is connectable from the Internet. When a peer is not connectable, it will not use *AnnouncePeer*. It will only be learn peers through *GetPeers*.

Chapter 4

Experiments And Evaluation

In this chapter we present the experiments and evaluation that have been performed to identify weaknesses in order to be able improve the Web 2.0 Browser and our modified Tribler. In particular, the performance of the Web 2.0 interfaces and the Khashmir swarm discovery have been evaluated. Our primary interest in the performance lies in response times of the various operations available by the Web 2.0 interfaces and Khashmir. These measurements indicate how long the user has to wait before search results are available or a video can be streamed for playback.

Furthermore, we present the data that has been collected as a result of usage of the Web 2.0 Browser by the community. This data is collected by the rating server and consists of retrieval and storage requests of user ratings. We combine this data with the tags associated with items to identify popular tags and average ratings.

Finally, we present the end-to-end video experiment, which shows that our system actually works. This experiment shows the entire flow of a YouTube video that is downloaded from the web site and which is distributed with Tribler in a scalable manner. The video is first downloaded through the Web 2.0 interface for YouTube. As soon as the download is completed, a torrent is created for the video which is then announced to other Tribler clients with torrent gossiping. Another Tribler client will then download the same video through the scalable BitTorrent and Khashmir.

This chapter first discusses the experiments and evaluations that concern the Web 2.0 Browser. These are the Web 2.0 Interfaces performance measurements and the data collected by the rating server. Subsequently, the experiments and evaluations concerning our extended version of Tribler are presented. These are the Khashmir performance measurements and the end-to-end video experiment.

4.1 Web 2.0 Interfaces

In this section, we present the performance analysis of the Web 2.0 interfaces. The Web 2.0 interfaces are one of the most crucial components of the Web 2.0 Browser since these provide the actual content. Performance is important for usability of

the Web 2.0 Browser because users do not like to have to wait for search results. When the users use the traditional Internet browser and perform search keywords on Web 2.0 sites, search results are available within a few seconds. For the Web 2.0 Browser, and also the Tribler integration, to be a viable alternative for the traditional Internet Browser, the performance of the Web 2.0 interfaces should be in the same order of magnitude as the performance of a usual Internet browser, like Firefox. Ideally, the Web 2.0 interfaces are fast enough to load the next page of search results while the user is viewing the current page. In that case, when the user proceeds to the next page, that page is available immediately, and the user does not have to wait.

Note that measurements are subject to limited availability of bandwidth on both the client side and the server side.

As discussed in Chapter 3 there are three types of Web 2.0 interfaces. The first version is a single-threaded search that uses a single thread to perform synchronous communication. The second version was a multithreaded search that uses multiple threads. Finally, the third version combined multiple searches to create a single compound search that uses multiple sites.

Figure 4.1 shows the response times for a single keyword search for these three versions and different number of threads. The figure shows the amount of time needed to fetch the first 100 items of the search results. Additionally, the figure shows an magnification of the first 10 seconds of the measurements to clearly display the latency to the first returned item for each measurement.

These measurements have been made with an automated version of the Web 2.0 Browser, which at startup immediately executes a search and logs at what time search item results are delivered by the Web 2.0 interfaces to the GUI. Before each measurement the cache of the Web 2.0 Browser is flushed because the cache may greatly reduce the amount of necessary communication and thus also the amount of time. A non-empty cache may therefore give a distorted reflection of the performance of the Web 2.0 interfaces.

The time is measured from the moment that the search is started until the moment at which the 100th item is delivered by the Web 2.0 interface. Included in these measurements is the time needed to display the search results in the GUI, which among others includes rescaling and rendering thumbnails.

These measurements have been run on the video Web 2.0 interfaces. Measurements on a single Web 2.0 interfaces have been made on YouTube, since it is the leading Web 2.0 video site. Measurements of the compound search have been made with all video Web 2.0 interfaces combined, i.e., YouTube, LiveLeak, and Revver.

Measurement Results

From Figure 4.1 it is clear that a single threaded Web 2.0 with just one web site is by far the slowest method for retrieving search results. The first item is available 2.7 seconds after the search has been initiated. The 100 items are delivered in approximately 148 seconds. On average, between each new item delivered by the

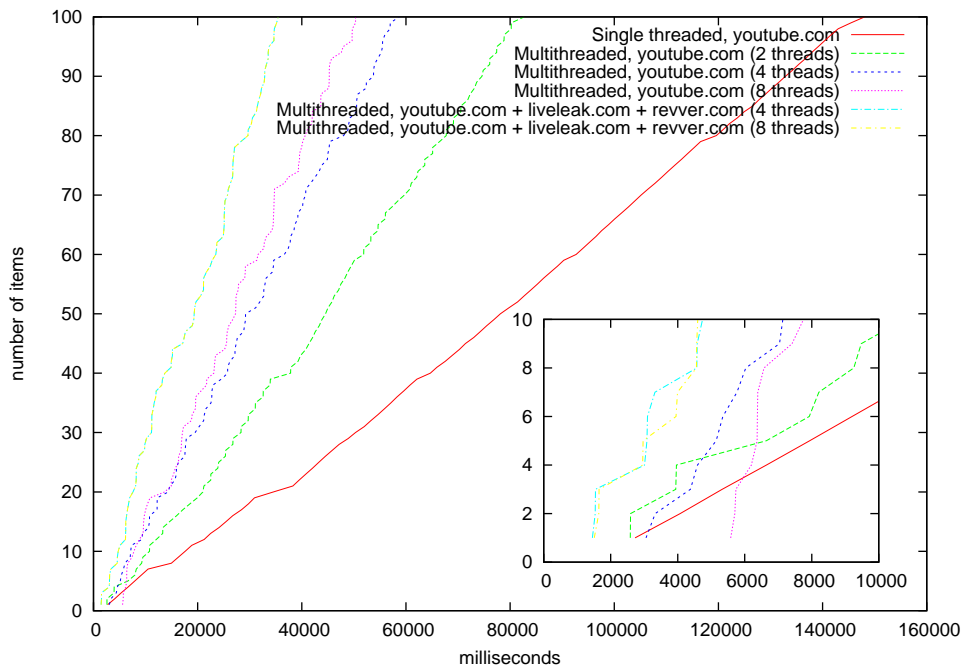


Figure 4.1: Performance of Web 2.0 Interfaces in milliseconds.

Web 2.0 interface the user has to wait 1.48 seconds.

The multithreaded YouTube interface with two threads provides a substantial speedup over the single threaded interface. All the 100 items were available in under 83 seconds. This is a speedup of $\frac{148.064}{82.686} \approx 1.79$ compared to the single threaded interface, and has an efficiency of approximately 0.895. With four threads, the speedup equals to $\frac{148.064}{58.699} \approx 2.94$ and the efficiency to approximately 0.74. Using eight threads compared to four threads yields a speedup of only $\frac{58.669}{50.883} \approx 1.15$. The limited speedup is caused by the limited available bandwidth.

The speedup from using multiple sites is limited compared to using multiple threads. Using multiple sites comes with other advantages: smaller delays and a more diverse collection. The smaller diagram in Figure 4.1 shows an enlargement of the first 10 seconds.

When only YouTube is used as source for videos then the first item is available in approximately 2 to 3 seconds. This is a result of the delay of YouTube. With multiple sites the first item is available in 1.5 second. One or more of the other sites have a lower latency than YouTube and provide the first item quicker. In this case, LiveLeak has a smaller delay and provides virtually always the first item. See Table 4.1 for a comparison on the latency of YouTube, LiveLeak, and Revver.

To provide a smaller delay, it is possible to only use LiveLeak instead of combining YouTube and LiveLeak. In that case, however, users would be deprived of the wealth of content provided by YouTube. Using multiple sites yields a greater variety of content. Especially when the focus of the videos sites is different. For

	YouTube.com (20 items)	LiveLeak.com (12 items)	Revver.com (15 items)
first run	4331	1659	7727
second run	4997	1761	13240
third run	4700	1712	9860
average	4676	1711	10276
Web 2.0 Browser	7575	6158	6200

Table 4.1: Firefox response time vs. Web 2.0 Browser in milliseconds

example, YouTube profiles itself as the family friendly video sharing site. Therefore, any videos that are considered offensive or inappropriate in some other way are removed. On the other hand, LiveLeak and Revver do not strive to be family friendly like YouTube.

Web 2.0 Browser vs. Firefox

In this section we evaluate the response time of our Web 2.0 interfaces in comparison with the response time that is incurred by using an ordinary web browser. For each web site we have measured how long it takes to load the first page of search results.

To measure this we used Firefox in combination with the Load Time Analyzer extension, which is developed by Google and measures how long it takes to fully load a web page. These measurements are compared with the multithreaded multi-site version of the Web 2.0 Browser with four threads per site.

The measurements are shown in Table 4.1. With Firefox, we measured how long it takes the load the first page of search results. To improve the accuracy of the measurements, we performed three runs and use the average of those runs for our comparison. The number of items displayed per search result page differs per web site. YouTube, LiveLeak, and Revver display respectively 20, 12, and 15 items per search result page. As last row, the table includes the response time of the Web 2.0 Browser for the equivalent number of items shown per search result page by the web site.

From the measurements, we see that the Web 2.0 Browser is approximately 63% slower than YouTube, 260% slower than LiveLeak, and 40% faster than Revver. Overall, the Web 2.0 Browser is slower than the using an Internet Browser. However, note that the Web 2.0 Browser also fetches the individual page for each item, from which it extracts more metadata that is presented to the user. For example, YouTube and Revver do not provide the associated tags with each item on the search results page, while the Web 2.0 Browser does. However, this requires additional data retrieval, which costs extra time.



Figure 4.2: Announcement of the public release of the Web 2.0 Browser

4.2 Real world usage

The Web 2.0 Browser has been released into the public, which was announced on April 10th 2007 (see Figure 4.2). Through usage of the Web 2.0 Browser by the public, we were able to collect data as a result of Web 2.0 Browser installations contacting the rating server.

In this section we present this data. First, we describe which datasets have been obtained. Subsequently, we present the statistics that have been extracted from these datasets.

4.2.1 Datasets

The rating server is contacted by the deployed Web 2.0 Browsers to retrieve average ratings for items and to post the user rating. Consequently, we have two datasets, the user ratings and the average rating requests. To distinguish the ratings posted and requested from the various Web 2.0 Browser installations, each installation has an associated unique installation ID.

The dataset of user ratings is a collection of records that contain the installation ID, the unique item ID, and the rating. Every time a user rates an item, a message is sent to the rating server a new record is added to the user ratings. A new rating for a particular combination of installation ID and item ID overwrites any previous records with that combination. Thus the most recent ratings issued by an installation are kept in the database.

The dataset of average rating requests is a collection of records that contain the installation ID, the unique item ID, and the time at which the request was submitted. Every time the Web 2.0 Browser needs to display the average rating, the average rating is requested from the rating server. The rating is displayed in two places. First, the rating is displayed on the detail panel of the search results. Thus

an average rating request for an item is sent when it is selected in the search results overview, it does not have to be downloaded. Second, the average rating is also displayed on the viewer tab and is for the current showing or playing item. Consequently, an average rating for an item is requested every time that item is viewed.

The two datasets mentioned above are the only datasets that have been collected by means of the Web 2.0 Browser. However, itemIDs are not very interesting, and therefore we use additional datasets that we extract from the web sites that the Web 2.0 Browser uses. Especially the datasets item IDs and the corresponding tags is useful, because it gives of an indication of the subject of the item's content. However, this dataset is not complete, which is mainly because YouTube has been forced to remove videos from their web site due to copyright violations. Consequently, for a few number of items from our datasets we have no corresponding tags. These items have been omitted from results that involve tags.

4.2.2 Average Rating Requests

In this section, we present the number of installations and the day of the first run of each installation. Furthermore, we look at the most active users and the most popular tags. These statistics have been extracted from the average rating requests.

Number of installations

Practically every installation has sent an average rating request. Along with each request, installation ID is also sent. From the average rating request it is thus possible to extract the number of installations. This number is 420.

The installation ID is generated on the first run after the installation of the Web 2.0 Browser and is partially constructed with the current time. It is possible to extract this time from the installation ID, and therefore we know for each installation at what date and time the installation was run for the first time with a precision of seconds. Figure 4.3 shows for each day since the April 10th the number of installations with the first run on that day.

Most first runs have been performed on the first few days since the release. After one week, 146 installations already had their first run, and after two weeks more than half of the total number of installations, 221, had their first run. After two weeks, each day approximately two new installations had their first run.

Most Active Users

The average number requests can also be used as a loose indication of usage of the Web 2.0 Browser. When the Web 2.0 Browser is used to search for new content and to view items, rating requests are sent to the rating server, and thus indicates that that specific installation is actively used. By counting the number of rating requests for each installation ID, we have an indication of how actively each installation has

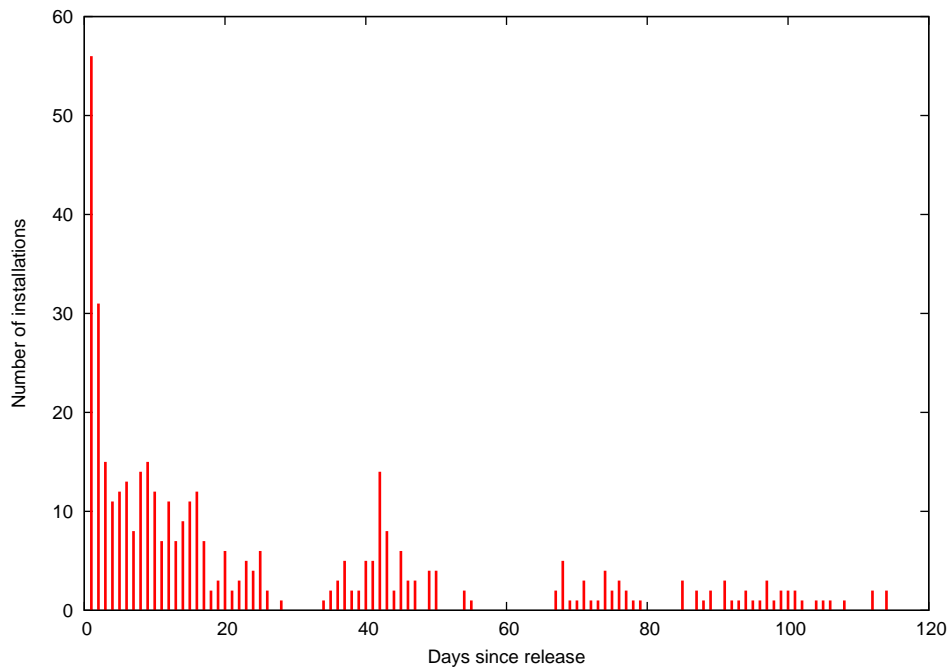


Figure 4.3: The number of first runs of installations since April 10th, 2007

been used. Figure 4.4 shows the twenty most actively used installations with the corresponding number of rating requests sent to the rating server.

The most used installation is extremely active compared to the other installations. The most used installation has sent more than 2.5 times more rating requests than the number of rating requests sent by the second most used installation. The difference among the other installation is much less. The number of rating requests decreases more moderately between installations.

Most Viewed Content

In this section, we consider the content that was viewed by the Web 2.0 Browser users. To get an indication of the type of content that has been looked at, we use the tags that are associated with a particular item as an indication of the content of the item. We combine the average rating requests with the associated tags for each item, and the tags are ranked by the number of rating requests for items that are associated with the particular tag. Figure 4.5 shows the 100 tags with the most number of rating requests in the form of a tagcloud.

The tagcloud shows that was a strong interest in the musical event that was held on queen's day by radio station "radio 538" at the museumplein in Amsterdam. Furthermore there was a strong interest in a tv show called "Simplisties Verbond" starring Van Kooten and De Bie. Both subjects are Dutch orientated, which is not surprising because the public release of the Web 2.0 Browser was announced on a

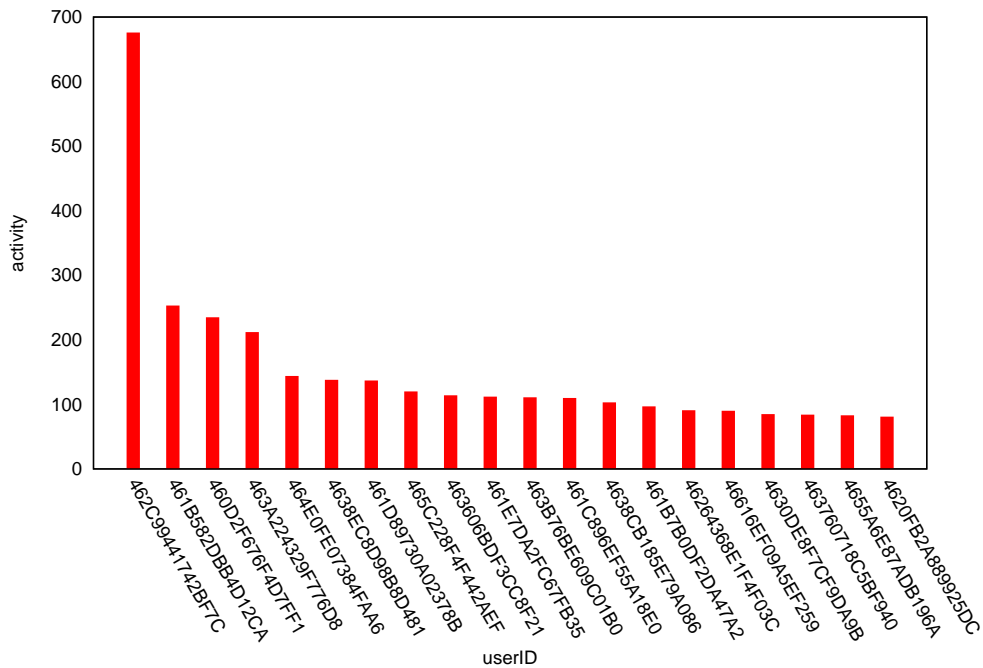


Figure 4.4: The 20 most active users and their activity.

2007 538 amsterdam and animation art arts ass babe babes beach belly bie
 bikini bloopers boobs breast butt comedy commercial cool cute dance dansen de dirk dj
 dutch en eric erotic fetish fight fu fun funny gar girl girls gung het
 hot humor hung ijs in juinen jump jumpen karate kees
 koninginnedag koot&bie kooten kuen kung kungfu lau lee live love
 man marion martial movie museumplein music naked nathan
 nederlands new nude of op porn radio retro sex sexy shaolin show
 simplisties simplistisch sita sterren the thong tits to tv van video
 vieze vpro wallpaper war webcam wild xxx you

Figure 4.5: The 100 tags with the most rating requests.

LiveLeak	Revver	YouTube
humor	sexy	dutch
crash	hot	kooten
video	girl	bie
funny	babe	the
commercial	cute	vieze
wtf	bikini	vpro
iraq	girls	kees
tv	boobs	simplisties
cam	fetish	nederlands
animation	tits	retro

Table 4.2: The top ten tags with greatest covariance with the site for each site

Dutch web site, so most of the users are likely Dutch.

Site And Content

In section 4.1, we claimed that one of the advantages of combining multiple sites is the greater variety of content because sites usually have a focus on a subject or audience. In this section, we support this claim by looking at the content that is delivered by the three different videos sites.

To find out the type of content delivered by the three different sites, we have computed the covariance between the web sites and tags. The dataset that was used to compute the covariances is the unique items from the average rating requests with the associated tags and the site that delivered the item. Table 4.2 shows the ten tags with the strongest covariances with each site.

YouTube is a very popular video site with many uploaders. Consequently its video collection is very diverse, and it even has dutch content. LiveLeak and Revver are less popular than YouTube and there is much less dutch content available from these sites.

The content that has been provided by Revver is mainly adult oriented. Although adult content is not the primary focus of Revver, it is not excluded by the site as YouTube does. Thus Revver is available to provide certain content that is not available from YouTube or LiveLeak.

Most of the tags that are correlated with LiveLeak have a more general meaning and do not point to a specific subject. The tags “humor” and “funny” indicate humorous videos. The tag “iraq” indicates videos on the war in Iraq. LiveLeak has a focus on news videos and the site even has a separate category for the war in Iraq.

4.2.3 Ratings

In this section we present the collected data from user ratings. In total, 291 ratings have been issued by Web 2.0 Browser users. Table 4.3 shows the distribution of

Rating	Number	Percentage
1	33	11.3%
2	18	6.2%
3	26	8.9%
4	41	14.1%
5	173	59.5%

Table 4.3: The distribution of the ratings.

Web Site	No. ratings	Average rating
LiveLeak	23	3.91
Revver	24	2.63
YouTube	228	4.28
Flickr	12	3.25
Zoomr	2	2.00
Wikipedia	2	2.50
Total	291	4.04

Table 4.4: The number of ratings and the average ratings per site.

the ratings over the rating levels.

The ratings are strongly biased towards a positive rating, i.e., four or five stars. Nearly 60% of all ratings have a level of five stars, and nearly 74% is either four or five stars. Among the negative ratings, the one star rating is issued the most. Either users find all videos of good quality or users tend to only vote for quality items and do not vote at all for bad quality items.

Table 4.4 shows the total number of ratings and the average rating for each site. By far, the most ratings were issued for items that come from YouTube, and users were also the most positive about the content of YouTube compared to the other web sites.

4.3 Khashmir

In this section, we discuss the behavior and performance of Khashmir. For the sake of clarity, we refer to hosts participating in a BitTorrent swarm as peers and hosts participating in Khashmir as nodes.

A single swarm discovery operation in Khashmir does not yield a single list of peers that Khashmir found to be in the particular BitTorrent swarm at the end of the operation. The storage of a swarm is distributed across multiple Khashmir nodes, as soon as Khashmir encounters a node that knows of peers in a particular swarm, then these peers are returned immediately, and Khashmir continues searching for more nodes that knows of peers in the swarm. So the results of a swarm discovery operation consists of tuples of a timestamp and a list of peers.

In this section, we first show the behaviour of Khashmir for a large swarm. From

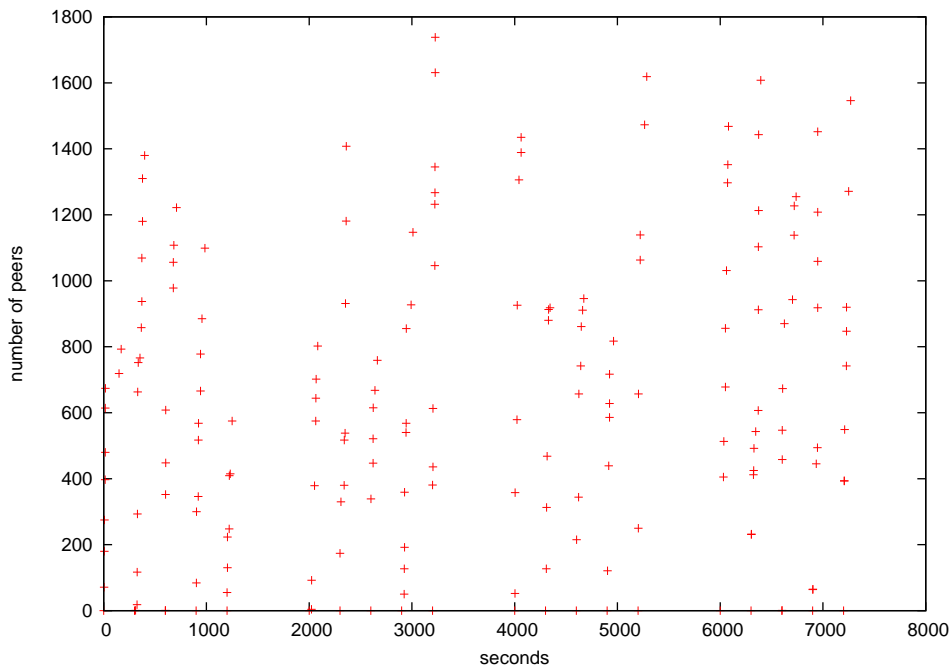


Figure 4.6: Swarm discovery operations for a large swarm.

these measurements, we conclude that the timeout value used by Khashmir could be set much stricter in order to improve performance.

4.3.1 Khashmir behaviour

Figure 4.6 shows the result of regular swarm discovery with Khashmir for the most popular torrent in the category “TV Shows” from Mininova (mininova.org). The figure shows four runs at an interval of 2000 seconds. A single run consists of 5 swarm discovery operations with an interval of 5 minutes. Before each run, the routing table is flushed, causing the node to forget any peers it has learnt in a previous run. The Khashmir node performing the measurements is bootstrapped with a single other stable Khashmir node.

Peers found in the swarm by Khashmir are indicated by dots. The number of peers found as indicated by Figure 4.6 are accumulative within a single swarm discovery operation.

From Figure 4.6 we notice that the number of peers yielded by different swarm discovery operations vary strongly from 575 to 1738. And there seems no relation to whether the routing table has just been flushed or whether it is already filled with nodes. This strong variation occurs because the swarm storage is distributed across many peers, while in a single run only a few of these peers is contacted. If the results of all the discovery operations are merged, then we find 11507 peers in the swarm. However, this swarm information is retrieved from 88 different Khashmir

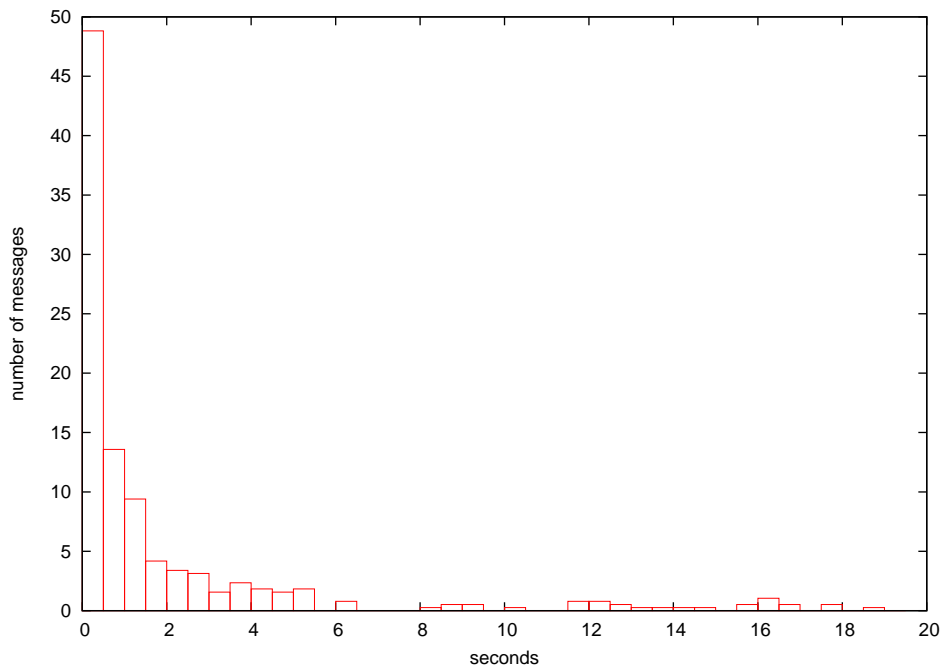


Figure 4.7:

nodes. During a single discovery, only a small number of nodes that have swarm information is contacted. In the current implementation this number is at most 8. For all swarm discovery operations, in total 792 requests were sent to other Khashmir nodes. For 52% of these requests, no response was received. There may be multiple reasons for not receiving a response. First, the node to which the request was sent may be offline. This means that our rating table or routing tables of other nodes contain stale entries. Second, the request or the response may never have reached its target due to the unreliability of UDP traffic. Finally, the node to which the request was sent did not respond in order to limit the upload rate. The Khashmir implementation that we used has a rate limiter that limits the upload rate of sent responses. The rate limitation is enforced by simply not sending responses. The default Khashmir upload rate limit is 1% of the maximum upload rate for BitTorrent, which is a user setting.

Figure 4.7 shows the distribution of the latency of the responses. A vast majority of the responses have a latency of a few seconds.

4.3.2 Khashmir Timeout

The number of concurrent outstanding requests of a DHT operation is limited by Khashmir. As shown in the previous experiment, on average, 52% of the requests receive no response. For these requests, Khashmir waits for a timeout, which is 20 seconds. While waiting for a timeout, no new request can be sent to another node

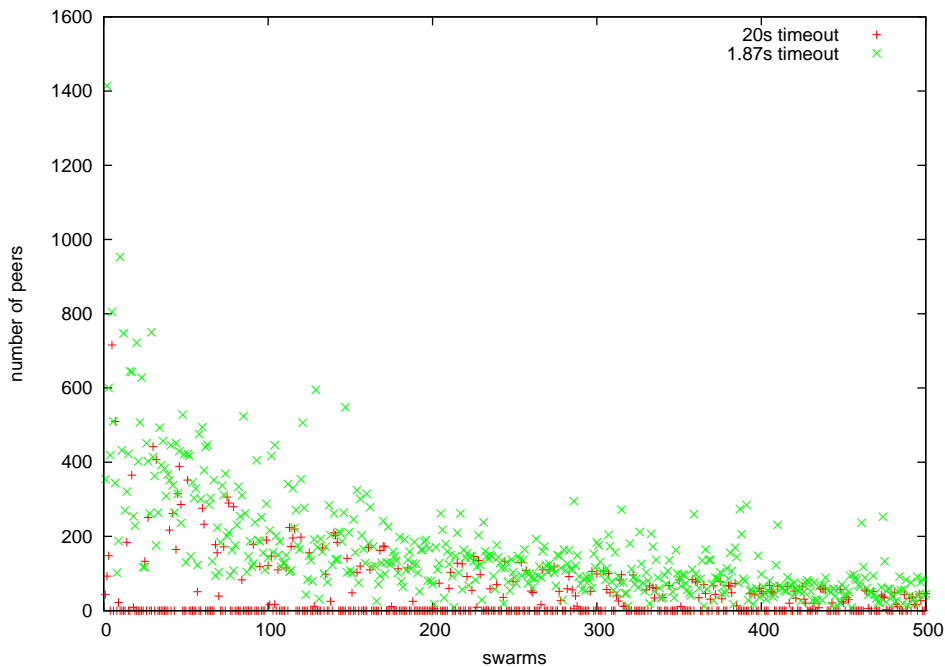


Figure 4.8: Number of peers discovered after 10 seconds.

due to the limitation on the number of outstanding requests. Considering the 20 second timeout and the 52% unsuccessful requests, Khashmir spends a significant amount of time waiting for responses that will not come.

Furthermore, note from Figure 4.7 that most response arrive within a few seconds. Therefore, the time spent on waiting for timeouts can significantly be decreased by using a much smaller timeout value at the cost of few responses that will arrive too late and will be discarded.

We aim to maximize the performance by exploring the boundaries of Khashmir and will test a very strict timeout value. We set the timeout to the point at which 75% responses will be in time, which is based on the dataset of the previous 1.87 second. To compare the stricter timeout with the default 20 second timeout, we have run swarm discovery operations for over 1000 torrents, once using a 20 second timeout and once using 1.87 second timeout. The torrents selected by taken the most popular torrents from `mininova.org` in the category Movies and TV Shows. The torrents are sorted by the total number of peers that is found with a 20 second timeout Khashmir swarm discovery operation.

Figure 4.8 shows for the top 500 torrents the number of peers that is discovered 10 seconds after the swarm discovery operation is started. It is clear that using a 1.87 second timeout yields much better results after ten seconds. With a 20 second timeout, for 62.4% of the torrents no peers at all were yet found after ten seconds. Figure 4.9 shows for the top 500 torrents the number of peers that is discovered at the end of the swarm discovery operation. In general, the 20 second timeout

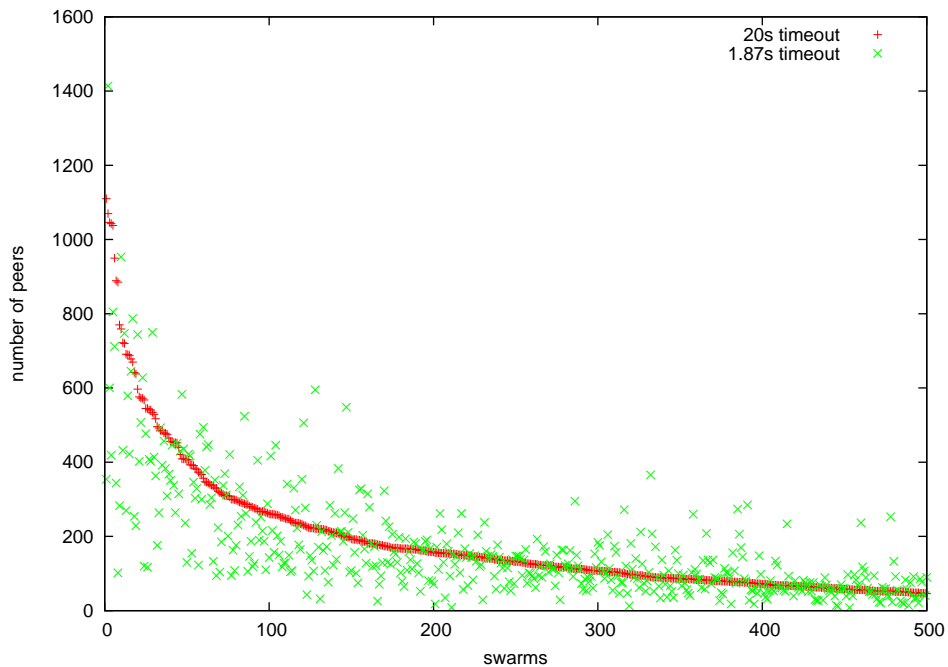


Figure 4.9: Number of peers discovered at the end of the swarm discovery operation.

Khashmir finds more peers.

25% of the responses that would be in time with a 20 second timeout fails to meet the 1.87 second timeout. This loss of messages, causes that fewer peers are eventually found in Khashmir. Note the similarity of both graphs for 1.87 second timeout. For 85.4% of the torrents, no new peers are learnt after ten seconds, i.e., the swarm discovery operation has finished.

By using two different timeout values, a short and a long timeout, the best of both worlds can be achieved. When the short timeout times out for a request, new requests can be send to other nodes, however Khashmir will still wait for the response or the long timeout before giving up the request. This combination of timeouts delivers fast results like using a short timeout and still finds all the peers that would be found by using a long timeout.

Furthermore, this experiment shows clearly the main advantage of Khashmir over the traditional tracker, the independence of a central component. In total for 1016 torrents a Khashmir swarm discover operation has been performed. For each of these torrents, the scrape extension of the tracker was contacted, up to three attempts, to learn the swarm size according the tracker. For 38.4% of the torrents the tracker failed to respond. For another 2.3%, the tracker indicated the swarm was empty. Khashmir found an empty swarm for 2.6% of the torrents, and found at least one peer for the rest of the swarms.

4.4 End-to-End Video

In this section, we present the entire process of downloading a Web 2.0 video to the distribution of it to other peers with the Tribler network. With this experiment, we show that all processes of data exchange occur within the Tribler system. So our system is able to provide end-to-end delivery of videos, photos, etc.

This experiment presents the logs of two clients. Figure 4.10 shows the log of the Tribler client that views a Web 2.0 video and spreads it with Tribler. We call this client the *publisher*. Figure 4.11 shows the log of a Tribler client that downloads the Web 2.0 video that is distributed by the publisher. The logs have been filtered to only display relevant messages. We call this client the *consumer*. Both clients have their own clocks, therefore they may be a small skew between timestamps of both logs.

The publisher performs a keyword search. From the search results, the publisher downloads a video that is retrieved from a Web 2.0 site. Once the download is complete, the publisher creates a torrent file for the downloaded video, and announces itself as a seeder for the video in Khashmir. The torrent file is spread to other peers by Tribler.

The consumer searches on keyword. Note, this search also searches for torrents. Subsequently, the consumer receives the video torrent from the publisher and the torrent is displayed in the search results, which is then downloaded. The consumer finds the publisher as seeder in a Khashmir swarm lookup, and downloads the video from the seeder. This completes the publishing and distribution of a video that is retrieved from an external web site.

Except for the content that is taken from an external web site, there is no dependence on an external system to get the data from the publisher to the consumer. Furthermore, no supplementary action of the user is required in addition to selecting the item it wants to publish or consume. If the publisher publishes an item that it owns, i.e., an item that is stored locally at the publisher, then there is no dependence on any system outside Tribler.

```
fabian@laptop: ~/web2browser/bcexperiment
15:40:05.304 Web2: search for "p2p"
15:40:13.837 Web2: downloading item "Peer-to-peer"
15:40:49.403 Web2: created .torrent file for "Peer-to-peer"
15:40:49.664 Torrent: torrent completed d6cc0cda9e1b3f3c2c98777d2cb2951df90b173f
15:42:25.205 Khashmir: announce for d6cc0cda9e1b3f3c2c98777d2cb2951df90b173f
15:42:49.859 Khashmir: found nodes in swarm d6cc0cda9e1b3f3c2c98777d2cb2951df90b173f:
130.161.158.60:7762
15:44:10.003 Khashmir: found nodes in swarm d6cc0cda9e1b3f3c2c98777d2cb2951df90b173f:
130.161.158.60:7762
130.161.158.216:7762
15:44:14.282 Khashmir: announce for d6cc0cda9e1b3f3c2c98777d2cb2951df90b173f
15:44:52.298 Khashmir: found nodes in swarm d6cc0cda9e1b3f3c2c98777d2cb2951df90b173f:
130.161.158.216:7762
15:44:53.033 Khashmir: announce for d6cc0cda9e1b3f3c2c98777d2cb2951df90b173f
15:45:13.283 Khashmir: found nodes in swarm d6cc0cda9e1b3f3c2c98777d2cb2951df90b173f:
130.161.158.60:7762
15:45:30.935 Khashmir: announce for d6cc0cda9e1b3f3c2c98777d2cb2951df90b173f
15:45:50.118 Khashmir: found nodes in swarm d6cc0cda9e1b3f3c2c98777d2cb2951df90b173f:
130.161.158.60:7762
130.161.158.216:7762
15:45:57.961 Khashmir: announce for d6cc0cda9e1b3f3c2c98777d2cb2951df90b173f
15:46:34.455 Khashmir: announce for d6cc0cda9e1b3f3c2c98777d2cb2951df90b173f
15:46:56.342 Khashmir: found nodes in swarm d6cc0cda9e1b3f3c2c98777d2cb2951df90b173f:
130.161.158.60:7762
130.161.158.216:7762
15:47:11.975 Overlay: sent torrent d6cc0cda9e1b3f3c2c98777d2cb2951df90b173f
15:47:52.342 Khashmir: found nodes in swarm d6cc0cda9e1b3f3c2c98777d2cb2951df90b173f:
130.161.158.216:7762
15:47:52.983 Khashmir: found nodes in swarm d6cc0cda9e1b3f3c2c98777d2cb2951df90b173f:
130.161.158.60:7762
```

Figure 4.10: Log of the publisher with IP 130.161.158.60.

```
fabian@laptop: ~/web2browser/bcexperiment
15:46:48.219 Web2: search for "peer"
15:47:39.374 Overlay: received torrent d6cc0cda9e1b3f3c2c98777d2cb2951df90b173f
15:47:40.442 Torrent: download started d6cc0cda9e1b3f3c2c98777d2cb2951df90b173f
15:47:58.979 Khashmir: lookup for d6cc0cda9e1b3f3c2c98777d2cb2951df90b173f
15:48:21.501 Khashmir: found nodes in swarm d6cc0cda9e1b3f3c2c98777d2cb2951df90b173f:
130.161.158.60:7762
15:48:50.441 Torrent: torrent d6cc0cda9e1b3f3c2c98777d2cb2951df90b173f completed
```

Figure 4.11: Log of the consumer with IP 130.161.158.216.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

Our research is bipartite. First, we have developed accelerated interfaces towards sharing web sites, which can easily be aggregated. The Web 2.0 interfaces provide a simple interface to search in the collection of a Web 2.0 sharing site. The Web 2.0 interface transforms items into a uniform representation independent of the site. Consequently, the end user is not able to distinguish items from different sites.

By use of multithreading the Web 2.0 interfaces have been substantially accelerated. Furthermore, multiple Web 2.0 interfaces can be combined into a single Web 2.0 interface, which allows for aggregation of data from many sites. Because of the easily extendable framework, developing a Web 2.0 does not require much time and can easily be aggregation with other interfaces. Because our research overarches every existing Web 2.0 sharing system we consider this a disruptive technology.

Second, we have extended Tribler such that it is capable of end-to-end delivery of video, music, and other media. Where other peer-to-peer distribution systems still rely on central components for the publication of items, our system does not depend on a single central component for publication. As a consequence, the entire flow of data distribution, from publication to the delivery to the end users, occurs completely within the Tribler network and benefits from its scalability.

5.2 Future Work

In this section we discuss how our system can further be improved in terms of performance and functionality.

5.2.1 Performance

The performance of the web interfaces can further be improved by restructuring the data retrieval. The data retrieval can be divided in multiple stages, and in each

stage only a part of metadata is retrieved. For example, at first only the title of an item is fetched, then the thumbnail, and finally the more detailed metadata. The basic metadata is fetched quicker and can directly be presented to the user while the other metadata is fetched.

Another improvement, is to delay the retrieval of detailed metadata until the user decides to view the detailed metadata. Thus not all metadata of all items is always fetched. This reduces amount of communication required to fetch search results, and thus presents search results to the user more quickly.

5.2.2 Functionality

Compared to Web 2.0 web sites, we feel that the community tools provided by Tribler come short. Web 2.0 web sites lets a user comment items, and send messages to other users. From our experience, commenting is the primary tool that is used to communicate with other users.

In addition to the community tools, the collection offered by sharing sites are usually more complex than a set of items. Items are often interrelated. For example, photos on sites such as Flickr and Zoomr are organized in photo albums. YouTube has the concept of video channels and *video responses*. The concept of related items is also present in texts. Texts available from Wikipedia, usually contain several links to other Wikipedia texts, which are on related subjects.

Bibliography

- [1] Scott A. Crosby and Dan S. Wallach. An analysis of bittorrent's two kademia-based dhds. Technical Report TR07-04, Rice University, May 2007.
- [2] Dan Frommer. Your tube, whose dime?, April 2006. http://www.forbes.com/intelligentinfrastructure/2006/04/27/video-youtube-myspace_cx_df_0428video.html.
- [3] Lee Gomes. Will all of us get our 15 minutes on a youtube video?, August 2006. http://online.wsj.com/public/article/SB115689298168048904-5wWyrSwyn6RfVfz9NwLk774VUWc_20070829.html.
- [4] ipoque GmbH. ipoque's 2007 p2p survey to be presented at technology review's emerging technologies conference at mit, August 2007. http://www.ipoque.com/media/news/ipoques_2007_p2p_survey_to_be_presented_at_technology_reviews_emerging_technologies_conference_at_mit.html.
- [5] Petar Maymounkov and David Mazières. Kademia: A peer-to-peer information system based on the xor metric. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 53–65, London, UK, 2002. Springer-Verlag.
- [6] Tim O'Reilly. What is web 2.0: Design patterns and business models for the next generations software, September 2005. <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web20.html>.
- [7] Inc. Wikimedia Foundation. Frequently asked question - wikimedia foundation. <http://wikimediafoundation.org/wiki/Wikimedia:About>.
- [8] Inc. Wikimedia Foundation. Financial statements, june 30, 2006, 2005, and 2004, November 2006. http://upload.wikimedia.org/wikipedia/foundation/2/28/Wikimedia_2006_fs.pdf.