A Scalable Web 2.0 Platform

Fabian van der Werf
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

4th November 2007
Abstract

TODO ABSTRACT
Preface

TODO MOTIVATION FOR RESEARCH TOPIC

TODO ACKNOWLEDGEMENTS

Fabian van der Werf

Delft, The Netherlands
4th November 2007
## Contents

<table>
<thead>
<tr>
<th>Preface</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Web 2.0</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Peer-to-peer networks</td>
<td>4</td>
</tr>
<tr>
<td>1.2.1 BitTorrent</td>
<td>5</td>
</tr>
<tr>
<td>1.2.2 Tribler</td>
<td>7</td>
</tr>
<tr>
<td>1.3 Outline</td>
<td>8</td>
</tr>
<tr>
<td>2 Problem Description</td>
<td>9</td>
</tr>
<tr>
<td>2.1 Scalability and Robustness</td>
<td>9</td>
</tr>
<tr>
<td>2.2 Our Aim</td>
<td>11</td>
</tr>
<tr>
<td>2.2.1 YouTube</td>
<td>11</td>
</tr>
<tr>
<td>2.2.2 Tribler</td>
<td>12</td>
</tr>
<tr>
<td>2.2.3 Our Vision</td>
<td>13</td>
</tr>
<tr>
<td>2.3 Related Work</td>
<td>14</td>
</tr>
<tr>
<td>2.3.1 Vuze</td>
<td>14</td>
</tr>
<tr>
<td>2.3.2 Joost</td>
<td>16</td>
</tr>
<tr>
<td>2.3.3 Babelgum</td>
<td>16</td>
</tr>
<tr>
<td>3 Design And Implementation</td>
<td>19</td>
</tr>
<tr>
<td>3.1 Software Development Process and Functionality</td>
<td>20</td>
</tr>
<tr>
<td>3.1.1 Single-threaded prototype</td>
<td>20</td>
</tr>
<tr>
<td>3.1.2 Multithreaded, multi-site prototype – Web 2.0 Browser</td>
<td>20</td>
</tr>
<tr>
<td>3.1.3 Integration of Video Browsing with P2P</td>
<td>22</td>
</tr>
<tr>
<td>3.2 Web 2.0 Browser</td>
<td>23</td>
</tr>
<tr>
<td>3.2.1 Architecture</td>
<td>23</td>
</tr>
<tr>
<td>3.2.2 GUI design</td>
<td>26</td>
</tr>
<tr>
<td>3.2.3 Web 2.0 Interfaces</td>
<td>28</td>
</tr>
<tr>
<td>3.2.4 Ratings</td>
<td>31</td>
</tr>
<tr>
<td>3.2.5 Download Manager</td>
<td>32</td>
</tr>
<tr>
<td>3.3 Tribler Integration</td>
<td>32</td>
</tr>
<tr>
<td>3.3.1 Integration with Tribler</td>
<td>33</td>
</tr>
</tbody>
</table>
3.3.2 Decentralized Tracking ........................................... 34

4 Experiments And Evaluation ........................................... 39
  4.1 Web 2.0 Interfaces .................................................. 39
  4.2 Real world usage .................................................... 43
    4.2.1 Datasets ....................................................... 43
    4.2.2 Average Rating Requests ..................................... 44
    4.2.3 Ratings ....................................................... 48
  4.3 Khashmir ............................................................. 50
    4.3.1 Large Public Swarms ........................................... 50
    4.3.2 One Peer Swarms .............................................. 53

5 Conclusions and Future Work ........................................ 57
  5.1 Conclusions .......................................................... 57
    5.1.1 Goal ............................................................ 57
    5.1.2 Our Solution ................................................... 58
    5.1.3 Viability ........................................................ 58
  5.2 Future Work ........................................................ 59
    5.2.1 Performance Improvements .................................... 59
    5.2.2 Web 2.0 Functionality ......................................... 60
    5.2.3 Hybrid Downloads .............................................. 61
Chapter 1

Introduction

Originally, the World Wide Web (WWW) was a collection of a static web pages which an Internet surfer could look up. The last few years, the data and services offered via the WWW have changed, and this change is referred to as Web 2.0[4]. There is no fixed checklist according which a web site can be qualified 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 to create web applications with rich user experiences. Web 2.0 is a new approach to creating web applications and is usually associated with a new role of users. 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, because the data is collected only from browsers equipped with the Alexa Toolbar. It gives a clear indication of the significance and success of Web 2.0 applications, though. Six out of ten sites in the top 10 are Web 2.0 sites, 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. Our aim is to create a scalable Web 2.0 platform without 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. Furthermore, the platform has to keep track of the popularity of items as most Web 2.0 applications do.

This section gives an introduction into the world of Web 2.0 and peer-to-peer applications which will provide a scalable environment. Finally, an outline of the remainder of this thesis is given.
<table>
<thead>
<tr>
<th>Rank</th>
<th>Web Site</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Yahoo!</td>
<td><a href="http://www.yahoo.com">www.yahoo.com</a></td>
</tr>
<tr>
<td>2.</td>
<td>Microsoft Network (MSN)</td>
<td><a href="http://www.msn.com">www.msn.com</a></td>
</tr>
<tr>
<td>3.</td>
<td>Google</td>
<td><a href="http://www.google.com">www.google.com</a></td>
</tr>
<tr>
<td>4.</td>
<td>YouTube</td>
<td><a href="http://www.youtube.com">www.youtube.com</a></td>
</tr>
<tr>
<td>5.</td>
<td>Windows Live</td>
<td><a href="http://www.live.com">www.live.com</a></td>
</tr>
<tr>
<td>7.</td>
<td>Baidu.com</td>
<td><a href="http://www.baidu.com">www.baidu.com</a></td>
</tr>
<tr>
<td>8.</td>
<td>Orkut</td>
<td><a href="http://www.orkut.com">www.orkut.com</a></td>
</tr>
<tr>
<td>10.</td>
<td>Tencent QQ</td>
<td><a href="http://www.qq.com">www.qq.com</a></td>
</tr>
</tbody>
</table>

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 behind Web 2.0 is to use the WWW as a platform for web applications. The meme map contains six core competencies.

These six competencies do not form a checklist to check whether an application is Web 2.0 or not, but are characteristics of successful 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, and are therefore accessible from any platform without the need to port it each specific platform as 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 mostly used to be one way traffic of information, i.e., users looked up information from web pages but did not contribute 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 still 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 another 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 to allow multiple persons working on the same document.

Harnessing collective intelligence  The classic example of a 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 usually the only reason why a node is participating in a P2P system.

P2P networks have gained a lot of popularity. Research shows that Internet traffic is dominated by P2P traffic. By the end of 2004, 60% 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 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 the downloaded the file partially. Seeders are peers who already have a complete copy of the file and upload to 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 connects with 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 simplified BitTorrent network.

Peers exchange data in a tit-for-tat manner. If a peer wants to increase its download rate, it will 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.
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 a web server that hosts the .torrent files. Examples of such sites are mininova.org, piratebay.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 mostly 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 take 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.

The Tribler tries to find other users with a similar taste, and using this information it provides recommendations of other files. Tribler searches for other users with a similar taste 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 unless the file has been deleted again. 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 you 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 our scalable Web 2.0 platform is described. Our platform builds upon Tribler and leverages its scalability, and we discuss the adaptations and extensions made to Tribler. Chapter 4 presents experiments and evaluation: performance measurements as well as community measurements of the Web 2.0 Browser are presented. The Web 2.0 Browser is an application which provides an alternative way to search for videos, photos, and articles on multiple Web 2.0 sites and to view them as well. Finally, Chapter 5 gives our conclusions and future work.
Chapter 2

Problem Description

Web 2.0 applications usually deliver their services 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. Peer-to-peer networks, on the contrary, perform better as the user base grows.

In this chapter, we start by examining the problem of un scalable 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 draw up the criteria of a potential solution. 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 then, we take the best of both worlds. Finally, we describe current solutions to the problem and indicate why these attempts are not satisfying. Thereby, the need for a new scalable Web 2.0 platform is justified.

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 website 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 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, 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, 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 to 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, so this raises the question to what extent will organisations be able to continue to expand server farms before the costs of keeping the service running become too high to actually keep it running.

WikiMedia

Take for example 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[5]. One of their most well-known projects is 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 also Table 1.1).

The total expenses for WikiMedia have increased tremendously for the last few years. For the years 2004 to 2006, the total expenses were respectively $23,463, $177,670, and $791,907[6]. WikiMedia is a foundation, and therefore, these expenses have to be covered by donations and gifts. Considering the growing popularity of Wikipedia, it is likely that the costs will continue to grow. Will WikiMedia in a few years still to be able to cover the fast growing expenses with donations and gifts?

YouTube

YouTube is another web site that which have high expenses to keep it running. The size of videos several magnitudes larger than of photos and articles, and therefore, the amount of required bandwidth and storage are higher than those of similar site for photos or articles with the same userbase.

In April 2006, the total amount of bandwidth usage by YouTube was estimated at 200 terabytes per day[1][2]. 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 huge, and accordingly, 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 let users 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 intend 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 as one of the best there is at the moment. Besides, 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 to formulate the requirements.

2.2.1 YouTube

YouTube is to 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 fills in a title, description, tags, and category, and then 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 other 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, for users do not like to wait. YouTube provides streaming video
with Flash, so the user does not need any third-party software to play back the video.

**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. YouTube has two other figures which gives an indication of the appreciation of the community. Each user can keep a list of favorite videos. That is the number of views combined with the number of times the video has been favorited by a viewer.

**Wealth Of Content**  The number of videos hosted by YouTube is immense. The total amount of video is estimated at 45 terabytes[2]. This immense amount of videos contributes to the popularity and success of YouTube, for more videos attracts more users.

### 2.2.2 Tribler

In this section, we analyze the functionality of Tribler.

**Decentralization And Scalability**  Tribler is a decentralized system, and does, therefore, not have the scalability issues from which centralized architectures suffer. This is the most important quality of Tribler, for 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. Because, with the client/server architecture, all content is stored on servers that is controlled by a single authority. With Tribler, the only component is the client software with which it is not possible to control the network or its content.

**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. So published HD videos are still HD when they are downloaded.

**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 less rare pieces.

The *play ASAP* mode is intended for video and audio downloads. The policy for this mode, prefers pieces at the start of file over pieces at the end of the file. Audio
Table 2.1: A summary of the functionality of Web 2.0 applications, Tribler, and our vision of a scalable Web 2.0 platform.

<table>
<thead>
<tr>
<th></th>
<th>Web 2.0</th>
<th>Tribler</th>
<th>Our Vision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ease Of Use</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>HD Videos</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Community</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Scalability</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Video-on-Demand</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Wealth Of Content</td>
<td>×</td>
<td></td>
<td>×</td>
</tr>
</tbody>
</table>

and video files downloaded in play ASAP mode can be played back as if they are being streamed. Once the first piece is downloaded video playback can be started, and the other pieces will be downloaded when they are necessary. 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 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 that easy. To publish an item, the user needs to take the same steps as publishing with BitTorrent. Thus, a user needs a tracker that is always running, and it needs to 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 video has 27 times more picture information compared to YouTube videos. Recall from Section 2.1 that the operating costs of YouTube are huge. If YouTube would provide HD videos, then the costs would probably too high to keep YouTube running. Thus it is the scalability that makes HD videos possible. To take advantage of 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 network. Because then, our network no longer depends on the external site for that specific item, and further distribution of that item will benefit of the scalability of our platform. To increase usability, our platform integrates with these sites seamlessly, and users would not be able to tell whether an item comes 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 also available.

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 has become 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 also search for content using the keyword search. Finally, users can browse content by applying criteria such as popularity, duration, price, and date of publication on the content. Downloading of items occurs through the BitTorrent protocol, actually every item in Vuze is a torrent download. The downloaded copy of the file is identical to the original version that was uploaded. There is no transformation of files. 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 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 vary from 10 to 30 gigabytes which must all be downloaded before the movie can watched. This requirement is a consequence of the BitTorrent protocol. To achieve an uniform distribution of the availability of pieces of files BitTorrent clients have as policy to download the rarest piece first. Given the availability of the pieces, the policy is deterministic. However, the availability of pieces depends on the behavior of the other peers in the swarm which cannot be predicted. Therefore, it is unclear in which order the pieces of a file will be downloaded. On the other hand, Video-on-Demand requires the files are downloaded in order, i.e., first pieces first.
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.

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 open up the API for creating new widgets to allow external developers to write plug-ins. The widget system is thus actually a flexible plug-in system which allows add extra functionality to Joost.

Joost is not a free publishing platform like Vuze is. Users are not allowed to publish any videos, and therefore, Joost does not satisfy our criteria of a scalable Web 2.0 platform. 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 programs 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 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. Publishing is reserved for professional and semi professional content owners and
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 in action
requires to first contact Babelgum.
Chapter 3

Design And Implementation

This chapter presents the design of our Web 2.0 platform system which does not suffer of 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 the .torrent file gossiping capability. 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 written 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 requirements of the Web 2.0 Browser. 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 describe the replacement for traditional trackers. This replacements removes the need for publishers to set up a tracker.
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 with 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 BitTorrent distribution system.

Besides these requirements, there are no clear requirements and it is not clear what functionality the Web 2.0 Browser will have. Due to the experimental nature of development of the proof-of-concept, we took a exploratory approach to the implementation. The implementation is evaluated at regular intervals to identify what components have to be improved and what features are desirable.

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 stage 1. 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. Video playback is provided by using an external video player. 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. To increase the performance of search results, all retrieved metadata and items are stored in an unlimited on-disk cache.

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,
Figure 3.1: First Prototype
• combining multiple sites,
• improved GUI with integrated viewer,
• cross Web 2.0 sites rating, and
• iPod video encoding for videos.

In the first prototype explicitly selected on which site the search operation had to be performed by selecting. 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, which is performed by the GUI. 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 Web 2.0 Browser no longer depends on an external video player for video playback. The integrated viewer is capable of handling all downloaded items, that is videos, photos, and HTML.

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 vast 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.

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. Furthermore, users do no have to install a separate application. However, a Firefox extension has also several limitations. Usage is limited to Firefox users, users with different browser will not be able to the application. Thus the large user group of Internet Explorer will not be able to use the Web 2.0 Browser. In addition, Firefox extension 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.

Virtually all of the network connections are used for retrieving web pages and thus carry HTTP traffic. The Python standard library provides a module to retrieve web pages which uses blocking IO operations. Because of the blocking IO operations, using this modules implies using a Multi-threaded IO approach. Taking the asynchronous IO approach means that an third-party library would be necessary to handle HTTP, or that we have to develop our own HTTP handler that uses asynchronous communication. In order to minimize dependencies on third-party
libraries and shorten development time, we chose to take the multi-threaded approach.

**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 is 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 clear 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 usually is often 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 needless 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
Figure 3.2: The architecture of the Web 2.0 Browser.
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 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.

### 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 3.4. 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.

Figure 3.3 shows a screenshot of the Web 2.0 Browser.

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 to view these items.
Precaching

The search results are divided over multiple pages like most websites 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 waiting time for the user for viewing the next page is reduced or even eliminated.

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 operation, 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 improve responsiveness of the application, results 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 a single thread only, 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 the item which contains extended metadata. ThreadedDBSearch exploits this structure allows 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 to 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 webpage of the item and extract metadata. Before actually downloading the webpage, `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 create 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 webpage. First, a site may provide an API for external applications. Such an API provides 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 be 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 expressions. 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 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 a trial-and-error process, they need to be restrictive enough that only the desired data is extracted and they need to work correctly for all items. From our experience, this takes only a few hours of programming.

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. To reduce communication, any ratings retrieved from the rating server are cached and remain valid for a short period. 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 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. Thus 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 fits in with the second modification of Tribler that is required, 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 set 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 swarm discovery method that is called decentralized tracking. With decentralized tracking, each peer becomes a potential tracker for each 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.
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 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.

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

In this section, we first present the integration of the Web 2.0 interfaces with Tribler, and subsequently, the decentralized tracking is explained.
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 a 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 is then passed on to the Tribler Library. 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, 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. If a DHT is the only swarm discovery for a particular torrent, then there will be two separate swarms for that torrent, one swarm in the Mainline DHT and one in the Azureus DHT. For a BitTorrent client to discover all the peers in the DHTs, it needs implementations for both DHTs.

We have chosen to integrate the DHT from the Mainline BitTorrent client into Tribler mainly to reduce development time. The Mainline DHT can easily be integrated with Tribler, because just like Tribler it is written in Python. The Azureus DHT is written in Java, and therefore, for integration with Tribler first a Python version would have to be developed. Furthermore, at the moment, there is no documentation available of the Azureus DHT thus porting it to Python also requires to elicit its working from the Azureus source code.

We have integrated the Mainline DHT with Tribler. The Mainline DHT is called Khashmir, and is based on Kademlia[3]. This section first explains Kademlia, and then the modifications for BitTorrent.
Kademlia

Kademlia is a DHT that support the usual set and get operations. Kademlia uses 160-bit values to represent keys, and participating nodes have a nodeID in the same 160-bit key space. Key/value pairs are stored at nodes with nodeIDs close 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.

Routing Table  The routing table of a node is organized in so-called $k$-buckets. A $k$-bucket covers an ID range of the key space, and can contain up to $k$ nodes which are ordered from least-recently seen to most-recently seen. If a node is to be added to a bucket, it is first checked if the node is not already in the bucket. If the node is already in the bucket, then it is moved to the end of the list as it has become the most-recently seen node for that bucket.

If the bucket does not yet hold the node and the bucket has less than $k$ entries, then the node is added at the end of the list. If the bucket contains $k$ nodes, the least-recently seen node is contacted to check whether it is still live. If the least-recently seen node fails to respond it is removed from the bucket, and the new node is added to the bucket. If the node does respond, then the new node is not added to the bucket and discarded.

The preference for old live nodes over new nodes 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.

$k$-buckets get filled with nodes as a side effect of lookup and store operations. The distance metric, XOR, is symmetric, i.e., for all nodes $A$ and $B$ the distance from node $A$ to node $B$ is equal to the distance from node $B$ to node $A$. A node receiving messages from other nodes adds these nodes to its routing table, because the node can send message to the nodes.

The routing table constitutes of dynamically allocated $k$-buckets. The routing table starts as a single $k$-bucket, which covers the entire key space. The routing table is expanded as new nodes are added to the routing table in the $k$-bucket for that new node is full. If a $k$-bucket is full and a new node is added, then the bucket is split in two separate buckets with each bucket covering half the range of the original node. However, a bucket is only split if the nodeID of the root node lies in the range of that bucket. See Figure 3.7 for the evolvement of a routing table of node with ID 00..00.

As a consequence, the routing table has a finer granularity for nearby nodes. More specifically, a completely filled routing table stores nodes with distance $d$ in a bucket that covers $2^{\left\lfloor \log_2 d \right\rfloor}$ nodeIDs, i.e., the greatest power of two smaller than or equal to $d$. Thus the greater the distance, the larger is range that bucket covers.
Figure 3.7: The evolvement of a routing table of the node with ID 00..00.
for that distance.

**Operations** The basic two operations that a hash table and thus also a DHT must support are a set and get operation. Kademlia uses sloppy storage, and stores a value on the $k$ nodes closest to the key. A node, that decides to store or lookup a value, first needs to find the $k$ closest nodes, which is an iterative process. During the search, the node keeps a list of the $k$ closest nodes it has found so far, and it is initially filled by the $k$ closest nodes from the routing table. Then the node will perform a number of iterations to find closer nodes. A single iteration consists of selecting $\alpha$ nodes that have not been queried yet from the $k$ closest nodes list. These selected nodes are queried for the $k$ closest nodes they know of. The $k$ closest nodes is updated with new nodes received from the queried nodes. If no new closest node is learned, all other nodes that have not yet been queried are queried for nodes. The iterative process ends when all the $k$ nodes are queried and no new close nodes are learned. The parameter $\alpha$ is the concurrency parameter, and determines how many nodes are requested for closers nodes concurrently.

To store value, first the $k$ closest nodes have to be found using the algorithm described above. Subsequently, a store request is sent to these nodes.

In order to keep values alive, each node republishes the value as needed. Furthermore, to clear stale data from Kademlia, the original publisher has to republish the value on a fixed interval.

Reading a value works similar to storing a value, with the exception that once a value is found, the procedure stops immediately. A node starts by searching close nodes as in the algorithm mentioned above. However, if a queried node has a value for the key it returns the value instead of the $k$ closest nodes it knows.

**Khashmir**

To store BitTorrent uses the infohash of torrent files as key. The values are lists of IP address and port number pairs of peers in the swarm. The store and read operations of Kademlia are replaced by respectively the *AnnouncePeer* and *GetPeers* primitives, though they show similarity to the store and read operations mentioned above.

In Khashmir, values are lists, and each list item is checked separately if it has expired. Normally, a write operation replaces any old value. However, when a peer performs an AnnouncePeer operation it does not overwrite any old values in place. The AnnouncePeer operation looks up the $k$ closest nodes just as in Kademlia, but instead of writing a value the value is added to the list.

The GetPeers operation is similar to the read operation, except that the operation does not end after a value has been found. The search continues until all the $k$ closest nodes have returned their.

A Tribler node can learn from other nodes its own *connectability*, i.e., thus it finds out whether it is firewalled or not. This information is used when a node wants to do a download. If the node is connectable, then it will perform GetPeers as well as
AnnouncePeer. Thus it retrieves contact information of other peers in the download swarm, and it stores itself as participating in the swarm so that others nodes can contact it. If the node is not connectable, then the AnnouncePeer is useless, for others peers cannot set up a connection. Therefore, when a non-connectable node decides to participate in a swarm, it only performs a 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 these weaknesses. 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 for tags. Finally, we present the BuddyCast 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 BuddyCast. 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 BuddyCast 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 component 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 versions of Web 2.0 Interface. 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 our measurements of the performance of the Web 2.0 interfaces 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 of particular search. 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 purged, 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 Browser.

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 including 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 after 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 Web 2.0 interface the user has to wait 1.48 seconds.

The threaded 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 $\frac{148.064}{58.699} \approx 1.79$ 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 $\frac{58.699}{50.883} \approx 1.15$. The limited speedup is caused by the limited available bandwidth. Furthermore, when the work queue of the ThreadedDBSearch gets depleted, one thread is refilling the work queue while other threads may be temporarily. This part of the Web 2.0 Interfaces cannot be speed up by using multiple threads.

The speedup from using multiple sites is limited compared to using multiple threads. Using multiple sites comes with other advantages: smaller delays and more diverse content. 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 smaller 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 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 focuses on news and is not focused to be family friendly than 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.

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 the usage of the 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 these measurements. 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 sub-
mitted. 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, item IDs 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

Each installation of the Web 2.0 Browser that has been used has sent average rating requests to the rating server. The only useful thing to do with a fresh installation is to do a search and download items. When an item is selected, which is also the case when it is downloaded, an average rating request is sent. Thus every installation that has been used in a useful way has sent rating requests. Because, the installation ID is also sent along with the request, the number of unique installation IDs in the rating request dataset is the number of installations that has been used in a useful way. 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 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, because there is a great difference between the most used installation and the second most used installation. 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 between other installation is much less. The number of rating requests decreases more moderately between installations. The number of rating requests of the latter ten installations from this top twenty varies from 81 to 111.

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 Dutch web site, so most of the users are 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
Figure 4.5: The 100 tags with the most rating requests.

<table>
<thead>
<tr>
<th></th>
<th>LiveLeak</th>
<th>Revver</th>
<th>YouTube</th>
</tr>
</thead>
<tbody>
<tr>
<td>humor</td>
<td>sexy</td>
<td>dutch</td>
<td></td>
</tr>
<tr>
<td>crash</td>
<td>hot</td>
<td>kooten</td>
<td></td>
</tr>
<tr>
<td>video</td>
<td>girl</td>
<td>bie</td>
<td></td>
</tr>
<tr>
<td>funny</td>
<td>babe</td>
<td>the</td>
<td></td>
</tr>
<tr>
<td>comm</td>
<td>cute</td>
<td>vieze</td>
<td></td>
</tr>
<tr>
<td>wtf</td>
<td>bikini</td>
<td>vpro</td>
<td></td>
</tr>
<tr>
<td>iraq</td>
<td>girls</td>
<td>kees</td>
<td></td>
</tr>
<tr>
<td>tv</td>
<td>boobs</td>
<td>simplisties</td>
<td></td>
</tr>
<tr>
<td>cam</td>
<td>fetish</td>
<td>nederlands</td>
<td></td>
</tr>
<tr>
<td>anim</td>
<td>tits</td>
<td>retro</td>
<td></td>
</tr>
</tbody>
</table>

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

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 hutch content. LiveLeak and Revver are less popular than YouTube and there is much less hutch content available from these sites.

The content that has been provided by Revver is mainly adult content. 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.
<table>
<thead>
<tr>
<th>Rating</th>
<th>Number</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>33</td>
<td>11.3%</td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td>6.2%</td>
</tr>
<tr>
<td>3</td>
<td>26</td>
<td>8.9%</td>
</tr>
<tr>
<td>4</td>
<td>41</td>
<td>14.1%</td>
</tr>
<tr>
<td>5</td>
<td>173</td>
<td>59.5%</td>
</tr>
</tbody>
</table>

Table 4.3: The distribution of the ratings.

<table>
<thead>
<tr>
<th>Web Site</th>
<th>No. ratings</th>
<th>Average rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>LiveLeak</td>
<td>23</td>
<td>3.91</td>
</tr>
<tr>
<td>Revver</td>
<td>24</td>
<td>2.63</td>
</tr>
<tr>
<td>YouTube</td>
<td>228</td>
<td>4.28</td>
</tr>
<tr>
<td>Flickr</td>
<td>12</td>
<td>3.25</td>
</tr>
<tr>
<td>Zooomr</td>
<td>2</td>
<td>2.00</td>
</tr>
<tr>
<td>Wikipedia</td>
<td>2</td>
<td>2.50</td>
</tr>
<tr>
<td>Total</td>
<td>291</td>
<td>4.04</td>
</tr>
</tbody>
</table>

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

### 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 the ratings over the rating levels.

The ratings are strongly biased towards a positive rating, that is 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, one star is the most issued rating.

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.

### Popular tags

In this section, we present the average ratings for tags. Figure 4.6 shows these tags according to their average rating. To keep the number of tags in the tagcloud limited, Figure 4.6 only shows tags with three or more ratings.

In accordance with Table 4.4, the majority of tags of a high rating (four or five starts). Furthermore, note that there is virtually no intersection of tags shown in Figure 4.6 and the popular tags that we retrieved from the rating requests, shown in Figure tagcloud1.
Figure 4.6: The average rating of all tags with three or more ratings.
4.3 Khashmir

In this section we evaluate the performance of Khashmir as an alternative method for swarm discovery. The usual primary swarm discovery is the tracker, and therefore we will compare the performance of Khashmir with the performance of the tracker.

We have tested Khashmir in two different environments. First, we evaluated Khashmir with large public swarms, and specifically the swarm discovery operation. The primary interests of these tests are the number of peers that is returned by Khashmir and the latency from the start of the operation to the point in time at which the first set of peers is returned by Khashmir.

Second, Khashmir has been evaluated in swarms consisting of a single peer. For these tests, the main primary interests is whether Khashmir actually finds this peer despite of the unreliability of the DHT, stale entries in the routing table, and the sloppy storage used by Khashmir. The performance of Khashmir in this environment is important for our system, because any published item starts as a one peer swarm, and there is no tracker to fall back to. If the performance of Khashmir is not enough, then other users will not be able to download such a published item, and the swarm will never grow to a larger swarm.

4.3.1 Large Public Swarms

In this section we present the measurements on Khashmir for large public swarms. The focus lies on the number of peers returned by Khashmir and the latency. Khashmir returns the found peers in multiple sets dispersed over time. We look primarily at the latency to the first set of peers that is returned, because from this point in time, the Tribler client can start connecting to peers and start downloading the item.

For a tracker, predicting the number of peers and the latency is fairly easy. Trackers have global knowledge of the swarm and are configured to return at most 50 peers. Therefore, trackers will likely return 50 peers in all cases. The latency will likely be comparable to a web server, because a tracker is actually a simple web server. We expect the latency to be within one second.

However, predicting the number of peers and the latency is much more difficult due to the unreliability of UDP, and the unpredictable behavior of other peers participating in Khashmir. However, we expect the latency to be more than the latency of the tracker, because during swarm discovery multiple nodes have to be contacted.

These torrents were selected by taking the three most popular torrents, i.e., the three torrents with the largest swarms, from torrent site The Pirate Bay (thepiratebay.org). The primary swarm discovery for these torrents is through trackers. In addition to the Khashmir measurements, we have also performed measurements the same measurements for the trackers of these torrents. To improve the accuracy, all measurements have been performed three times.

For each swarm, we have also recorded the swarm size according to the tracker
using the scrape extension to get an indication of the swarm size. However, note that this is not the number of peers Khashmir should be able to return under optimal circumstances, because not all peers registered with the tracker are participating in Khashmir. The number of peers registered in Khashmir depends on the actual client software of the users and is likely to be much smaller. Some clients do not use Khashmir or use a different DHT based swarm discovery method, i.e., Azureus. In this section, we first present the measurements on the three large swarms and finally discuss the performance Khashmir in general.

**American Gangster**

The most popular torrent from The Pirate Bay was American.Gangster.DVD.SCREENER.XViD-PUKKA. At the time of these measurements, the scrape extension of the tracker for this torrent reported a total swarm size of 12109 peers consisting of 3621 seeders and 8480 leechers.

Figure 4.7 shows the measurements of Khashmir swarm discovery, and Table 4.5 shows the measurements of tracker swarm discovery.

Not surprisingly, the tracker returns in all three runs 50 peers with an average latency of 0.14s.

In the first run it takes 20 seconds before the first peers in the swarms are returned.

In the other two runs the first peers are returned in respectively 0.5s and 0.76s. As
### Table 4.5: Tracker swarm discovery for torrent American.Gangster.DVD.SCREENER.XViD-PUKKA

<table>
<thead>
<tr>
<th>#peers</th>
<th>latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>50</td>
</tr>
<tr>
<td>2.</td>
<td>50</td>
</tr>
<tr>
<td>3.</td>
<td>50</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>#peers</th>
<th>latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>50</td>
</tr>
<tr>
<td>2.</td>
<td>50</td>
</tr>
<tr>
<td>3.</td>
<td>50</td>
</tr>
</tbody>
</table>

with the latency, there are significant differences in the number of peers returned by different runs. However, the run that returns the fewest number of peers, returns 133 peers which is much more peers than is returned by the tracker. For this torrent, Khashmir is slower than the tracker. However, for two runs it still manages to return the first response within the first second. In terms of number of peers, Khashmir clearly outperforms the tracker.

**I Now Pronounce You Chuck And Larry**

The second most popular torrent from The Pirate Bay was I.Now.Pronounce.You.Chuck.And.Larry[2007]DvDrip[Eng]-aXXo. At the time of these measurements, the scrape extension of the tracker for this torrent reported a total swarm size of 5366 peers consisting of 4131 seeders and 5366 leechers.

Figure 4.8 shows the Khashmir performance measurements for this swarm, and Table 4.6 shows the tracker performance measurements.

The tracker measurements produce expected results, 50 peers in an average of 0.13s. The latency to the first set of peers is equal or less than one second. The number of peers by the runs are respectively 2252, 969, and 646.

**Pirates Of The Carribean - At World’s End**

The third most popular torrent from The Pirate Bay was Pirates.Of.The.Caribbean-At.World’s_End[2007]DvDrip[Eng]-aXXo. At the time of these measurements, the scrape extension of the tracker for this torrent reported a total swarm size of 2280 peers consisting of 987 seeders and 1293 leechers.

Considering the swarm size, we doubt that this torrent is actually the third most popular torrent. The statistics of most popular torrents as presented by the Pirate Bay are likely somewhat stale. However, we are interested in large public swarms...
Figure 4.8: Khashmir swarm discovery for torrent I.Now.Pronounce.You.Chuck.And.Larry[2007]DvDrip[Eng]-aXXo

<table>
<thead>
<tr>
<th>#peers</th>
<th>latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>50</td>
</tr>
<tr>
<td>2.</td>
<td>50</td>
</tr>
<tr>
<td>3.</td>
<td>50</td>
</tr>
</tbody>
</table>

Table 4.7: Tracker swarm discovery for torrent Pirate Bay was Pirates.Of.The.Caribbean-At.World-_.s.End[2007]DvDrip[Eng]-aXXo

and not necessarily the most popular, and therefore this swarm is suitable for our evaluation experiments.

Figure 4.9 shows the measurements of Khashmir swarm discovery, and Table 4.7 shows the measurements of tracker swarm discovery.

The tracker returns 50 peers with an average latency of 0.25s. The latencies of Khashmir runs are respectively 1.14, 0.06, and 0.59.

4.3.2 One Peer Swarms

In this section we test the performance of Khashmir in one peer swarms. The performance in one peer swarms is important because a that is published starts as a one peer swarm. As opposed the tests for large swarms, our main concern is whether other peers are able to find the one peer swarms. The unreliability of UDP,
sloppy storage that used by Khashmir, and the unpredictable behavior of other peers participating in Khashmir may cause peers not to be able to successfully lookup the swarm of a torrent.

To test the performance of Khashmir in one peer swarms, we have created our own swarms using two Khashmir clients. One client announces itself as a peer for a torrent, and the other clients performs a lookup for the same torrent. Both clients have a separate routing table, and have been bootstrapped using router.bittorrent.com.

Table 4.8 shows the results of ten measurements and shows the latency of the lookup.

For all lookups, Khashmir was able to successfully find the registered peer. On average finding this peer took 3.27 seconds. Conclusively, we state that Khashmir performs very well for one peer swarms. If it may occur that a one peer swarm is not found by Khashmir, then this does not mean that the item with that cannot be downloaded, because it is very likely that the swarm is found in the next Khashmir swarm discovery operation.

**Conclusion**

Of the total nine Khashmir runs that have been performed in the large swarms seven runs had the first set of peers available within the first second, and for one
run it takes slightly more than a second. This means that the Tribler client can start connecting peers and begin downloading an item within the second. Considering the buffering time needed by videos that are streamed over the Internet, we believe that these latencies are acceptable for users.

In terms of number of peers, Khashmir also performs good. For all measurements, Khashmir clearly outperforms the tracker.

Khashmir also performs very good for one peer swarms. For all measurements, Khashmir was successful in finding the one peer swarm. Again, we believe that the latencies are very acceptable for users.

In the in the public swarm measurements as well as in the one peer swarm measurements, we encountered a run that had a latency of roughly 20 seconds. These large latencies are caused by nodes that are off-line and requests that result in a time-out. The time-out for Khashmir node response is set to 20 seconds. Additionally, Khashmir has a limit for outstanding requests. If all outstanding requests, are requests to off-line nodes, then Khashmir has to wait for a time-out before it can send requests to other nodes.

Table 4.8: Latency of Khashmir of ten one peer swarms lookups in seconds.

<table>
<thead>
<tr>
<th>latency in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. 0.49</td>
</tr>
<tr>
<td>2. 2.80</td>
</tr>
<tr>
<td>3. 1.22</td>
</tr>
<tr>
<td>4. 1.93</td>
</tr>
<tr>
<td>5. 1.34</td>
</tr>
<tr>
<td>6. 1.25</td>
</tr>
<tr>
<td>7. 3.58</td>
</tr>
<tr>
<td>8. 0.79</td>
</tr>
<tr>
<td>9. 20.44</td>
</tr>
<tr>
<td>10. 0.86</td>
</tr>
</tbody>
</table>

Table 4.8: Latency of Khashmir of ten one peer swarms lookups in seconds.
Chapter 5

Conclusions and Future Work

In this section, we present our conclusions and we provide some at which to aim for future work. We look at whether we have accomplished our goal, which we have defined in Chapter 2. Furthermore, we discuss the viability of the developed system by considering whether it is a suitable replacement for Web 2.0 sharing web sites.

5.1 Conclusions

First, the goal that we stated in Chapter 2 summarized, then we look at whether we have accomplished this goal. Finally, we discuss the viability of Tribler as a sharing application.

5.1.1 Goal

In Chapter 2, we defined our goal as a sharing application that exhibits the following properties:

- HD videos,
- community,
- scalability,
- video-on-demand,
- ease of use, and
- wealth of content.

The first four properties mentioned above are already provided by the Tribler peer-to-peer client software. Our objective is to add “ease of use”, and “wealth of content” to Tribler to turn it into a sharing application for the masses. Concerning the “ease of use”, in particular, the ease of publishing with Tribler had to be improved.
Downloading and viewing items in Tribler is already user friendly, it is just a matter of point-and-click. Currently, there are many sharing web sites with a large aggregated collection of items. We want our users not to miss out on this content and must therefore also be able to access that content. Additionally, we want all content that is retrieved from external sites also to be injected in the Tribler network.

5.1.2 Our Solution

The main aspect that makes publishing difficult is that it requires the publisher to set up a tracker. Most users do not have a server at their disposal that can be used as tracker. Instead of the traditional tracker for swarm discovery, we have chosen to use decentralized tracking and in particular Khashmir. With Khashmir, users are not required to set up a tracker or something similar. A user only have to select the file it wishes to publish, and the rest is taken care of by Tribler. Because of this minimal user action required, we think we have successfully made publishing with Tribler user friendly.

Khashmir performs well in terms of latency and the number of peers that is returned. Except for a few cases, the latency of the Khashmir is less than three seconds. To improve the performance for these exceptional cases, we opt for an improvement in Section 5.2.1. Khashmir works also very well. For large swarms, Khashmir yields many more peers than the traditional, which often have an imposed limit to the number of peers that is returned. Considering these performance measurement results, we also think that Khashmir is a well performing swarm discovery method and suitable for replacing the traditional tracker.

To let users access content from various sites we have defined an interface to access this content. The interface allows the user to execute a keyword search and to retrieve the search results on demand. We have implemented these interfaces for several sites including YouTube, LiveLeak, Flickr, and Wikipedia. Additionally, we have implemented a web interface that aggregates other web interfaces. Thus instead of searching YouTube, it is possible to let a users for videos and YouTube, LiveLeak, and other video sites can be aggregated as if a single site is searched. When an item is downloaded and viewed from these sites, it is published with Tribler using the decentralized tracking method mentioned above.

5.1.3 Viability

As mentioned above, we believe we have succeeded in making Tribler more easy to use and adding access to external web sites. But, is our modified Tribler a viable replacement for Web 2.0 sharing sites? Compared to a Web 2.0 sharing site, Tribler comes short in two aspects. First, the community tools provided in Tribler are very limited to those available on web sites. For example, virtually every web site allows a user to leave comments with each item and discuss the item with other users.
The second aspect is availability. Furthermore, the content of such sharing sites are usually meant for immediate consumption. We think a user is not willing to wait longer than a few seconds before it can view an item. Thus immediate availability is very important for the type of content of sharing sites. The availability of items downloaded with Tribler depends fully on other peers that can and will upload that item. For popular items this will likely not be an issue, but for less popular items this can be problematic.

5.2 Future Work

This section discusses work to be done to further improve Tribler as a sharing application with Web 2.0-like functionality. First, performance improvements are discussed and subsequently functional improvements.

5.2.1 Performance Improvements

In this section, we look at modifications for the decentralized tracking and the Web 2.0 interfaces that improve the performance.

Improved Web 2.0 Interfaces

The Web 2.0 Interfaces retrieve metadata for items and are then passed on to the GUI. However, if an item is not selected in the GUI, then the only metadata that is needed is the title and thumbnail and the retrieval of other metadata is not necessary.

Thus the more detailed metadata is only necessary if the item is selected by the user in the GUI. To reduce communication while retrieving search results is to only retrieve the title and preview and not the more detailed metadata. The title and preview is enough metadata for display in the overview grid. This reduction of communication will translate into a sooner available of search results. The more detailed metadata will then be retrieved on demand when the user selects the item in the overview grid. A disadvantage of this improvement is that the user may experience a small delay between selecting an item and the display of the metadata by the GUI due to the communication required to retrieve the detailed metadata. An alternative to improve the performance of the Web 2.0 Interfaces is to pipeline the retrieval of metadata. All metadata of all items will be downloaded, even if the user does not select the item in the GUI, in contradistinction to the improvement mentioned above. Such a pipeline would have three stages. In the first stage, the title of the item is retrieved; in the second stage, the preview of the item is retrieved; and in the third stage, the detailed metadata is retrieved. As soon as the title of an item is available, the item can be passed on to the GUI, which will show the title. When the preview is available, the GUI will be notified and will then also the preview. Pipelined Web 2.0 Interface is able deliver search results faster.
than a normal Web 2.0 Interface, because items are available even if not all of the metadata has been downloaded.

Ultimately, it is also possible to combine both improvements mentioned above. The retrieval of the title and preview is pipelined, and the detailed metadata is retrieved on demand when the user decides to view this detailed metadata.

Decentralized Tracking Improvements

The Khashmir measurements, described in Chapter 4, exhibit in a few cases in long delays of approximately 20 seconds. The long delays is a result of non-responding nodes, which caused Khashmir to have to wait for a time-out of 20 seconds. From our experience, most nodes reply within a second, thus the time-out can be greatly reduced. Due to a more strict time-out, the possibility of late responses is much larger. To ensure correct and accurate functioning, late responses should not be ignored be should be processes as if they were in time unless the Khashmir operation has already been completed. This is actually how Kademlia works, however Khashmir does not implement this behavior.

As a result, Khashmir will spend much less time waiting on responses, and the accuracy of the Khashmir will be equal. And in the few cases, where the response of a node is late the actual number of outstanding might be slightly higher than the limit set by Khashmir. However, we do not think that incidentally exceeding this limit has a significant negative influence on the performance.

5.2.2 Web 2.0 Functionality

As mentioned in Section 5.1.3, the community tools provided by Tribler are limited compared to what is provided by sharing sites. Especially, we think the following three community tools are important: comments, directed recommendations, and messages. By comments, we mean the possibility for a user to leave a comment with an item that other users are able to view. From our own experience, this is one of the most community functionality used by users.

Another functionality that many sites provides is to send messages to other users. Messages are collected in the inbox of the recipient, which is comparable to an email inbox. In addition to the community tools the collection of 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 responses. A video response is a video that is created as a reaction on another video. The concept of related items is also present in text items. Texts available from Wikipedia, contain many links to other Wikipedia texts which are usually on related subjects.
5.2.3 Hybrid Downloads

BitTorrent works very well if the number of peers is high. Thus BitTorrent performs good if the item is popular, however, when an item is not so popular (any more) the performance is less good. Consequently, a user may encounter difficulties when downloading a not so popular item.

If such an item is originally retrieved from a web site, the item is very likely still available from that web site. The download of a not so popular item can be aided by using the original source of the item. Thus a peer retrieve data from other peers and a HTTP source and combines the received data into a single item. This concept is not new and known as web seeding, and applying it in this case can improve the performance for items with small swarms.
Bibliography


