TriblerShare: A Scalable P2P-Based Web 2.0 Platform

Fabian van der Werf

TriblerShare: A Scalable P2P-Based 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

24th February 2008

Author
Fabian van der Werf
Title
TriblerShare: A Scalable P2P-Based Web 2.0 Platform
MSc presentation
7th March 2008

Graduation Committee

prof. dr. ir. H. J. Sips (chair)	Delft University of Technology
dr. ir. J. A. Pouwelse	Delft University of Technology
dr. ir. D. H. J. Epema	Delft University of Technology
dr. ir. F. A. Kuipers	Delft University of Technology

Abstract

Over the last few years, there has been a rise on the Internet of so-called *social* web services. In contrast with traditional web services where users only consume information, social web services enable users to interact with each other. This new approach to web services is often referred to as *Web 2.0*. Socialized web services have become very popular as shown by examples like YouTube, Flickr, and Wikipedia. However, these sites operate in a centralized way, and the drawback of their popularity is the increased operating costs. At the same time, Peer-to-Peer (P2P) technology has gained much popularity in the area of content distribution because of its lack of central components, which causes P2P systems to scale well. Therefore, P2P technology may be a solution for the poor scalability of current Web 2.0 services.

This thesis describes the research we have conducted in providing Web 2.0 services with scalable P2P technology. We have extended Tribler, an existing P2P client, such that it enables each user to easily share his videos, photos, etc., with other users. Furthermore, we have developed a flexible system that provides access to the large collections of content items available from current popular Web 2.0 web sites. This system is easily extensible, and adding support for a new web site requires only to define structure of the site. Using these interfaces, we also decentralize current Web 2.0 web sites by distributing retrieved items in the Tribler network.

Preface

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

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

Fabian van der Werf

Delft, The Netherlands
24th February 2008

Contents

Preface v

1 Introduction 1
1.1 Web 2.0 2
1.2 Peer-to-Peer Systems 4
1.2.1 Peer-to-Peer Technology 5
1.2.2 BitTorrent 5
1.2.3 Tribler 7
1.3 Outline 8

2 Problem Description 9
2.1 Scalability and Robustness 9
2.2 The Aim of this Thesis 11
2.2.1 YouTube 11
2.2.2 Tribler 12
2.2.3 Our Vision 13
2.3 Related Work 14
2.3.1 Vuze 14
2.3.2 Joost 16
2.3.3 Babelgum 16

3 The Design and Implementation of TRIBLERSHARE 19
3.1 Software Development Process and Functionality 19
3.1.1 Single-Threaded Prototype 20
3.1.2 Multithreaded, Multi-Site Prototype 21
3.1.3 P2P Sharing and Video Browsing 22
3.2 WEB2.0LEECHER 22
3.2.1 Architecture 22
3.2.2 GUI Design 25
3.2.3 Web 2.0 Interfaces 27
3.2.4 Ratings 30
3.2.5 Download Manager 30
3.3 TRIBLERSHARE 31

3.3.1 WEB2.0LEECHER-TRIBLERSHARE Integration 31
3.3.2 Decentralized Tracking 33

4 Experiments and Evaluation 35
4.1 Web 2.0 Interfaces 35
4.2 Real world usage 38
4.2.1 Datasets 39
4.2.2 Average Rating Requests 40
4.2.3 Ratings 43
4.3 Khashmir Problems 44
4.3.1 Khashmir Behavior 44
4.3.2 Khashmir Timeout 46
4.4 End-to-End Video 49

5 Conclusions and Future Work 51

Chapter 1

Introduction

Traditionally, the World Wide Web (WWW) was a collection of a static web pages that could be accessed with an Internet browser. Since the last decade, increasingly more web services are aimed at user interaction and collaboration, and are said to have *social* features. With these social features, users are no longer only consumers of information, but they are also participants in so-called *online communities* in which two-way communication is possible. This new kind of web services with social functionality is referred to as *Web 2.0* [17]. Unlike the term implies, Web 2.0 is not a standard or a technology and so, it lacks a specification. Accordingly, there is no clear boundary between Web 2.0 applications and non-Web 2.0 applications, but there is a gray area of web services that are Web 2.0 to a smaller or larger extent.

Web 2.0 applications have gained a significant share in the 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 web sites, but it gives a good indication of the significance and success of Web 2.0 applications. Six out of ten sites listed in the top 10 are Web 2.0 sites; these are Google, YouTube, MySpace, Orkut, Wikipedia, and QQ.

Most Web 2.0 applications are served by a single server or several computer centers, and consequently, they lack scalability and robustness. Peer-to-Peer (P2P) systems have become popular because of their scalability and robustness. P2P systems exhibit these properties because they are not driven by central servers but by their users.

The aim of this thesis is to create a robust and scalable Web 2.0 application by applying P2P technology that enables users to share videos, photos, etc., and to navigate easily through the available content. Current existing Web 2.0 services like YouTube and Flickr have huge collections of content items, and our integrated solution enables users to access these collections.

This chapter provides an introduction to Web 2.0 and P2P technology. Section 1.1 gives a further explanation of what Web 2.0 encompasses, Section 1.2 explains general P2P technology and discusses in more detail the P2P systems that are relevant

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

Table 1.1: The global top10 ranking sites assessed by Alexa on July 5$^{\text{th}}$, 2007 (source: alexa.org).

to this thesis, and Section 1.3 provides the outline of this thesis.

1.1 Web 2.0

As mentioned above, there is not specification of Web 2.0, and there is still much debate to what Web 2.0 exactly is. The term Web 2.0 was first used by O'Reilly Media. Figure 1.1 shows a meme map that was developed during a brainstorm session at O'Reilly Media. To further explain Web 2.0, the six core competencies mentioned in the meme map are discussed in this section. These six competencies do not form a checklist to check whether an application is Web 2.0 or not, but are characteristics of Web 2.0 applications.

Services, Not Packaged Software Traditional packaged software and Web 2.0 applications differ in the way the functionality is brought to the end user. Traditional packaged software runs on the host of the user, while Web 2.0 applications run on the servers of the service provider and are accessed with an Internet browser. This difference causes that Web 2.0 services can be used with virtually any platform without have to port the services for each platform. Furthermore, services can be upgraded without any action from the user. Therefore, services have no fixed release schedules but instead follow the "release early, release often" principle.

Architecture of Participation The WWW was mostly used for one-way traffic of information, with users only consuming but not producing information. Web 2.0 applications let users participate and add value to the Web 2.0 application for other users, for example, a user sharing his videos with others.

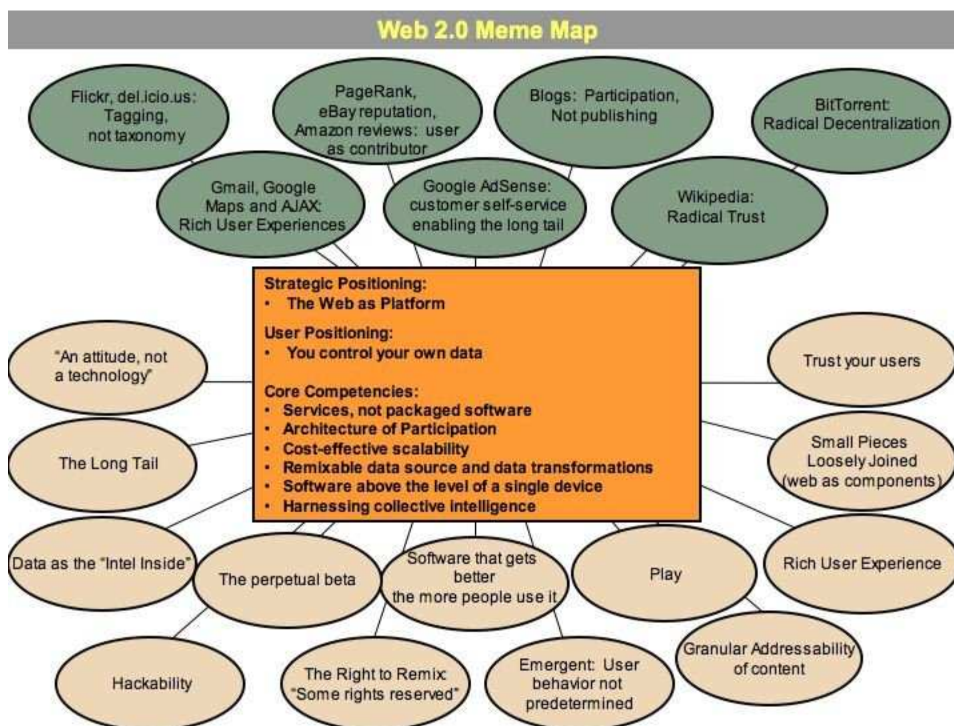

Figure 1.1: Web 2.0 meme map by O'Reilly Media.

Cost-effective Scaling Most Web 2.0 applications are still not scalable in a cost-effective way. These applications are being served by a client-server architecture just as the world's first web page. This non-scaling architecture causes sites to have to degrade the quality of their content items in order to limit the necessary bandwidth. There exist a few well-scaling applications which are mostly P2P file sharing systems, such as BitTorrent [6]. However, most of these file sharing applications only harness the bandwidth of the users and have a total lack of social interaction.

Remixable Data Source and Data Transformations Remixable data source and data transformations means that data and services provided by a Web 2.0 application should be easy to integrate with other applications. In practice, this results in Web 2.0 applications having a documented API through which their data and services are exposed. A good example is the usage of Yahoo! Maps by Flickr to indicate the location of pictures on a map.

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 restricted to a single PC. An example of a Web 2.0 counterpart is Google Docs & Spreadsheets, which stores documents online that are accessible and editable with any PC that is equipped with an Internet browser. Google Docs & Spreadsheets also enables collaboration through document sharing.

Harnessing Collective Intelligence The classic example of a web application that harnesses collective intelligence is Wikipedia, which is an online encyclopedia. The articles of Wikipedia are written by volunteers from around the world, and can be written by anyone with Internet access. The accuracy of Wikipedia may not be as high as the accuracy of an encyclopedia that is composed by experts, but in many cases its accuracy is good enough [10].

1.2 Peer-to-Peer Systems

Peer-to-Peer systems enable users to share resources such as information and bandwidth. Peer-to-Peer systems are usually associated with file sharing, but have also been in applied to other domains, such as telephony [20] and Video-on-Demand [16, 12]. This section gives an overview of the Peer-to-Peer systems that are relevant to this thesis. Section 1.2.1 discusses general Peer-to-Peer technology and its advantages, Section 1.2.2 discusses BitTorrent, and Section 1.2.3 provides an overview of Tribler.

1.2.1 Peer-to-Peer Technology

P2P systems have two defining characteristics: first, exchange of resources between peers occurs directly, instead of requiring an intermediate centralized server, and second, the ability to handle instability and failures [5, 15]. In pure P2P systems all nodes are functionally equivalent and have equal tasks, but similar systems that employ centralized components for non-critical tasks, such as bootstrapping, are generally also considered to be P2P systems.

P2P technology is already applied in many areas, but it is most notable for its contribution in the area of content distribution. In content distribution, the core operations such as publishing, searching, and retrieval are provided by the distributed object location and routing algorithms.

Compared to centralized systems, P2P systems have two major advantages: scalability and robustness. P2P systems without any central components scale automatically as the user base grows, because each participating node contributes resources to the system. Hence, as a new node enters the system, the total capacity of the system increases. A P2P system will always have enough resources as long as each user contributes at least as much as it consumes.

P2P systems are much more robust than centralized systems because of the lack of central components. Critical centralized components are single points of failure, and if one of these components fail, the entire system is no longer accessible by anyone. In P2P systems the main components are peers, and if a peer fails then only the resources contributed by that peer become unavailable.

Besides scalability and robustness, some systems may provide features such as fairness, integrity, privacy, and persistence.

P2P networks have gained much popularity, and preliminary results of research show that a vast amount of Internet communication originates from P2P applications [13]. Measurements conducted in 2007 indicate that 50% to 90% of all Internet communication is P2P communication, and BitTorrent is responsible for 50% to 75% of all P2P communication.

1.2.2 BitTorrent

BitTorrent [6] is a popular P2P file sharing system designed and implemented by Bram Cohen. The BitTorrent system consists of peers and trackers. A shared file in BitTorrent is often called a *torrent*, and the set of peers that are sharing the file is called the swarm of that torrent. In each swarm there are two types of peers, leechers and seeders. Leechers are peers that want to have a copy of the file and may already have partially downloaded the file. Seeders are peers who already have a complete copy of the file and are sharing it with leechers in order to help the distribution of the file. Each torrent is associated with at least one tracker which serves a meeting point for peers.

When a peer decides to distribute a file with BitTorrent, it has to create a so-called *.torrent file*. This .torrent file holds metadata and provides enough information

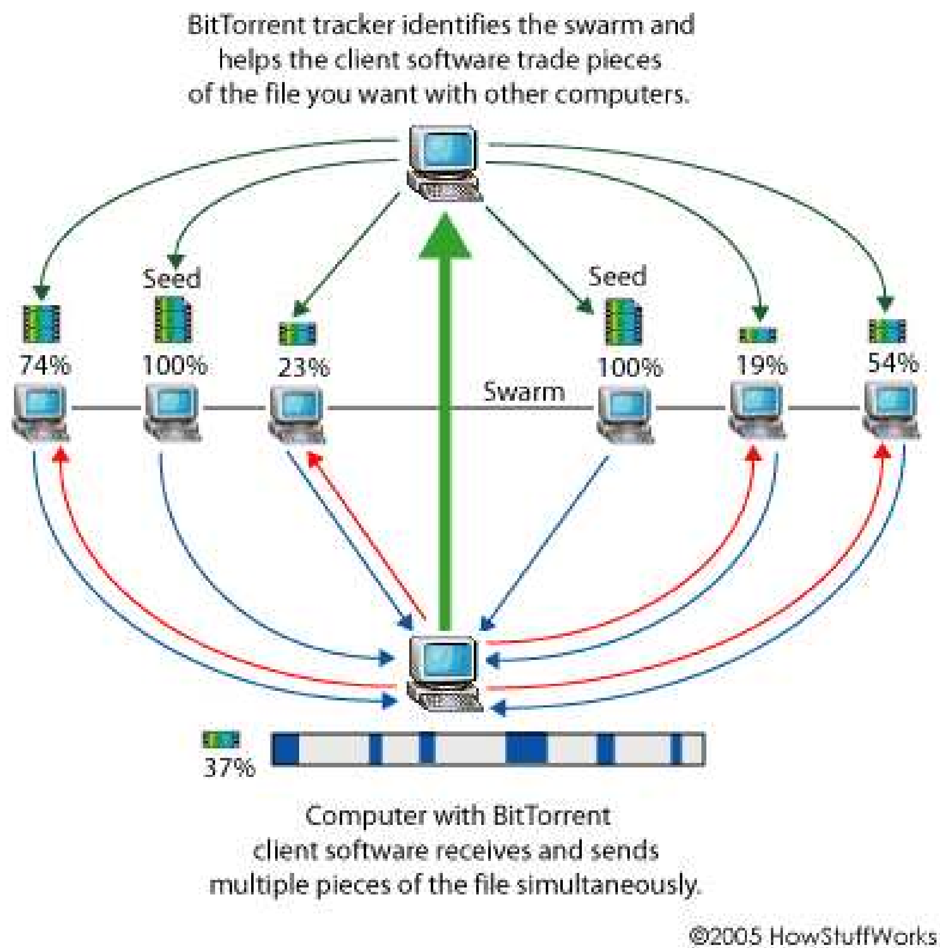

Figure 1.2: A simplified BitTorrent network (source: http://www.kevinwolf.com).

for other BitTorrent peers to download the published file, like filename, length, and tracker URL. 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. Initially, the publisher is the only peer with a copy of the file, and should therefore register itself with the tracker as a seeder.

To download a file, a peer contacts the tracker that is stored in the .torrent file and request swarm information. Using the swarm information, the peer contacts 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 shows a small and simplified BitTorrent network.

Leechers exchange data in a tit-for-tat manner. So for a peer to attain a higher

download rate, it must increase its upload rate. Most peers are likely to be subject to bandwidth limitations, so it is possible that peers do not respond to increased download rates. Therefore, during the entire download process, peers continue to search for other peers that can provide a higher download rate. The behavior of a peer that is only downloading and not uploading is called *free-riding* [8] and is a common problem in P2P systems. The tit-for-tat data exchange method is BitTorrent's approach to free-riding. For seeders to exchange data in this way is senseless because they already possess a copy of the file and thus no do need to download any data.
In addition to the tracker, BitTorrent can also use a *Distributed Hash Table* (DHT) for peer discovery. The DHT spreads the task of storing swarm information across all peers and each peer becomes a potential tracker. There is only a single global DHT that stores the swarm information of all torrents. Thus a single DHT replaces all trackers. However, currently the DHT is mostly used as secondary peer discovery method in addition to trackers.
The BitTorrent system does not address the dissemination of .torrent files. An often used solution is to simply deploy a web server that hosts the .torrent files, e.g., Mininova (`mininova.org`). Additionally, many of these sites generate RSS feeds of new torrents in a certain category or that match a certain search query.

1.2.3 Tribler

Tribler [18] is a social-based P2P file sharing system based on BitTorrent. Tribler is a joint research effort by The Delft University of Technology and Vrije Universiteit Amsterdam and is part of the I-Share project, which conducts research in the 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.
A user in Tribler is more than an IP address and port number as in most BitTorrent clients. Users have nicknames and small pictures as visual representation of themselves. Users are connected with each other by an overlay network and are continuously exchanging information such as .torrent files and information about other users. The exchange of .torrent files removes the need for users to have to search the web for .torrent files, because they are provided by other Tribler clients. A Tribler client tries to find other Tribler users with a similar taste, which are used by Tribler to provide recommendations for other files. So-called *taste buddies* are found by comparing the set of completed downloads. If the similarity for two users is high enough, then these users are considered to be taste buddies. Every file that a user has downloaded is regarded by Tribler as a recommendation for that file to his taste buddies. Thus Tribler gives the user recommendations aimed at his taste, and the more downloads a person has completed, the better will the recommendations be.
A user can add another user as a friend, which can help improve download rates.

When helping a friend, data is no longer exchanged in a tit-for-tat manner as usual, but data is uploaded as fast as possible. It is required that the friendship is mutual, so both users have to add each other as a friend.
Besides social features, Tribler provides P2P video-on-demand. Tribler has adapted BitTorrent such that it is possible to download files in order, which enables to stream media files. When downloading a movie, users can start watching the movie before the download process is completed.

1.3 Outline

The remainder of this thesis is organized as follows. In Chapter 2, we present the problem of the poor scalability of Web 2.0 applications. First, we explain the cause of this problem and the its significance by discussing existing applications. In this chapter, we also discuss the functionality offered by existing Web 2.0 applications and we identify what functionality Tribler is lacking in order to be used as a Web 2.0 sharing application. In Chapter 3, the architecture and design of our solution is presented. First, WEB2.0LEECHER is discussed, which is a system that provides an alternative way to navigating through videos, photos, and articles from multiple web sites. Then we discuss our Tribler-based system, TRIBLERSHARE, that enables users to share content items. In Chapter 4, we evaluate the implementation of TRIBLERSHARE by means of experiments. Performance measurements as well as community measurements of WEB2.0LEECHER are presented. Finally, in Chapter 5 we give our conclusions and future work.

Chapter 2

Problem Description

Web 2.0 applications are usually delivered through an Internet browser by a single server or several computer centers. In both cases, the performance of these applications decreases when the number of users increases because the resources of the servers have to be share by all users. P2P networks, on the contrary, perform better as the user base grows. In this chapter we present the problem statement of Web 2.0 applications with poor scalability.
We start this chapter by examining the problem of poor scalable Web 2.0 applications and the magnitude of this problem in Section 2.1. Our goal is to create a scalable Web 2.0 application that offers Web 2.0 functionality and is hosted with P2P technology in order to provide scalability. Before looking for a solution to this problem, it is necessary to formulate the criteria which a solution must satisfy. Therefore, in Section 2.2, we first analyze the functionality offered by Web 2.0 applications and P2P networks, and we define the criteria of a P2P-based Web 2.0 application as best of both worlds. In Section 2.3, we describe current solutions to the problem and indicate why these attempts are not satisfying, thereby justifying the need for a new scalable Web 2.0 system.

2.1 Scalability and Robustness

The content of most Web 2.0 applications is brought to the user through web browsers and is served by collections of powerful web servers. This traditional approach to web services has two major drawbacks, poor scalability and poor robustness.
The client/server architecture scales poorly because the web site is driven by a collection of dedicated servers. The poor scalability is caused by the limited bandwidth, computational power, and storage that is available. The amount of these resources remains constant while the number of users increases. So the resources allocated per user decreases and with it the per user performance. This problem is usually overcome by expanding the collection of servers.
The client/server model provides no robustness because a web server is a single

point of failure. Once a web server fails, content hosted by that server is no longer available to anyone. To improve robustness, large web sites are often 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 client/server architecture. Therefore, complete site outages can still occur. Moreover, server failures may even lead to loss of vast amounts of data.
So to partially improve robustness and deal with poor scalability, web sites have to expand their server farm so that enough resources are available to handle client requests. However, expanding server farms comes with costs, and so this raises the question to what extent will organizations be able to continue to expand server farms before the operating costs of the service become too high for it to be profitable.

WikiMedia

WikiMedia Foundation is a good example of a web service provider that is suffering from increasing operating costs due to poor scalability. 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 [21]. One of the most well-known projects of WikiMedia is Wikipedia (`Wikipedia.org`), which is a free online encyclopedia that is editable by the public. Wikipedia is in the top 10 of the globally most visited sites (see Table 1.1).
The total expenses for WikiMedia have increased tremendously over the last few years. For the years 2004 to 2006, the total expenses were respectively $ 23,463, $ 177,670, and $ 791,907 [22]. WikiMedia is a foundation, and therefore, these expenses have to be covered by donations and gifts. Considering the growing popularity of Wikipedia, these costs will continue to increase. Whether WikiMedia will in the future be able to cover the costs with just donations and gifts is questionable.

YouTube

YouTube is also a good example of a web service with huge operating costs. YouTube is a video sharing web site that allows users to share their videos with the rest of the world. Because video is one of the media types that requires the most storage space, YouTube has very high operational costs.
In April 2006, the total amount of bandwidth usage by YouTube was estimated at 200 terabytes per day [9, 11], and every day, over 100 million videos requests 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 an immense amount of storage space, for their video collection has been estimated at 45 terabytes. The operating costs for YouTube are immense, and as consequence, there is much speculation on YouTube collapsing under its own weight.

2.2 The Aim of this Thesis

The aim of this thesis is to provide the functionality offered by Web 2.0 sites with the scalability and robustness of P2P networks. Non-Web 2.0 services suffer from the same scalability and robustness issues as Web 2.0 applications, because they both have the same client/architecture. However, we restrict ourselves to Web 2.0 sharing sites, because we think that non-Web 2.0 sites are not suitable for P2P systems. In both, Web 2.0 sharing sites and P2P file distribution systems, the available content is determined by its users. Because of the radical decentralization of P2P systems the available content cannot be controlled by a single authority or is difficult implement, which is usually necessary for non-Web 2.0 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, it is our intent 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 P2P networks.
As a model Web 2.0 site, we use the popular video site YouTube. YouTube is one of the most popular Web 2.0 sharing site, and we therefore assume YouTube has the right approach to Web 2.0 sharing. In addition, most Web 2.0 sharing sites do not differ much in the functionality they provide. Tribler already has a number of social features and is, therefore, a suitable basis for our Web 2.0 sharing application.
In this section we analyze the functionality offered by YouTube and Tribler. Next, we define the desired features for our Web 2.0 platform as best of both worlds.

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, the most viewed videos, the most discussed videos, and the 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 keyword search. With each video, buttons are available to rate the video, to add the video to favorites, and to share the video with others. All this is presented in a "point-and-click" interface.
Publishing a video is also very easy. First, a user only enters some metadata and selects the video file from the local hard drive to upload. Alternatively, a user can also choose to create a video directly via his webcam. The collection of videos uploaded by a single user is considered to be a channel to which fellow users can subscribe.

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

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

Wealth of Content The number of videos hosted by YouTube is immense, and the total amount of video data is estimated at 45 terabytes [11]. This immense number of videos contributes to the popularity and success of YouTube, because a wide collection attracts many users.

2.2.2 Tribler

In this section, we analyze the functionality of Tribler.

Decentralization And Scalability Tribler is a decentralized system, and does not have the scalability issues from which centralized architectures suffer. This is the most important quality of Tribler, because it solves our initial problem, the poor scalability of Web 2.0 applications. A fully decentralized system does not have any maintenance costs. There are no central components that need to be maintained, and each user maintains his 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. In contrast. with a centralized architecture, the content can be managed by the supplier of the web service.

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

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

	Web 2.0	Tribler	Our Vision
Ease of Use	×		×
Scalability		×	×
High-Definition Video		×	×
Video-on-Demand	×	×	×
Community	×	×	×
Wealth of Content	×		×

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

them. For multi-file torrents, the *play ASAP* mode lets the user pick a file from the torrent which will be downloaded using this policy.

Community The peer review functionality of Tribler constitutes of popularity and recommendations. Tribler determines the popularity of a download by looking at the number of peers in the swarm of the download. The more peers in a swarm 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 consider the functionality offered by Web 2.0 applications and Tribler to formulate the criteria for a scalable Web 2.0 sharing application. Table 2.1 summarizes the functionality of Web 2.0 applications and Tribler, and it defines our vision of a scalable Web 2.0 platform as the best of both worlds. For our Web 2.0 application, we will use Tribler as basis, which we will extend in order to provide Web 2.0 functionality.
Tribler has an easy to use graphical user interface in order to download. But it does not enable a user to easily publish a content item. To publish an item, the user needs to take the same steps as publishing with BitTorrent. A publisher has to set up a tracker and create a .torrent file which is then disseminated with Tribler to other users. Ideally, a user does not require knowledge about trackers and .torrent files, and only has to select which files to publish with the graphical user interface. Because our sharing application will be based on Tribler, it will benefit from Tribler's scalability. This scalability provides the capacity to provide HD videos. 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 have a resolution of

1920×1080. So, a full HD quality videos contains 26 times more information than YouTube videos. Offering HD videos would increase the required resources approximately with a factor 27. Considering that the costs of YouTube are already immense, HD would make it infeasible. Furthermore, Tribler is able to deliver Video-on-Demand, which is important for the usability of Tribler. However, Video-on-Demand is only possible if the rate at which other peers are uploading is sufficient, whereas the Video-on-Demand of YouTube works always for every video.
Tribler as well as YouTube provides means to let users communicate with each other. However, with YouTube, user communication occurs explicitly. YouTube users, can favorite videos, leave comments, and send messages. On the contrary, the user communication in Tribler occurs without any user intervention. Content items downloaded and kept in the Tribler Library are considered as a vote for those items. Tribler clients exchanges these votes in order to provide the user recommendations.
To leverage existing content, our platform must integrate with popular Web 2.0 applications. Once an item is retrieved from an external Web 2.0 site, further distribution of that item benefits from the scalability of our application. To increase usability, our platform integrates with these sites seamlessly, and users should not be able to tell whether an item comes from an external site or from the Tribler network. Although our main focus lies been on video, we do not restrict to this media type. Photos from Flickr and texts from Wikipedia are must also be supported.

2.3 Related Work

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

2.3.1 Vuze

Azureus is one of the most popular BitTorrent clients. Version 3 of the Azureus is named Vuze [4], and it is a platform for free publishing video, audio, and games. Users browse the content through a stunning graphical user interface (see Figure 2.1). Content is divided into several categories. In addition to the categories, Vuze offers a few fixed channels, such as *HD Trailers*, *BBC*, and *Anime!* Users can search for content using keyword search and by making a selection using criteria such as popularity, duration, price, and date of publication of the content.
Downloading of items occurs through the BitTorrent protocol, and every item in Vuze is a torrent download. The downloaded copy of the file is identical to the original version that was uploaded. Consequently, Vuze also supports High-Definition content unlike YouTube which rescales every video to a resolution of 320×240

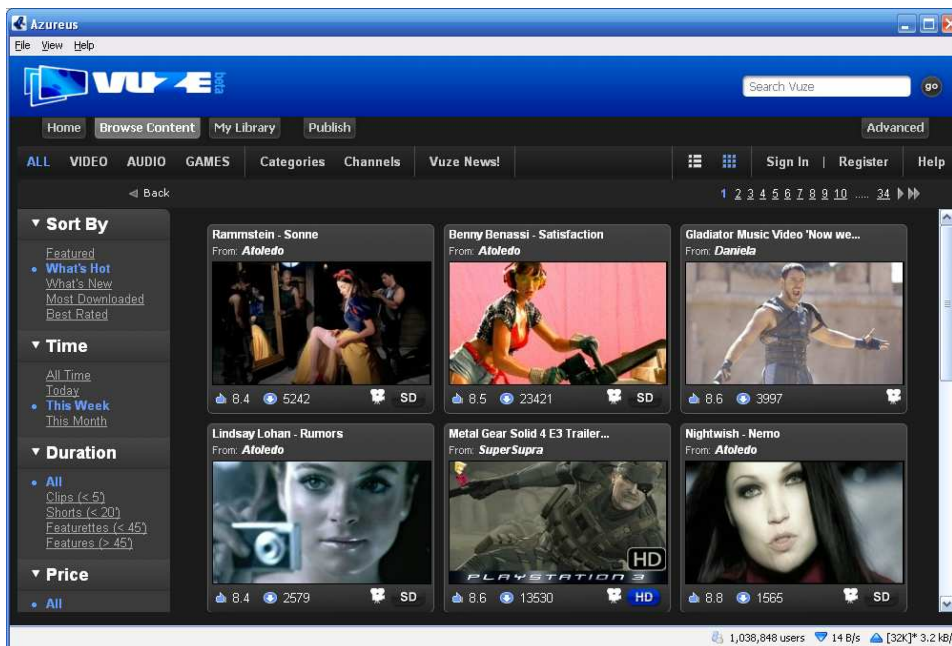

Figure 2.1: Vuze showing an overview of the hottest videos.

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 trackers by means of a DHT, the primary method for swarm discovery is by means of trackers. Vuze has a few centralized trackers which have to be expanded as the user base grows.

The second central component are the indexing servers. As discussed in Section 1.2.2, BitTorrent does not provide a solution for the dissemination of the .torrent-files. Vuze simply uses central indexing servers to retrieve items with or without specific properties. Like trackers, the indexing server capacity has to be expanded as the user base grows.

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

2.3.2 Joost

Joost [2] is a P2P TV application created by the founders of Skype [3]. A user can view the channels in his 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 that user can watch whenever he desires. 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 these 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 expose and document the API for creating new widgets to allow external developers to write plug-ins. The widget system then becomes a flexible plug-in system which enables users to add extra functionality to Joost.
Joost is not a free publishing platform like Vuze is. Publishing is only limited to parties that have a business deal with Joost.

2.3.3 Babelgum

Compared to Vuze and Joost, the functionality offered by Babelgum [1] (see Figure 2.3) is somewhat simple. Babelgum features a number of channels, currently nine, and each channel serves multiple programs. Users can view programs on demand. A unique feature that distinguishes Babelgum from Joost and Vuze is the ability for users to create so-called *smart channels*. A smart channel is created by entering a few tags, and the smart channel will be composed of programs that are relevant to the entered tags. Like Vuze and Joost, Babelgum offers users to rate programs. Furthermore, Babelgum allows users to report inappropriate content.
Babelgum does not satisfy our criteria, because it does not provide High-Definition video quality and publishing content is not straightforward. The video quality provided by Babelgum is mediocre and far from High-Definition quality.
Babelgum is not a free publishing application. Publishing is reserved for professional and semi-professional content owners.

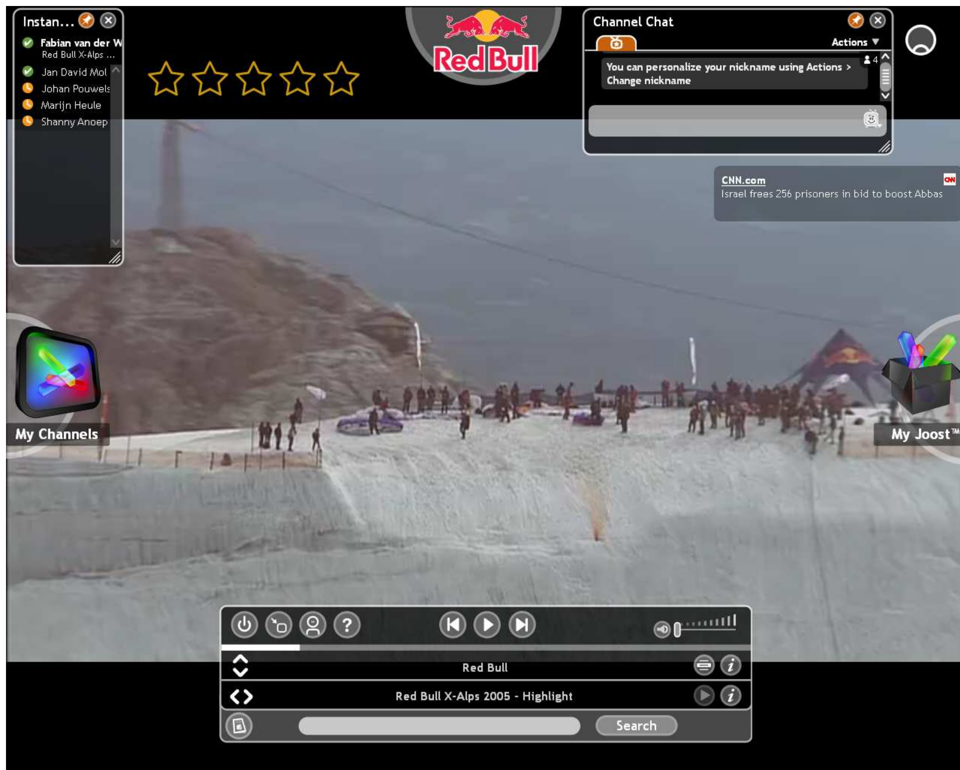

Figure 2.2: A screenshot of Joost displaying a program from the Red Bull channel with widgets for instant messaging, ratings, and RSS feeds.

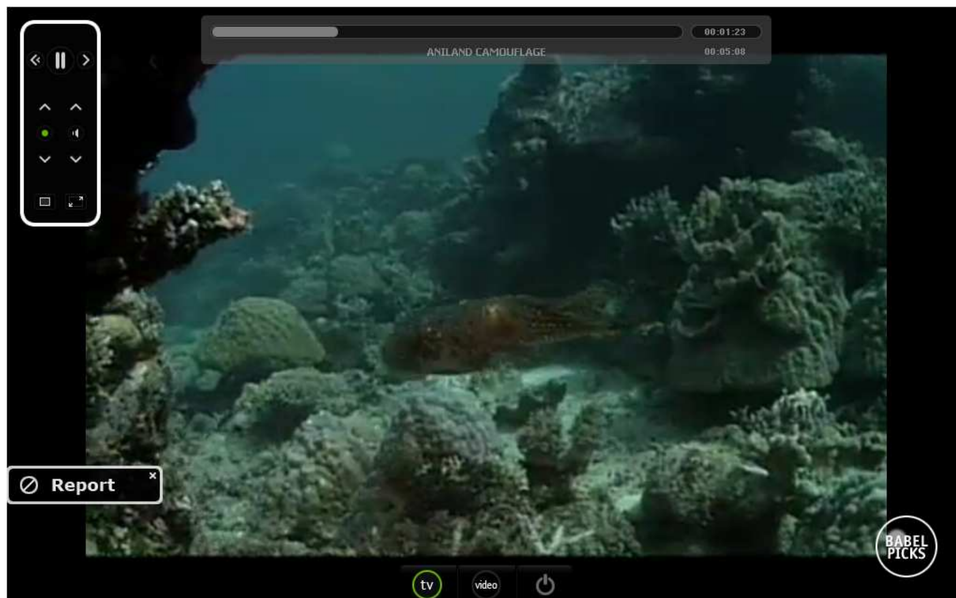

Figure 2.3: A screenshot of Babelgum playing a TV program.

Chapter 3

The Design and Implementation of TRIBLERSHARE

This chapter presents the design of a scalable Web 2.0 platform which is not hampered by scalability issues. Our Web 2.0 platform is built upon Tribler and extends it with Web 2.0 functionality. To satisfy our criteria as defined in the previous chapter, Tribler needs to be extended in two ways (see Table 2.1). First, the ease of use of Tribler must be improved, and in particular, the ease of publishing. Second, Tribler must integrate seamlessly with existing Web 2.0 sites, so users can access the content of these sites.

We have developed WEB2.0LEECHER, which is a stand-alone application that interfaces multiple Web 2.0 sites. These interfaces enables users to access content items from multiple web sites in a uniform fashion. Subsequently, the functionality of WEB2.0LEECHER has been partially integrated with Tribler, and Tribler has been extended with easy publishing capabilities. We name our version of Tribler TRIBLERSHARE.

In this chapter, we first discuss the functionality of WEB2.0LEECHER and TRIBLERSHARE in Section 3.1. Section 3.2 describes the architecture and design of WEB2.0LEECHER. In Section 3.3, we discuss the integration of WEB2.0LEECHER with TRIBLERSHARE and the publishing capabilities of TRIBLERSHARE.

3.1 Software Development Process and Functionality

Due to the experimental nature of the development of a proof-of-concept, we have taken an incremental approach to the implementation. The implementation has been evaluated at regular intervals to identify strong and weak points of the implementation in order to improve the proof-of-concept. First, we have developed a stand-alone application that is able interface multiple Web 2.0 sites. As a consequence of our incremental approach, the development and functionality of the prototype can be divided into three stages:

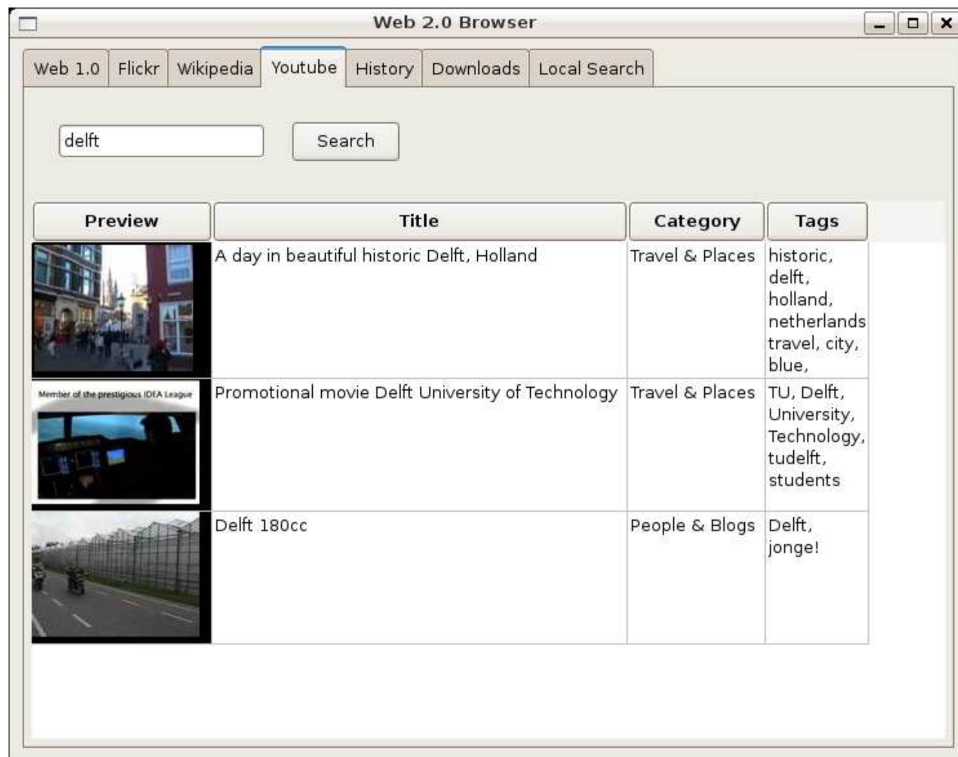

Figure 3.1: A screenshot of the first prototype performing a keyword search on YouTube videos.

1. Single-Threaded Prototype
2. Multithreaded, Multi-Site Prototype
3. Integration of Video Browsing with P2P

We will now describe these steps in turn.

3.1.1 Single-Threaded Prototype

In the first stage, we have developed a first version of WEB2.0LEECHER which only uses a single thread for interfacing external Web 2.0 sites. Due to the combination of synchronous I/O and a single thread, the performance of WEB2.0-LEECHER is not very well. Result items of a search become available roughly every 1.5 second for each item. In Section 4.1 we will present more response time measurements of search operations.
Figure 3.1 shows a screenshot of WEB2.0LEECHER in the first stage. By means of a keyword search, users can search the supported web sites for content. When a search is executed, WEB2.0LEECHER 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. In order to demonstrate the generality with respect to the type of media, video, image, and text are supported.

3.1.2 Multithreaded, Multi-Site Prototype

The two main problems that we identified in the first prototype are the bad performance of search operations and the limited number of supported web sites. The bad performance of search operations is because only a single thread is used to retrieve data from the web sites. In the second stage, we have created a flexible web interface system that enables to easily create multithreaded web interfaces which improves the performance. Furthermore, this web interface system enables a programmer to add support for a web site in a matter of hours, and multiple web sites can be aggregated such that they appear as a single site with a single search operation that searches all these web sites.
Figures 3.3 and 3.4 show screenshots of the Web 2.0 Browser. WEB2.0LEECHER in this stage was released to the public on April 10th, 2007 as "Tribler Web 2.0 Browser". In Section 4.2, we present measurements of public usage of WEB2.0LEECHER.
The changes and improvements over the first stage are the following:

Multiple Sites In the first stage, a user selected explicitly on which site the search operation had to be performed. In order to make WEB2.0LEECHER more generic, it can aggregate multiple sites which will then appear as a single collection of content items. This enables the user to only select the media type it wants to search for, videos, photos, or text. WEB2.0LEECHERcontacts the appropriate web sites according to the selected media type and combines the search results of all sites into a single search results . For example, if the user searches for videos then YouTube, LiveLeak, and Revver are used to find videos matching the search query. The content items retrieved from the various sites are transformed into a uniform representation, and are presented as a single search result set. So a user cannot distinguish items from different sites, and it appears to the user as if he is searching a single large collection of videos.

Improved Search Performance Search operations are multithreaded with up to four threads per web site. The response times of search operation have improved vastly, see Section 4.1 for more details. The user experience of the search operations has additionally been improved through *precaching*. While the user watches a page of the search results, the next page of the search results is being loaded.

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

Content Item Rating 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, WEB2.0LEECHER is equipped with its own rating system. WEB2.0LEECHER provides a cross Web 2.0 site rating system; a single rating system is used for all sites.

iPod Video Encoding There is a growing demand for watching video clips on portable devices such as mobile phones and the iPod Video. To meet this demand, WEB2.0LEECHER is capable of transcoding any downloaded video into a format that the iPod Video is able to play.

3.1.3 P2P Sharing and Video Browsing

In the third stage, we have developed TRIBLERSHARE, an enhanced version of Tribler. Figure 3.6 shows a screenshot. TRIBLERSHARE provides single-click-sharing, which turns the process of publishing a file into a matter of selecting that file.
Furthermore, WEB2.0LEECHER is partially integrated with TRIBLERSHARE such that TRIBLERSHARE enables the user to search and watch videos from YouTube, and LiveLeak. TRIBLERSHARE has a single search functionality that merges both the torrent and Web 2.0 search functionality. Searching for torrents only requires searching a local database, while searching for Web 2.0 items requires network communication. So, the search results of Web 2.0 search operations are appended to the search results of torrent search operations.

3.2 WEB2.0LEECHER

In this section, we describe the architecture and design of WEB2.0LEECHER. In Section 3.2.1, we explain the two major architectural decisions, and provide an overview of WEB2.0LEECHER. Section 3.2.2 describes the design of the GUI, Section 3.2.3 discusses the flexible web interface system, Section 3.2.4 describes the ratings system, and Section 3.2.5 describes the design of the download manager.

3.2.1 Architecture

In this section, we describe the two major architectural decisions that have been made for WEB2.0LEECHER. These are the choice between a stand-alone application or a Firefox extension and the choice between multithreaded communication or asynchronous communication. This section also gives an overview of the components of WEB2.0LEECHER.

Stand-alone vs. Firefox extension

The WEB2.0LEECHER is a stand-alone application, but we have also considered to develop WEB2.0LEECHER as an extension for the Firefox Internet browser. Creating a Firefox extensions improves the browsing experience, because only a single application is used for surfing the web. Furthermore, users do not have to install a separate application to use WEB2.0LEECHER. However, usage would be limited to Firefox users, so users with different browser, like Internet Explorer, will not be able to use WEB2.0LEECHER. In addition, Firefox extensions are written in JavaScript, while WEB2.0LEECHER has to be written in Python for easy integration with Tribler, which is already written in Python. Therefore, WEB2.0-LEECHER is created as a stand-alone application in Python.

Multithreaded I/O vs. Asynchronous I/O

When used actively, WEB2.0LEECHER has simultaneously multiple open network connections. For example, while there may be multiple active downloads, a user may also execute a search operations which requires one or more network connections. There are basically two ways to handle multiple network connections, multiple threads and asynchronous communication. We chose a multithreaded I/O approach in order to reduce development time and dependencies on external libraries. The Python standard library provides a module to retrieve web pages, which uses blocking I/O operations. Using asynchronous I/O would require to use a third-party library, like Twisted, or to write our own asynchronous HTTP module.

WEB2.0LEECHER Overview

Figure 3.2 shows the architecture of WEB2.0LEECHER. The core of WEB2.0-LEECHER is formed by the databases, which provide interfaces for the available content by using one or web interfaces. For each media type there is a separate database. WEB2.0LEECHER currently supports video, photo, and text, and accordingly there are three databases. A database retrieves its items from one or more Internet sites. For example, the Video Database retrieves its items from YouTube, LiveLeak, and Revver. 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 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 a site, knows how to execute search queries, knows how to parse the search results, and hides these site-specific details for the database by providing a generic interface. The separation of site-specific details and the database provide a flexible and easily extendable system. Adding support for a new site requires only to specify the site-specific details in a Web 2.0 interface. When a 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

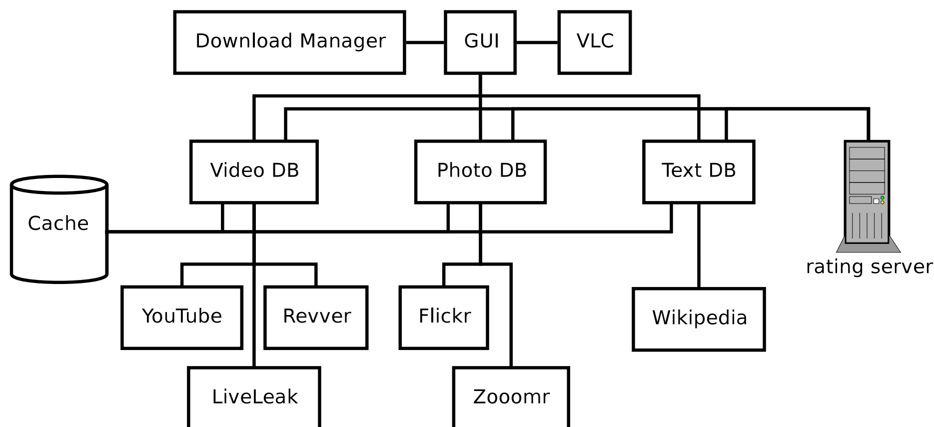

Figure 3.2: The architecture of WEB2.0LEECHER.

database aggregates the results of each interface into a single result set and returns this set as the result of the keyword search operation. The result set only contains metadata of items and not the item itself. Retrieving each item in the result set brings about a lot of extra communication, because the size of the metadata is several magnitudes smaller than the size of the item. Furthermore, the user may be only interested in a few items of the entire result set. In such cases, the extra communication is mostly wastage and the response times of the search operations would be 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, the databases are equipped with a cache. The cache is used to store the metadata of items which is retrieved as a result of search operations. Metadata that is stored in the cache can be retrieved without requiring any network communication, and thus speeding up the search operation. The size of the cache is unlimited, and metadata is never evicted from the cache.

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 a message is sent to the rating server containing the rating and IDs that uniquely identify the item and WEB2.0LEECHER installation. A new rating for a specific combination of item and installation IDs replaces any old rating for that combination. Using these records, the rating server keeps track of the average rating of each item. Average ratings are considered to be part of the metadata of an item. However, due to the temporal validity of average ratings, they are not stored locally and have to be requested from the rating server every time they are needed.

The GUI component enables the end user to execute search operations, view the search results, and select content items to download. When an item is selected to be downloaded, it is passed to the Download Manager, which will start a new download and notify the GUI of this download and the progress. With the GUI,

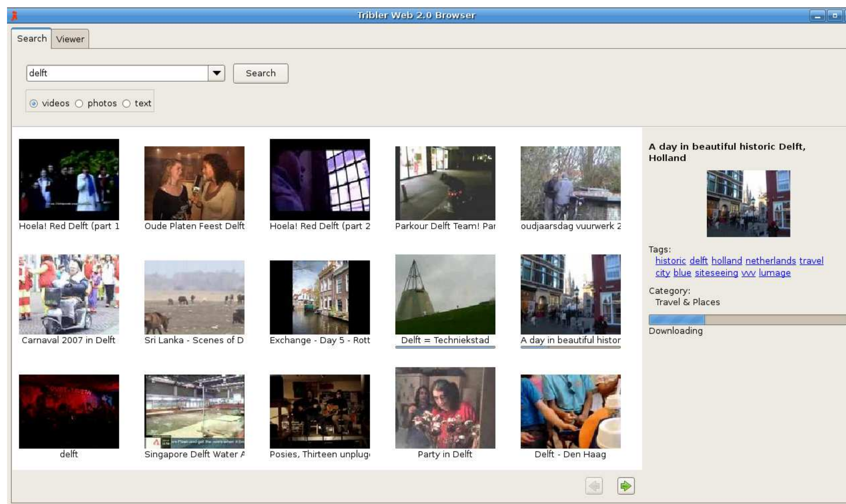

Figure 3.3: A screenshot of WEB2.0LEECHER after a search for videos on *delft* has been executed.

the user track and cancel downloads in the Download Manager. Furthermore, to downloaded items can also be viewed with the integrated viewer.
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 WEB2.0LEECHER consists of two tabs, see Figure 3.3 and Figure 3.3. The first tab, the *search tab*, allows users to enter search queries and shows the search results. The user enters a search query, selects the media type, and clicks the search button to start a new search.
The second tab is the *viewer* tab (see Figure 3.4). This tab shows the user a list of all files that have been downloaded or are being downloaded, and it allows the user the view these items.

Precaching

The WEB2.0LEECHER GUI divides search results over multiple pages like most web sites also do. Switching to a new page of the search results requires retrieve search results over the Internet, and may take therefore some time. To improve responsiveness, WEB2.0LEECHER does *precaching*. The WEB2.0LEECHER 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

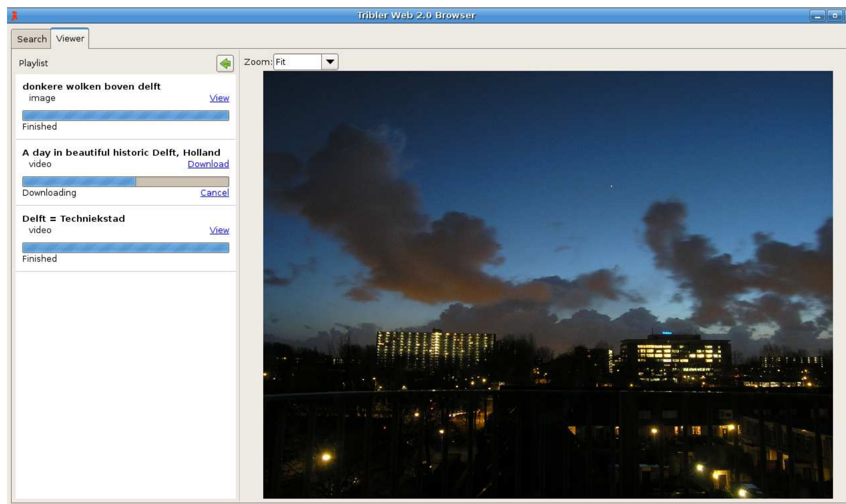

Figure 3.4: WEB2.0LEECHER viewing a photo of *Delft* that has been downloaded.

search results, the next two pages are being loaded. The time that a user has to wait for the next page to be loaded is reduced or may even be eliminated.

GUI updates

To keep the information presented in the GUI up-to-date we use a push model. The 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 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.
And finally, 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 most essential components of WEB2.0LEECHER are the databases with the Web 2.0 interfaces, because these components provide the functionality to retrieve content from sharing sites for videos, photos, etc. There are a number of base classes upon which a Web 2.0 interface can be built. These base classes handle requests for items and delivery of items to other components of the application. A Web 2.0 interface only needs to provide 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 WEB2.0-LEECHER. 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 exposes a few operations to retrieve the search results. These are *start*, *getMore*, *enough*, and *quit*.

start The *start* operations initializes the search operation sets up the necessary resources to perform 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 used operation. This operation submits requests for search results. To fulfill the requests, the Web 2.0 interface uses the keyword search of the site and parses the search results. Retrieving and parsing search results requires network communication over the Internet, so to maintain responsiveness of the application, results are returned asynchronously.

enough The *enough* operation is used in order to discard any outstanding requests. After *enough* is called, 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 named *getItem*. Each call to *getItem* must return the next item of the search results. The DBSearch is simple and uses only a single thread, and therefore its performance is not optimal. A performance gain can be achieved if multiple threads are used, which is exactly what ThreadedDBSearch does.

Threaded Web Scraping

DBSearch does not assume any structure of the web site, however, nearly all web sites share a similar structure of the search results page. This structure can be exploited for parallelism. The structure is as follows. After a user has started a keyword search operation, the search results page are presented to him. 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 most web sites, the limited metadata presented on the results search page is not sufficient for our purposes, and it is necessary to retrieve the web page of each item, which contains the complete metadata.

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

A Web 2.0 interface using the ThreadedDBSearch as base class must implement two operations, *parseItempage* and *parseItem*. The *parseItempage* operation is expected to fetch and parse the next search results page. *parseItempage* must a return a list of which the items each provides information for *parseItem* to fully retrieve the metadata of that item. ThreadedDBSearch regards the items in the list as opaque values, and Web 2.0 interfaces may put any type of data in this list. The *parseItem* method receives a single item from the list returned by *parseItempage*, and it must download the web page of the item and extract metadata. Before actually downloading the web page, *parseItem* should check the cache to reduce latency.

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

Multiple Web Sites

The third class is CompoundDBSearch which ,unlike the other classes, is not meant as base class for Web 2.0 interfaces. 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, in total too many items will be requested. 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-2.0LEECHER 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

Each Web 2.0 interface can use its own way for fetching data from web sites. There are basically two ways to retrieve data from a web site. 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 interpreted, and data can be identified within this structure. For example, the title of an item can be defined as the text enclosed by the `<h1>` element in the first paragraph (i.e., `<p>`) of the web page. Second, the web page can be treated as plain text from which data can be extracted using regular expression. Regular expression specify which web pages to fetch to perform a search, and how data can be extracted.
Each Web 2.0 Interface can use a different method for retrieving data. However, all the current interfaces fetch web pages and extract metadata using regular expression. The drawback of APIs is that each site has a different API and not all sites provide such a API. On the other hand, the approach of fetching web pages and parsing the results works universally for all web sites.
The few regular expressions shown in Figure 3.5 provide enough information to perform and parse keyword searches on YouTube. The few number of regular expressions also also reflect the flexibility and extensibility of the Web 2.0 Interfaces. Adding new support for a new Web 2.0 site basically consists of finding the correct regular expressions and applying them. Finding the correct regular expressions is trial-and-error process. They need to be restrictive enough that only the desired data is extracted and they need not to be too strict to correctly for all items. From our experience, this takes only a few hours of programming.

```
youtube.py (~/web2browser/web_2.0_browser/video) - VIM
File Edit View Terminal Tabs Help
youtube.py
14
15 ENCODING = "utf-8"
16
17 site = "youtube.com"
18
19
20 RE_SEARCHITEM = r"<a href=\"/watch\?v=(.*?)\".*?><img src=\"(.*?)\".*?></a>"
21 RE_TAG  = r'<meta name="keywords" content="(.*?)">'
22 RE_TAG2 = r'([^ ,]+)'
23 RE_CAT  = r'<a href="/browse\?s=.*?Video\+Category\+Link.*?>(.*?)</a>'
24 RE_NAME = r'<title>YouTube - (.*?)</title>'
25
26 URL_WATCH = "http://www.youtube.com/watch?v=%s"
27 URL_DL_VIDEO = 'http://www.youtube.com/get_video?video_id=%s&t=%s'
28 RE_VIDEOURL = r'player2\.swf\?video_id=([^&]+?)&.*?&t=([^&"]+?)(?:&|")'
29
30 URL_SEARCH =  "http://www.youtube.com/results?search_type=videos&search_quer
   y=%s&search_sort=relevance&search_category=0&page=%d"
31
32 RE_RESULTS_HASNEXT = r'class="pagerNotCurrent">Next</a>'
33
34
                                                          34,0-1         9%
```

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

3.2.4 Ratings

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

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

Storing and retrieving ratings are not vital to the primary functionality of WEB2.0-LEECHER. 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 WEB2.0LEECHER.

3.2.5 Download Manager

When a user decides to download a content item, the GUI passes the item to the Download Manager. The Download Manager notifies the GUI of new, finished,

and failed downloads and the progress of downloads. Furthermore, the Download Manager ensures that there is at most one active download for each content item.

3.3 TRIBLERSHARE

As mentioned in the introduction of this chapter, Tribler has to be enhanced in two ways. This enhanced version is named TRIBLERSHARE. First, TRIBLERSHAREhas to be able to take advantage from the large collection of content offered by various sites. This is exactly what WEB2.0LEECHER does. So WEB2.0LEECHER has been partially integrated with TRIBLERSHARE. In particular, the Web 2.0 interfaces and the Download Manager have been integrated. TRIBLERSHARE users can search for videos offered by YouTube and LiveLeak, download them, and view them. Any downloaded videos are automatically published using the scalable distribution system of Tribler. This requires that the process of publishing of an item can be carried out without any user involvement.
This is what the second modification of Tribler is about, which is the ease of publishing of Tribler has been improved. Ideally, the user only selects the file it wants to publish and Tribler takes care of the rest. The main problem in the BitTorrent process is the swarm discovery method. The traditional swarm discovery requires the publisher to set up a tracker, which will serve as a venue point for peers. It is likely that the majority of users do not have access to a server that is running continuously, which can be deployed up as tracker. Ideally, users do not require any knowledge about trackers or .torrent files, and just simply select the file to publish and WEB2.0LEECHER takes care of the rest. This can also be used for publishing videos from YouTube and LiveLeak because no user intervention will be necessary.
To remove the need for trackers, we use a different swarm discovery method that is called *decentralized tracking*. With decentralized tracking, each peer becomes a potential tracker for the torrent, so no dedicated trackers are necessary. This swarm discovery method requires no action from the user neither does creating a torrent with decentralized tracking.
In this section, we first present the integration of the Web 2.0 interfaces with Tribler in Section 3.3.1, and in Section 3.3.2, the decentralized tracking is explained.

3.3.1 WEB2.0LEECHER-TRIBLERSHARE Integration

We have only integrated the YouTube and LiveLeak interfaces with TRIBLERSHARE, 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 torrent search functionality to create a single search functionality to search in all content that is available through by means of torrents and Web 2.0 interfaces.
The WEB2.0LEECHER closely resembles the Tribler GUI. The search results are

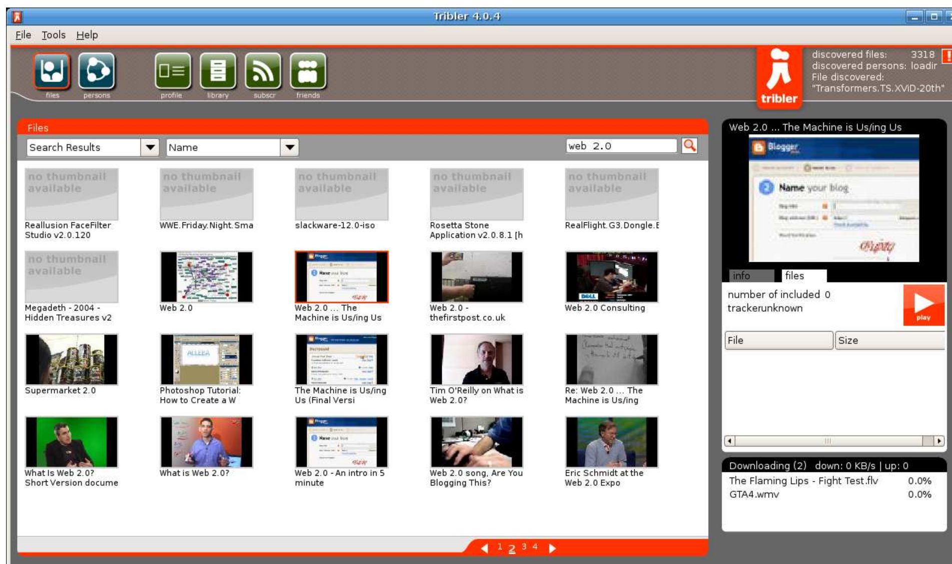

Figure 3.6: Keyword search on “web 2.0” in Tribler.

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 available very quick compared to Web 2.0 search operations because no network 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 TRIBLERSHARE item grid has been modified to request items from WEB2.0LEECHER 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 WEB2.0LEECHER. The downloads of the Download Manager are displayed together with the downloads of Tribler itself in the Tribler Library. When the item is completely downloaded it is treated the same as if the item was published by the user. A .torrent file is created which will cause the Tribler Library to view it as a torrent instead of a Web 2.0 item. The .torrent file is spread through BuddyCast to other Tribler users, so each item that is downloaded through a Web 2.0 interface is immediately injected in the Tribler network and, thereby, benefit from the scalability of Tribler. Thus, each item from a Web 2.0 interface needs only to be downloaded once through that interface, and then Tribler no longer depends on the Web 2.0 interface for that specific item and handles the

distribution of that item.

3.3.2 Decentralized Tracking

Besides the usual trackers, some BitTorrent clients support a distributed alternative for swarm discovery. This method is often referred to as *decentralized tracking*, and is achieved by using a *Distributed Hash Table* (DHT). A DHT is a hash table that spreads the storage of key/value pairs across the peers that are participating in the DHT.
There are two DHTs in use for BitTorrent swarm discovery, the 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 clients that use the other DHT, so peers registered in a swarm in the Mainline DHT cannot be discovered by peers using the Azureus DHT, and vice versa. Thus for each torrent there are two separate swarms in both DHTs.
We have chosen to integrate the DHT from the Mainline BitTorrent client into Tribler mainly to reduce development time. The Mainline DHT is, like Tribler, written in Python and can therefore easily be integrated. The Azureus DHT is, on the other hand, written in Java. And there is no documentation available of the Azureus DHT thus porting it to Python also requires to elicit its exact working from the Azureus source code. For a more in-depth analysis of both DHTs we refer to [7].
We have integrated the Mainline DHT with Tribler. The Mainline DHT is called *Khashmir*, and is based on the emphKademlia DHT [14]. This section first explains Kademlia, and then the modifications that are made to use if for swarm discovery.

Khashmir

Kademlia is a DHT that support the usual set and get operations. In Kademlia, keys are 160-bit values and participating nodes have a nodeID in the same 160-bit key space. Khashmir uses the infohashes of torrents as keys and the corresponding value is the list of peers that constitutes the swarm of the torrent. Key/value pairs are stored at nodes with the nodeIDs that are closest to the key. In order to calculate the distance between keys and nodeIDs, the XOR operator is applied to numbers, and the result is to be interpreted as an unsigned integer. Thus the distance between node A and B with IDs respectively A_{id} and B_{id} equals to $A_{id} \oplus B_{id}$.

Routing Table The routing table is organized in so-called k-buckets. A k-bucket covers a range of the key space, and can contain up to k nodes which are ordered from least-recently seen to most-recently seen. The k-buckets are organized such that the routing table can contain many nodes that are close and few nodes that are

far. k-buckets are filled with entries as a side effect of lookup and store operations and by a regular lookup for the node with distance 1.
When new nodes are added to the routing table there is a preference for old live nodes over new nodes. This preference has two advantages. First, research has shown that the longer a node is up the more likely it is that it will stay up for another hour [19]. Consequently, the nodes in k-buckets have a higher probability of being live. Second, it provides robustness in a hostile environment, because it is not possible to flush the routing table of nodes by flooding the system with new nodes. New nodes will only be added to the routing table if the old nodes have left the system.

Operations Instead of the usual get and set operations, Khashmir provides a *AnnouncePeer* and a *GetPeers* operation. The AnnouncePeer operation is used by node to announce to other nodes that it is participating in a particular swarm. The GetPeers operation returns the list of peers that are participating in a particular swarm. For both operations, the first step is to find the closest nodes to the target (i.e., the infohash). This is done by querying other nodes for the closest nodes to the target that they know of. This continues until no nodes can be found that are closer to the target. Subsequently, AnnouncePeer or GetPeers requests are sent to the closest nodes to a target.
When a node receives a GetPeers request for a particular torrent, the node will lookup the swarm information that it has locally stored and will send this list to the requester. This swarm information is a list of contact information of peers participating in the swarm of the torrent, and the requester can contact these peers to start exchanging pieces of a file.
When a node receives an AnnouncePeer request for a particular torrent, it looks up the peer list that it has stored for that torrent. Then, it adds the contact information of the requester to that list. This contact information will be included in responses of following GetPeers requests that the requested node will receive. Entries in swarm lists are only valid for a fixed period after which they are purged from the list. Consequently, peers in participating swarms have to execute the AnnouncePeer operation regularly.
For efficiency reasons, it is possible to perform both operations together for a particular torrent. In that case, the first step of finding the closest peers to the target is performed only once instead of twice (once for each operation).
When a peer is behind a firewall or is not able to receive any incoming connections for some other reason, it is not useful to perform AnnouncePeer because other peers will not able initiate connections to that peer. A Tribler peer learns from other Tribler peers whether it is connectable from the Internet. TRIBLERSHARE uses this information to decide whether to use the AnnouncePeer operation or only the GetPeers operation.

Chapter 4

Experiments and Evaluation

In this chapter we present the experiments and evaluation that we have performed to identify weaknesses in order to be able to improve WEB2.0LEECHER and TRIBLERSHARE. In particular, the performance of the Web 2.0 interfaces and swarm discovery method based on Khashmir have been evaluated. Furthermore, we present the data that has been collected as a result of usage of the Tribler 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. Finally, we present a end-to-end video experiment that shows the entire flow of a YouTube video that is downloaded from the web site and which is distributed with TRIBLERSHARE 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 TRIBLERSHARE clients with torrent gossiping. Another TRIBLERSHARE client will then download the same video through the scalable BitTorrent and Khashmir.
Section 4.1 describes the experiments and evaluation of WEB2.0LEECHER, Section 4.2 presents the community measurements of WEB2.0LEECHER, Section 4.3 discusses the Khashmir measurements, and Section 4.4 describes the end-to-end video experiment.

4.1 Web 2.0 Interfaces

In this section, we present the performance analysis of the Web 2.0 interfaces. The Web 2.0 interfaces are one of the most crucial components of the WEB2.0-LEECHER because they fetch content from the various web sites. Performance is important for the usability of WEB2.0LEECHER 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 WEB2.0-LEECHER to be a viable alternative for visiting Web 2.0 sites, the performance of the Web 2.0 interfaces should be in the same order of magnitude as the perfor-

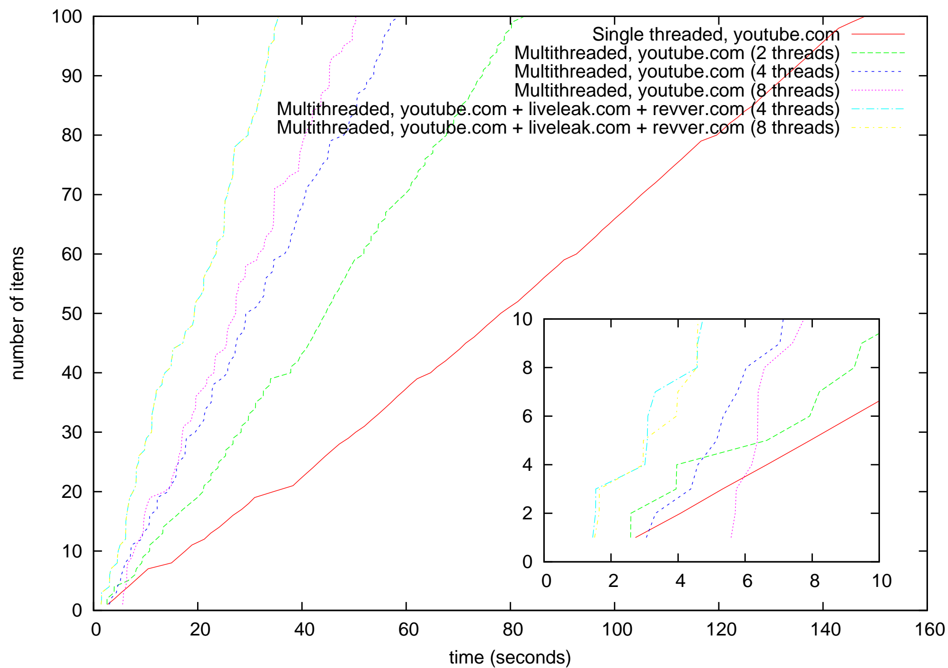

Figure 4.1: The number of items retrieved from Web 2.0 sites as a function of time for a single keyword search.

mance of a Internet browser such as 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.

As discussed in Chapter 3 there are three types of Web 2.0 interfaces. The first type is a single-threaded search that uses a single thread to perform synchronous communication, the second type is a multithreaded search that uses multiple threads. and, the third version combined multiple searches to create a single compound search that uses multiple sites. Figure 4.1 shows the response times for a single keyword search for these three versions and different number of threads. The figure shows the amount of time needed to fetch the first 100 items of the search results. Additionally, the figure shows an magnification of the first 10 seconds of the measurements to clearly display the latency to the first returned item for each measurement.

These measurements have been made with an automated version of the WEB2.0-LEECHER that at startup immediately executes a search operations 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 WEB2.0LEECHER is flushed because the cache may greatly reduce the amount of necessary communication and thus also the amount of time. A non-empty cache may therefore give a distorted reflection of the performance of the Web 2.0 interfaces.

The time is measured from the moment that the search is started until the moment at which the 100th item is delivered by the Web 2.0 interface. Included in these measurements is the time needed to display the search results in the GUI, which among others includes rescaling and rendering thumbnails.
These measurements have been run on the video Web 2.0 interfaces, because the WEB2.0LEECHER supports the most number of web sites for video. Furthermore, measurements on a single Web 2.0 interfaces have been made on YouTube, since it is the leading Web 2.0 sharing 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 interface with just one web site is by far the slowest method for retrieving search results. The first item is available 2.7 seconds after the search has been initiated. The 100 items are delivered in approximately 148 seconds. On average, between each new item delivered by the Web 2.0 interface the user has to wait 1.48 seconds.
The multithreaded YouTube interface with two threads provides a substantial speedup over the single threaded interface. All the 100 items were available in under 83 seconds. This means a speedup of approximately 1.79 compared to the single threaded interface, and an efficiency of approximately 0.895. With four threads, the speedup equals 2.94 and the efficiency approximately 0.74. Using eight threads compared to four threads yields a speedup of only 1.15. The limited speedup is caused by the limited available bandwidth.
The speedup from using multiple sites is limited compared to using multiple threads. However, using multiple sites comes with other advantages, smaller delays and a more diverse collection. The smaller diagram in Figure 4.1 shows an enlargement of the first 10 seconds. When only YouTube is used as source for videos then the first item is available in approximately 2 to 3 seconds. This is a result of the delay of YouTube. With multiple sites the first item is available in 1.5 second. One or more of the other sites have a lower latency than YouTube and provide the first item quicker. In this case, LiveLeak has a smaller delay and provides virtually always the first item. See Table 4.1 for a comparison on the latency of YouTube, LiveLeak, and Revver.
To provide a smaller delay, it is possible to only use LiveLeak instead of combining YouTube and LiveLeak. In that case, however, users would be deprived of the wealth of content provided by YouTube. Using multiple sites yields a greater variety of content. Especially when the focus of the videos sites is different. For example, YouTube profiles itself as the family friendly video sharing site. Therefore, any videos that are considered offensive or inappropriate in some other way are removed. On the other hand, LiveLeak and Revver do not strive to be family friendly like YouTube.

	YouTube.com (20 items)	LiveLeak.com (12 items)	Revver.com (15 items)
first run	4331	1659	7727
second run	4997	1761	13240
third run	4700	1712	9860
average	4676	1711	10276
WEB2.0LEECHER	7575	6158	6200

Table 4.1: Firefox response time vs. WEB2.0LEECHER in milliseconds.

WEB2.0LEECHER 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 WEB2.0LEECHER 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-2.0LEECHER for the equivalent number of items shown per search result page of the corresponding web site.
From Table 4.1, we see that the is approximately 62% slower than YouTube, 260% slower than LiveLeak, and 40% faster than Revver. Overall, WEB2.0LEECHER is slower than the using an Internet Browser. However, note that all the Web 2.0 interfaces also fetch the individual page for every item, from which extra metadata is extracted for the user. For example, YouTube and Revver do not provide the associated tags with each item on the search results page, while WEB2.0LEECHER does. However, this requires additional data retrieval, which costs extra time.

4.2 Real world usage

WEB2.0LEECHER has been released into the public as the Tribler Web 2.0 Browser, which was announced on April 10th, 2007 (see Figure 4.2). Through usage of the Web 2.0 Browser by the public, we were able to collect data as a result of Web 2.0 Browser installations contacting the rating server.
In this section we present this data. First, we describe which datasets have been obtained. Subsequently, we present the statistics that have been extracted from

EMERCE

YouTube-filmpjes downloaden

10 april 2007 - Door Tonie van Ringelestijn

Een student van de TU Delft heeft een programma gemaakt waarmee gebruikers filmpjes en foto's van sites als YouTube en Flickr direct kunnen opslaan op hun harde schijf of iPod. Deze 'Web 2.0 browser' is onderdeel van het p2p-project Tribler van de Nederlandse universiteiten. Het systeem draait straks ook op settopboxen, zodat YouTube-content op tv kan worden bekeken.

De Web 2.0-browser is een afstudeerproject van Fabian van der Werf van de Technische Universiteit Delft. Hij maakte het ietwat zware programma, dat naast Tribler geïnstalleerd moet worden, in drie maanden. Het meteen via de p2p-software van Tribler delen van de gedownloade YouTube-filmpjes en andere content is nog niet mogelijk, maar de experimentele software kan nog worden uitgebreid.

De Web 2.0-browser doorzoekt de sites van diverse foto- en videosites, zoals YouTube, Revver, Zudeo, Zooomer en Flickr. Het wegschrijven van YouTube-video's naar het

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

these datasets.

4.2.1 Datasets

In this section, we describe the data that has been collected by the rating server that is collected from April 10th, 2007 to August 3rd, 2007.
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 each contain the installation ID, the unique item ID, and the rating. Every time a user rates an item, a message is sent to the rating server a new record is added to the user ratings. A new rating for a particular combination of installation ID and item ID overwrites any previous records with that combination. Thus the most recent ratings issued by an installation are kept in the database.
The dataset of average rating requests is a collection of records that contain the installation ID, the unique item ID, and the time at which the request was submitted. Every time the Web 2.0 Browser needs to display the average rating, the average rating is requested from the rating server. The rating is displayed in two places. First, the rating is displayed on the detail panel of the search results. Thus an average rating request for an item is sent when it is selected in the search results overview, it does not have to be downloaded. Second, the average rating is also displayed on the viewer tab and is for the current showing or playing item. Consequently, an average rating for an item is requested every time that item is viewed.
The two datasets mentioned above are the only datasets that have been collected

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

4.2.2 Average Rating Requests

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

Number of installations

Practically every installation that has been used has sent at least one average rating request. Along with each rating request, installation ID is sent along. By counting the unique installation IDs in the rating requests, we can deduce the number of unique installations that have contacted the rating server, which equals to 420.
The installation ID is generated on the first run after the installation of the Web 2.0 Browser and incorporates the current time. It is possible to extract this time from the installation ID, so 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

Because average rating requests are sent with most usages of the Web 2.0 Browser, these requests can 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. The most used installation has sent more than 2.5 times more rating requests than

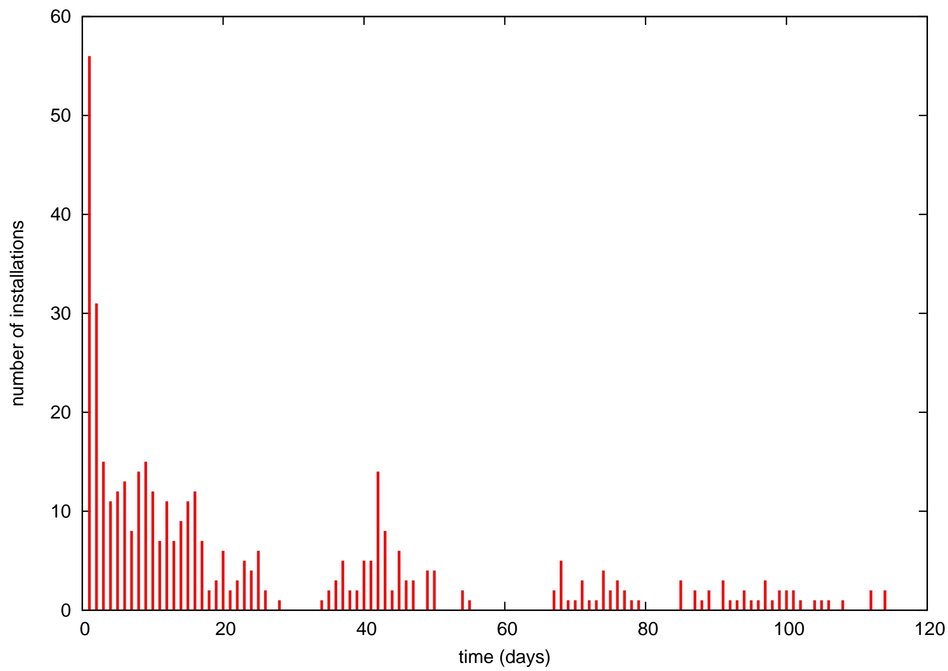

Figure 4.3: The number of first runs of installations of the Web 2.0 Browser since April 10th, 2007.

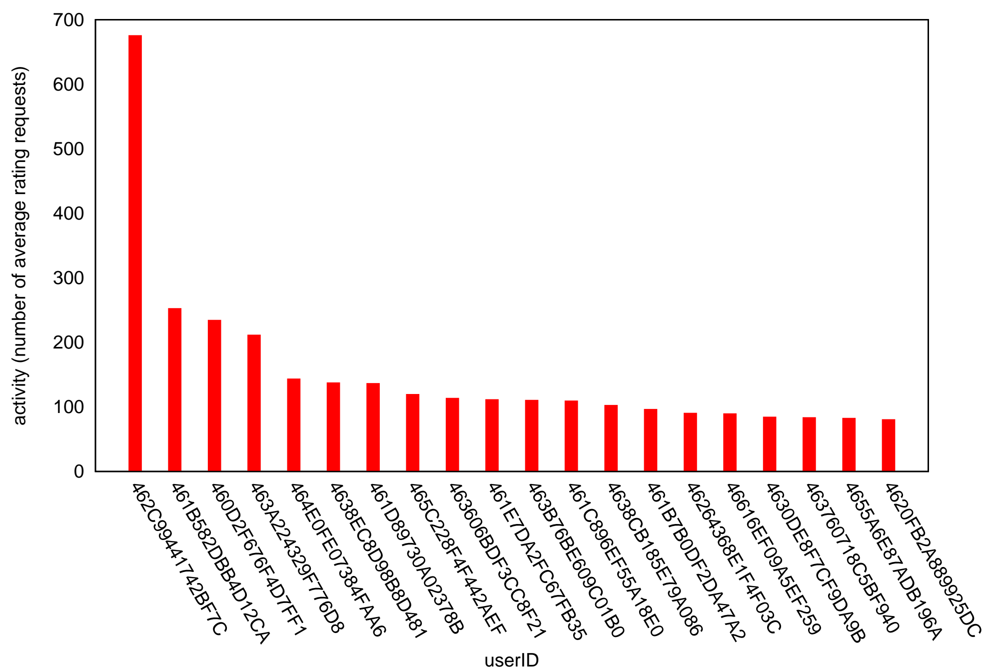

Figure 4.4: The activity of the 20 most active of the Web 2.0 Browser.

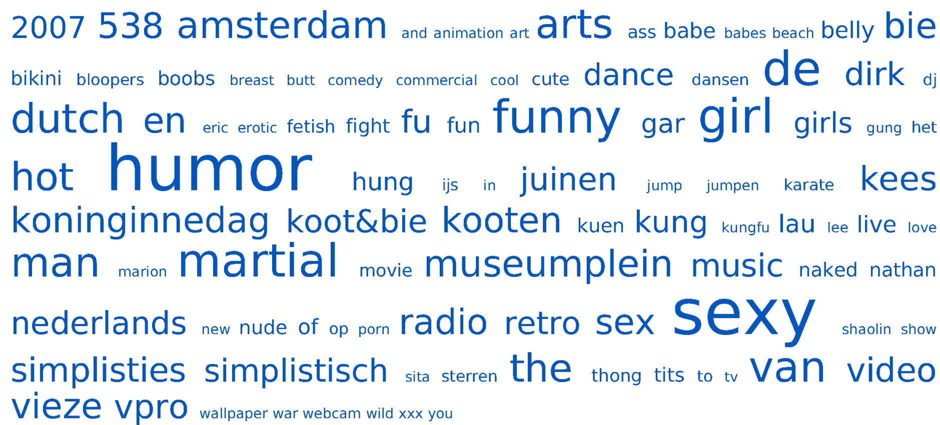

Figure 4.5: The 100 tags with the most rating requests.

the number of rating requests sent by the second most used installation. The difference among the other installation is much less. The number of rating requests decreases more moderately between installations.

Most Viewed Content

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

The tagcloud shows that there was likely 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 users are likely to be 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

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

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

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

Table 4.3: The distribution of the ratings.

compute the covariances is the unique items from the average rating requests with the associated tags and the site that delivered the item. Table 4.2 shows the ten tags with the strongest covariances with each site.
YouTube is a very popular video site with many uploaders. Consequently its video collection is very diverse, and it even has dutch content. LiveLeak and Revver are less popular than YouTube and there is much less dutch content available from these sites.
The content that has been provided by Revver is mainly adult oriented. Although adult content is not the primary focus of Revver, it is not excluded by the site as YouTube does, which aims to be family friendly. Thus Revver is able 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” suggest humor related videos. The tag “iraq” indicates videos on the war in Iraq. LiveLeak has a focus on news videos and the site even has a separate category for the war in Iraq.

4.2.3 Ratings

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

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

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

The ratings are strongly biased towards a positive rating, i.e., four or five stars. Nearly 60% of all ratings have a level of five stars, and nearly 74% is either four or five stars. Among the negative ratings, the one star rating is issued the most. Either users find all videos of good quality, or users tend to only vote for quality items and to not vote at all for bad quality items.
Table 4.4 shows the total number of ratings and the average rating for each site. By far, the most ratings were issued for items that come from YouTube, and users were also the most positive about the content of YouTube compared to the other web sites.

4.3 Khashmir Problems

In this section, we discuss the behavior and performance of Khashmir. For the sake of clarity, we refer to hosts participating in a BitTorrent swarm as peers and hosts participating in the Khashmir DHT as nodes. See Section 3.3.2 for more information on Khashmir.
A Khashmir swarm discovery operation is an iterative process in which new peers that are participating in a particular swarm may be learned in each iteration. Instead of accumulating the results of the iterations, peers found in an iteration are immediately made available to the caller of the discovery operation. This enables clients to connect to newly found peers, while the swarm discovery operation has not yet finished. So the results of a swarm discovery operation consists of tuples of a timestamp and a list of peers.
In this section, we first show the behavior of Khashmir for a large swarm. From these measurements, we conclude that the timeout value used by Khashmir could be set much stricter in order to improve performance.

4.3.1 Khashmir Behavior

In this section, we show Khashmir's behavior for a single torrent with a large swarm. We have performed a number of swarm discovery operations for the most popular torrent in the category "TV Shows" from Mininova (`mininova.org`).

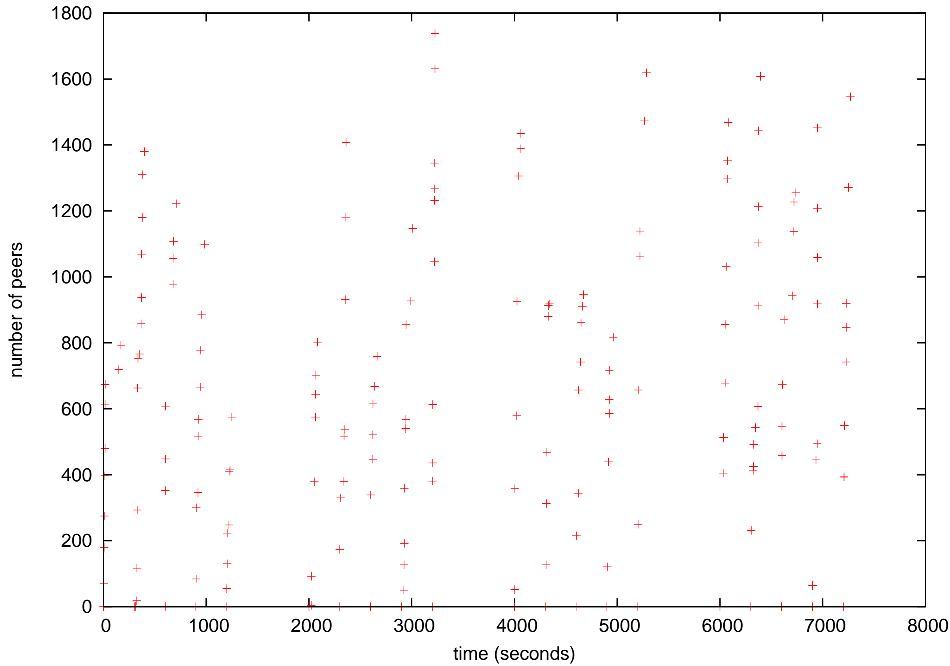

Figure 4.6: Swarm discovery operations for a large swarm.

Figure 4.6 shows the result of this experiment. Peers found in the swarm by Khashmir are indicated by plus signs. The number of peers discovered as indicated by Figure 4.6 are accumulative within a single swarm discovery operation. For every operation, there is an almost straight column of plus signs. For example, the first swarm discovery operation starts at 0 seconds with no peers yet discovered. Then Khashmir discovers quickly in total 674 unique peers. Then, after approximately 150 seconds, Khashmir discovers more peers and has in total discovered 793 unique peers in the first swarm discovery operation.

At 0, 2000, 4000, and 6000 seconds, we run five discovery operations with intervals of five minutes.

At 2000, 4000, and 6000 seconds, the routing table is flushed causing the node to forget any peers it has discovered in previous discovery operations. The Khashmir node performing the measurements is bootstrapped with a single other stable Khashmir node that we run ourselves.

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

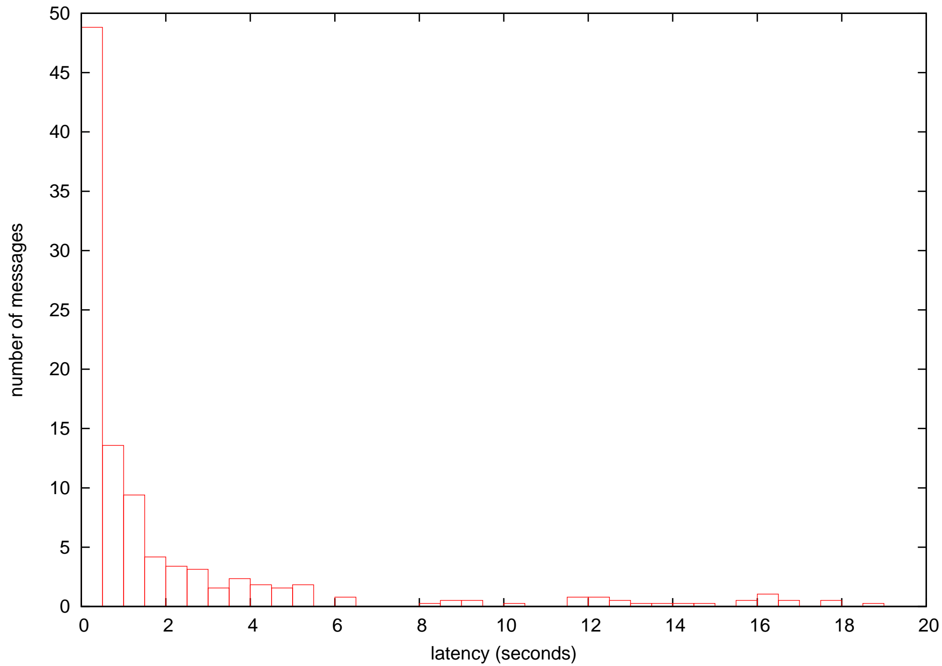

Figure 4.7: Distribution of the latency of Khashmir requests.

a small number of nodes that have swarm information is contacted. In the current implementation, a swarm discovery operation is stopped when 8 nodes that have swarm information have been contacted.

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

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

4.3.2 Khashmir Timeout

In this section, we try to improve the performance of Khashmir by using a more strict timeout value. As shown in the previous experiment, on average, 52% of the requests receive no response. For these requests, Khashmir waits for a timeout,

which is 20 seconds. The number of concurrent outstanding requests of a DHT operation is limited by Khashmir, so while waiting for a timeout, no new request can be sent to another node due to this limitation. Considering the 20 second timeout and the 52% of unsuccessful requests, Khashmir spends a significant amount of time waiting for responses that will not come.

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

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

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

Figure 4.9 shows for the top 500 torrents the number of peers that is discovered at the end of the swarm discovery operation. In general, the 20 second timeout Khashmir finds more peers.

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

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

Furthermore, this experiment shows clearly the main advantage of Khashmir over the traditional tracker, which is the independence of a central component. In total for 1016 torrents a Khashmir swarm discover operation has been performed. For each of these torrents, the scrape extension of the tracker was contacted, up to three attempts, to learn the swarm size according the tracker. For 38.4% of the torrents the tracker failed to respond. For another 2.3%, the tracker indicated there were no peers in the swarm. Khashmir found an empty swarm for 2.6% of the torrents, and found at least one peer for the rest of the swarms.

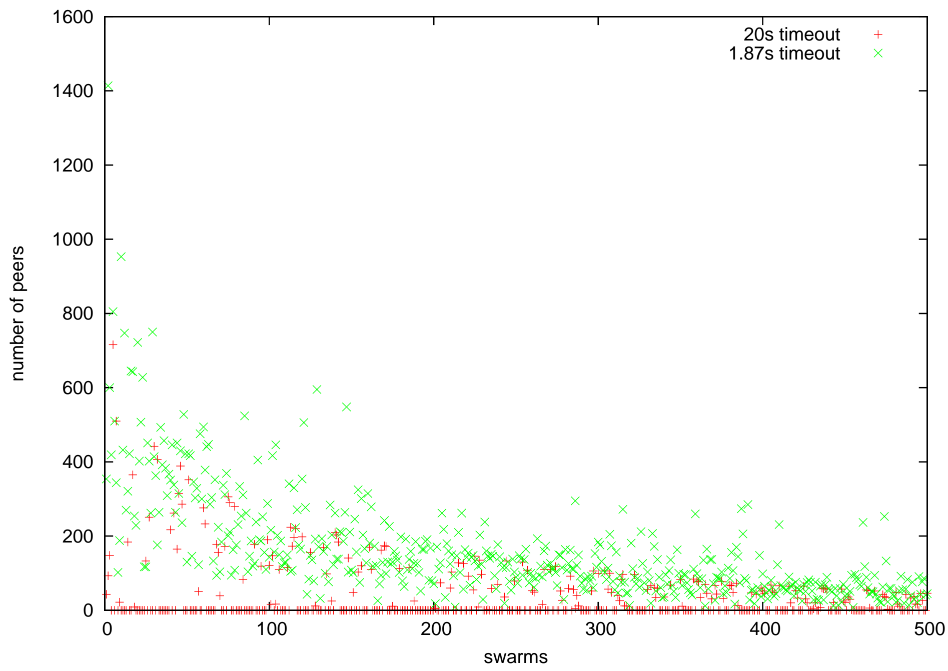

Figure 4.8: Number of peers discovered after 10 seconds.

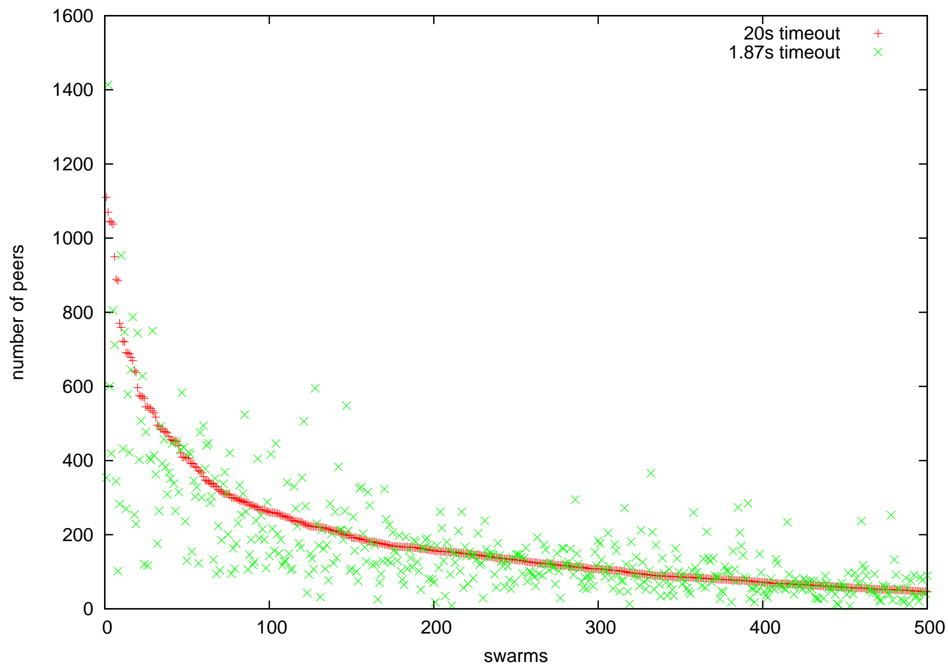

Figure 4.9: Number of peers discovered at the end of the swarm discovery operation.

4.4 End-to-End Video

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

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

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

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

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

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

Figure 4.10: Log of the publisher with IP 130.161.158.60.

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

Figure 4.11: Log of the consumer with IP 130.161.158.216.

Chapter 5

Conclusions and Future Work

In this thesis, we have addressed the problem of the poor scalability of Web 2.0 sites. For this purpose, we have created a mechanism for making content of external Web 2.0 sites available for Tribler users, we have provided the functionality in Tribler for users to publish and share their content with other users, and finally, by combining the previous two points, we have also decentralized external Web 2.0 sites. We have developed a proof of concept based on Tribler, called TRIBLERSHARE.

In order to provide access to the content offered by other Web 2.0 sharing sites, we have developed a Web 2.0 content aggregator called WEB2.0LEECHER, which is integrated with TRIBLERSHARE. WEB2.0LEECHER fetches content items from multiple Web 2.0 sites, transforms them into a uniform representation, and presents them to the end user. Content that is fetched using WEB2.0LEECHER is immediately published with TRIBLERSHARE in the Tribler network, so further distribution occurs in a scalable manner. WEB2.0LEECHER is easily extensible; adding support for a web site requires only to specify the structure of that site and takes only a few hours. Multiple web interfaces can be aggregated such that the collections of different sites appear as one logical collection. This enables users to navigate through a video collection which is created by aggregating the web interfaces of multiple video sharing sites. Because WEB2.0LEECHER provides an overarching architecture for Web 2.0 sites, we consider WEB2.0LEECHER a disruptive technology.

In order to enable users to publish content in a user-friendly point-and-click interface, TRIBLERSHARE uses a DHT for swarm discovery. The usual BitTorrent swarm discovery method by means of a tracker requires publishers to possess knowledge of the underlying BitTorrent system, because they are required to set up a tracker and to create a .torrent file. By using a DHT for swarm discovery, each peer essentially becomes a potential tracker for each torrent, so no dedicated tracker is necessary. The DHT takes care of the routing that is necessary to find the peers that function as trackers for a particular torrent. TRIBLERSHARE handles the .torrent creation, injects the .torrent file in the Tribler network, and makes the pub-

lisher the initial seeder for the torrent. The only thing the user has to do is simply select the file he wishes to publish.
By combining the WEB2.0LEECHER functionality with the publishing capabilities of TRIBLERSHARE, TRIBLERSHARE is also capable of decentralizing external Web 2.0 sites. Every content item that is retrieved from an external site with WEB-2.0LEECHER is immediately and automatically published into the Tribler network, just as if the item was published by the user. As a consequence, once a content item is fetched from an external site, further distribution of this item in the Tribler network benefit fully from the scalability and robustness of Tribler.
Our measurements indicate that the response times of search operations performed by WEB2.0LEECHERare of the same order of magnitude of the response times that are incurred when web interface of a Web 2.0 site is used. We believe these response times to be acceptable for the end user. By applying precaching, the time a user has to wait for search results can be greatly reduced dependent on the user's behavior.
The implementation of the DHT that we use is not optimal. The DHT often blocks waiting for timeouts, because of the limited number of concurrent outstanding requests and because many requests never receive a response. It would be better to apply two timeout values: a short timeout after which new requests may be sent and a longer timeout after which a request is given up. Furthermore, we have encountered two severe bugs in the implementation that greatly reduced the performance of the DHT. Therefore, we consider the implementation of the DHT to be immature.
Compared to existing Web 2.0 sites, TRIBLERSHARE falls short in functionality for letting users interact besides sharing videos, photos, etc. For example, web sites often enable users to send messages directly to other users, and many sites allow users to add comments to content items, which can be read by other users. These features enable users to interact and contribute to the *socialness* and thus the extent of Web 2.0. TRIBLERSHARE does not provide such functionality, and this lack of functionality limits its Web 2.0 level.
In addition to functionality, TRIBLERSHARE can also be improved in terms of performance. The performance of downloading items that originate from a web site can be improved by using the web site as a web seeder. So besides other peers, a leecher may also download parts from the original web site. The performance of BitTorrent is good for popular torrents, torrents with a smaller swarm do not perform as well, and for these cases, web seeding may strongly improve performance.

Bibliography

[1] Babelgum. www.babelgum.com.

[2] Joost. www.joost.com.

[3] Skype. www.skype.com.

[4] Vuze. www.vuze.com.

[5] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371, 2004.

[6] Bram Cohen. Incentives build robustness in bittorrent. In *Proceedings of Workshop on Economics of Peer-to-Peer Systems*, 2003.

[7] Scott A. Crosby and Dan S. Wallach. An analysis of bittorrent's two kademlia-based dhts. Technical Report TR07-04, Rice University, May 2007.

[8] Michal Feldman, Christos Papadimitriou, John Chuang, and Ion Stoica. Free-riding and whitewashing in peer-to-peer systems. In *PINS '04: Proceedings of the ACM SIGCOMM workshop on Practice and theory of incentives in networked systems*, pages 228–236, New York, NY, USA, 2004. ACM.

[9] Dan Frommer. Your tube, whose dime?, April 2006. http://www.forbes.com/intelligentinfrastructure/2006/04/27/video-youtube-myspace_cx_df_0428video.html.

[10] Jim Giles. Internet encyclopaedias go head to head. *Nature*, 438:900–901, December 2005.

[11] Lee Gomes. Will all of us get our 15 minutes on a youtube video?, August 2006. http://online.wsj.com/public/article/SB115689298168048904-5wWyrSwyn6RfVfz9NwLk774VUWc_20070829.html.

[12] Yang Guo, Kyoungwon Suh, Jim Kurose, and Don Towsley. P2cast: peer-to-peer patching scheme for vod service. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 301–309, New York, NY, USA, 2003. ACM.

[13] ipoque GmbH. Internet study 2007, November 2007. http://www.ipoque.com/media/internet_studies/internet_study_2007.

[14] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 53–65, London, UK, 2002. Springer-Verlag.

[15] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu. Peer-to-peer computing. Technical Report HPL-2002-52, HP Laboratories, March 2002.

[16] J.D. Mol, D.H.J. Epema, and H.J. Sips. The orchard algorithm: Building multicast trees for p2p video multicasting without free-riding. *IEEE Transactions on Multimedia*, 9(8):1593–1604, December 2007.

[17] Tim O'Reilly. What is web 2.0: Design patterns and business models for the next generations software, September 2005. http://www.oreilly.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web20.html.

[18] J.A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D.H.J. Epema, M. Reinders, M. van Steen, and H.J. Sips. Tribler: A social-based peer-to-peer system. *Concurrency and Computation: Practice and Experience*, 20:127–138, February 2008.

[19] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *MMCN '02: Proceedings of Multimedia Computing and Networking*, 2002.

[20] Kundan Singh and Henning Schulzrinne. Peer-to-peer internet telephony using sip. In *NOSSDAV '05: Proceedings of the international workshop on Network and operating systems support for digital audio and video*, pages 63–68, New York, NY, USA, 2005. ACM.

[21] Inc. WikiMedia Foundation. Frequently asked questions - wikimedia foundation. http://wikimediafoundation.org/wiki/Wikimedia:About.

[22] Inc. WikiMedia Foundation. Financial statements, june 30, 2006, 2005, and 2004, November 2006. http://upload.wikimedia.org/wikipedia/foundation/2/28/Wikimedia_2006_fs.pdf.