University of Zurich[UZH]

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Scraping BitTorrent Trackers

Gian Marco Jutz
Zürich, Switzerland
Student ID: 07-726-748

Supervisor: Andri Lareida, Dr. Thomas Bocek
Date of Submission: August 21, 2013

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Bachelor Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

BitTorrent is one of the most used file sharing protocols on the Internet. Understanding the distribution and behavior of BitTorrent users has a beneficial impact on validation of future BitTorrent related applications.
This work describes the design and evaluation of a tool to acquire data about peer behavior in a BitTorrent environment. A distributed approach has been chosen which allows collecting this information from different nodes with a variety of user configurable parameters. This tool has been deployed on Emanics Lab nodes and during a process of one week has been data collected.
The results uncover high similarity in the evolution of different BitTorrent swarms. It has revealed a periodic behavior of BitTorrent users as well as high fluctuation of peer numbers during short periods.

Contents

Chapter 1

Introduction

With the development of peer-to-peer (P2P) networks the traffic landscape of the internet has radically changed. Today, P2P applications are still responsible for a vast amount of internet traffic [1]. Even though, many peer-to-peer applications have been proposed and implemented in the past 10 years and different concepts have been improved and expended, the behaviour of P2P networks and its traffic handling is still a major field of research [2].

1.1 Motivation

Developers and scientists are working on improvements and extensions of the BitTorrent protocol or related applications. Traffic shaping techniques to discriminate the traffic in P2P networks can induce negative side effects on P2P users. Handling peaks with higher capacity infrastructure may lead to over provisioning. Caching might be a proposal for a solution of this problem. To evaluate and especially validate this approach, realistic experiment input data of BitTorrent user behaviour is beneficial. In a BitTorrent [3] P2P network a tracker [4] coordinates the communication between different clients. Whenever a peer connects or disconnects from a tracker an announce message is sent. Due to this fact, it is inevitable that a tracker has complete knowledge of all clients connected to its BitTorrent networks. Trackers offer a service to give a snapshot of the client constellation of a particular network, but only for a specific point in time. Observing the client behaviour over a longer period has an important impact on research and further protocol improvements. Therefore. this thesis designs, develops and tests an application to acquire sets of data from different torrent swarms. A BitTorrent swarm refers to the set of peers currently connected to a tracker. Due to the fact that this tool should provide flexibility, reusability as well as extensibility, a detailed documentation is required. To assure that the application is able to perform this task absolutely reliable, deployment in a distributed environment is crucial. In a final step, a collection of data from a real life environment is acquired for a period of at least one week and subsequently evaluated.

1.2 Description of Work

The work for this thesis was structured in three major parts, which had to be completed one after another. In a first step, background knowledge about the topic had to be acquired and possibly arising problems identified. Based on these findings, a first rough concept was worked out. After several iterations of improvement steps, a final design choice was made.
The following step included the implementation of the previously decided design choice. In an iterative and incremental development procedure, first a simple and basic application was implemented, improved and extended in several further steps. Between each step, a short period of testing in a real life environment was carried out to assure reliability, correct functionality and to find undetected problems. During all these steps, the documentation had been extended and improved.
This tool, which will be referred to as Scraper in further parts of this work had been deployed in a distributed test environment and the results evaluated. Final errors had been fixed and the application extended by adding smaller modifications.

1.3 Thesis Outline

In the following chapter 2, a brief outline on P2P in general and especially on BitTorrent is given. The main part of this chapter explains the essential mechanisms to gather peer information like IP addresses from trackers to identify peer swarms. As a final part of this chapter an overview on related work and their conclusions will be given.
In chapter 3, the design choice will be explained. Motivation and incitement of the final project architecture and its overall functionality will be explicated. In addition, a closer look will be taken at how this design choice tries to fulfil the requirements. Further, it will show the application design in detail. How the different encoding and decoding mechanisms have been implemented and how the tool configuration works.
In a fifth chapter follows an introduction to the test environment Emanics Lab and how tests have been performed. In second part of this chapter, results are presented and discussed.
The final chapter of this thesis provides a summary and a conclusion.

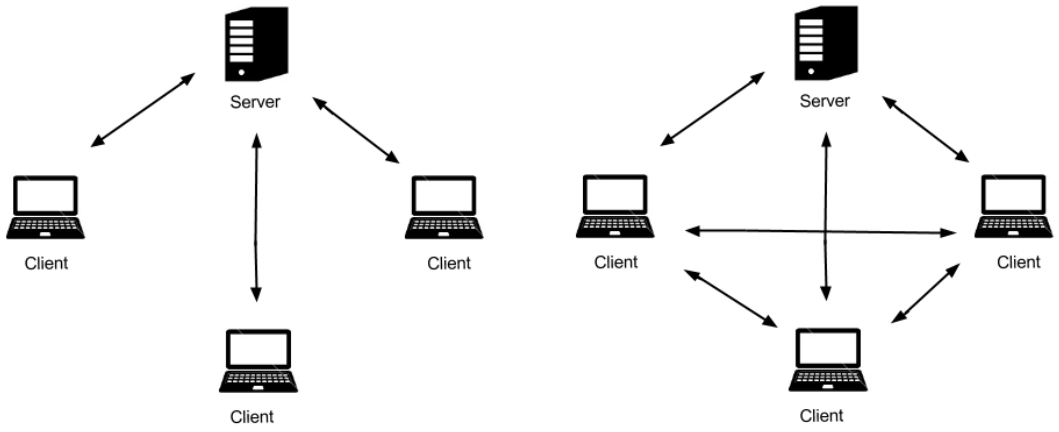
Chapter 2

Background and Related Work

Before the final design choice for the Scraper is presented, first, a brief description of the common P2P mechanisms is given. It will be explained why it outperforms in particular situations the traditional client/server architecture.
The section 2.2 will focus on the general BitTorrent architecture and functionality. Particularly, important aspects of BitTorrent regarding this work will be precisely explained. In addition, it will be demonstrated in detail how peers in a BitTorrent network communicate with each other and how an initial set of peers is obtained by a BitTorrent client. This chapter will be concluded by looking at related work.

2.1 Peer-to-Peer Architecture

Although, the basic client/server architecture is still important for computer networks, due to the demand of a continually growing amount of data on the internet, a new architecture approach has been designed. Especially file sharing, media streaming and communication through networks have gained popularity in recent years [5]. In traditional client/server architectures as shown in Figure 2.1a, the server can easily become a bottleneck for such a huge amount of users and demand of data, because a server acts as a single centralized system. Servers are not able to meet all the demands concurrently, as a result, the P2P architecture has been developed (Figure 2.1b).
In contrast to conventional single server systems, clients bear in a P2P design the cost of file transfer and organization together, which results in higher robustness and scalability. Clients not only act as consumer of a resource, they act as well as a supplier, the consequence is a better network utilization. Multiple connection with several sources at the same time are possible without a centralized instance.



(a) Client/Server (b) P2P

Figure 2.1: Comparison of C/S and P2P architecture

2.2 BitTorrent

Bram Cohen [6] developed with BitTorrent a peer-to-peer protocol that focuses on the distribution of large files in a sophisticated way. An important difference between BitTorrent and other P2P protocols is the implementation of a tit-for-tat strategy which rewards clients for sharing content in an unselfish and altruistic way. This strategy offers an effective way to make other clients not to just download files but also upload them, which is an important requirement for a successful peer-to-peer system.
BitTorrent file sharing is based on a common P2P sharing scheme. Clients subscribe and download chunks of a particular file from several different other peers. As a first step in joining a BitTorrent network, a client needs the address of other peers. To obtain this information, a user visits initially an index site for BitTorrent content and downloads the meta-data file(.torrent) which includes one or several tracker addresses. Trackers are servers, which are storing information about one or many BitTorrent networks from several different torrent files. Figure 2.2 shows the typical interaction between an index site, the peers and the trackers.
To publish content with BitTorrent a user generates a .torrent file. The content is separated into equal sized pieces. Hashes of the pieces are added to the meta-data file which allows the client application to verify the received data. A .torrent file provides clients with all the necessary information to participate the network.
Scraping a tracker denotes the sending of a request for information about one or several torrents to a tracker. A tracker that supports scraping responses with the number of peers currently connected to the requested torrents. In contrast to sending a scrape message implies the sending of an announce message that a client intends to connect to a particular torrent network for the first time or needs additional clients. The tracker responds beside the number of seeders and leechers with a list of 50 randomly chosen peers. Due to the fact that scrape messages are less bandwidth consuming, a client should send a scrape request in advance to decide whether an announce message is beneficial.

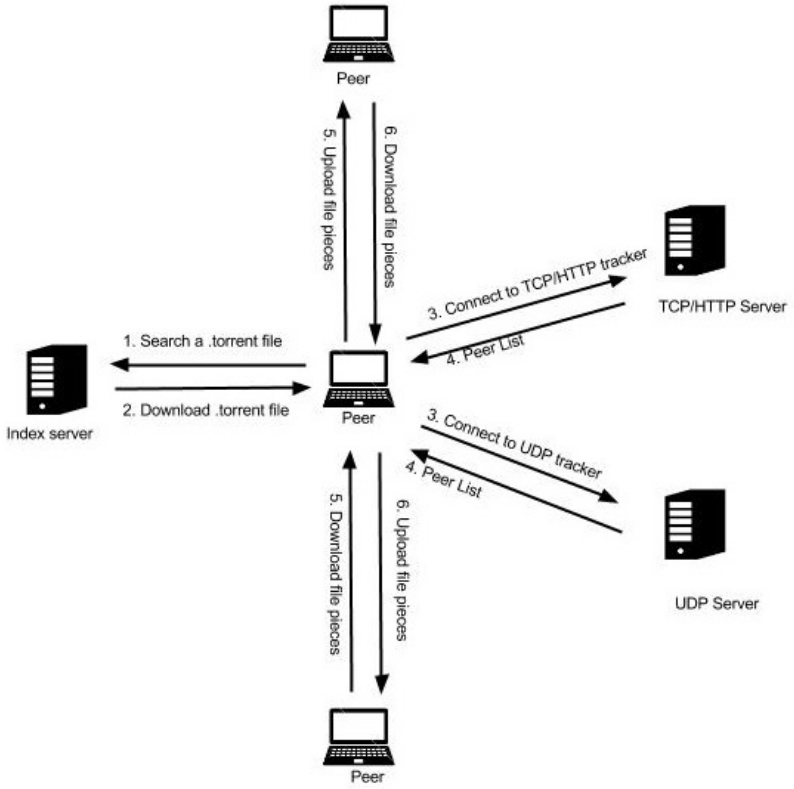


Figure 2.2: BitTorrent architecture

2.3 BEncoding

An ordinary HTTP Tracker responds to an announce request in an encoded format. The so called BEncoding offers the file sharing system BitTorrent to store and transmit data in an endian unspecific way, which is important for cross-platform communication. Torrent meta-data files take advantage of the same encoding mechanism. Bencoding allows the encoding of four different data types:

BEncoded strings: Strings are encoded without a beginning or ending delimiter and have always the following format: <*length of string*>:<*string*>. For example *5:world* represents the string *world*.

BEncoded integers: Integers encoded in BEncoding format contain a starting and an ending delimiter, negative values are allowed and have the format: i<*integer*>e. The encoding of the integer *54* would be encoded with string *i54e*.

BEncoded lists: The encoding of a list follows the format: l<*list elements in bencoded string format*>e, the character 'l' is the starting delimiter and the 'e' is used as an ending delimiter. An example for a BEncoded list is the sequence *l5:plant5:ocean4:snowe*, it is the representation of the list entries *plant, ocean, snow*.

BEncoded dictionaries: The final data type is the dictionary, it contains for each key a value. An initial 'd' and a trailing 'e' are used as delimiters. The dictionary values are BEncoded types as integers, strings, lists or even other dictionaries. The key must store a string value. The following encoding *d5:plant5:green:5ocean4:blue4:snow5:white* are dictionary entries $plant \Rightarrow green, ocean \Rightarrow blue, snow \Rightarrow white$.

2.4 Gathering Peer Information from Trackers

A typical BitTorrent client application as Vuze [7] or uTorrent [8] only needs the information from a .torrent file to obtain all necessary parameters to connect to other peers. The user allows the client application by downloading .torrent files to automatically connect to available trackers. The final communication to other peers is running in the background. There are two types of trackers in BitTorrent networks, the more common HTTP trackers and the UDP trackers, they are using different connection protocols and a different request and response format. In the following part of this chapter we will focus on the structure of announce requests and disregard scrape requests, since peer identification can only be obtained by announcing to a tracker.

2.4.1 HTTP/TCP Trackers

The more frequently used type of tracker is offering a HTTP/HTTPS service and responds to HTTP GET requests. A request for a peer list needs a specific structure and must include some information from the meta-data file to identify the particular torrent and to allow the tracker to update the statistics. The tracker responds to this request with a plain text format containing a bencoded dictionary.

Encoding the HTTP GET request URL to retrieve peer information includes the following steps:

1. The base format of a HTTP GET request is the tracker announce URL as encoded in the .torrent file. An example for a typical announce address is **http://tracker.com:80/announce**.

2. In a second step, a specific key called info hash needs to be extracted for this torrent from the meta-data file. The info hash is a unique fingerprint for each torrent. The content of all torrents are split during the transfer process between peers. Each piece has its own hash which is compared to the info hash in the torrent. The hash value has a 20-byte SHA1 format and needs to be converted to an URL friendly format to be usable in a HTTP GET message. As a result, all bytes not in the set of [[0-9], [a-z], [A-Z], [.], [-], [ ]] must be encoded in "%nn" format, with "nn" as the

hexadecimal representation of the bytes. The following 20-byte hash

x12\x34\x56\x78\x9a\xbc\xde\xf1\x23\x45\x67\x89\xab\xcd\xef\x12\ x34\x56\x78\x9

needs to be converted to this form:

%124Vx%9A%BC%DE%F1%23Eg%89%AB%CD%EF%124Vx%9A

3. In a final step, the converted info hash value is added to the announce URL using basic Common Gateway Interface(CGI) methods. The resulting announce URL looks like this:

   http://tracker.com:80/announce?info_hash=%124Vx%9A%BC%DE%F1 %23Eg%89%AB%CD%EF%124Vx%9A

A HTTP tracker responds to such a GET request message in a plain text document consisting of a BEncoded dictionary with either a single key, representing a failure message, or the following keys:

interval: An interval in seconds, that a client should wait for the next request.

tracker_id: A string the client should use for further requests.

complete: The number of other peers seeding the complete data (i.e. seeders).

incomplete: The number of non or partly seeding peers (i.e. leechers).

peers: The value for this key is another list of dictionaries with the following keys:

peer id: The ID, chosen by the client.

ip: The IP address.

port: The used port number.

Additionally, other optional, less important parameters are included.

A tracker responds to a HTTP request with a message containing not more than 50 different, randomly chosen peers. According to the BitTorrent protocol, such a number should be sufficient for a regular use of BitTorrent [3].

2.4.2 UDP Trackers

There are less trackers supporting the User Datagram Protocol (UDP) than HTTP. Until now, there are only a few tracker implementations supporting the communication over UDP. Compared to HTTP, UDP allows to reduce the necessary bandwidth by more than 50% [9]. UDP is a stateless connection and does not limit the number of open TCP connections a server or router can handle. Nevertheless, UDP is an unreliable protocol, there

is no guarantee packets reach their destination and lost packets need to be retransmitted manually.

Connection request:

Whether a client wants to send a scrape or an announce message to a UDP tracker, a connection ID needs to be obtained before. The following steps explain how it has to be done. In a second step, information about the torrent are sent to the tracker.

1. Randomly choosing a transaction ID(connection_id).
2. Creating a byte packet according to the following format:

Offset	**Size**	**Name**	**Value**
0	64-bit integer	connection_id	0x41727101980
8	32-bit integer	action	0 (=connect)
12	32-bit integer	transaction_id	(randomly chosen)

3. Extracting the tracker address and the port from the meta file and sending the packet to the tracker.

Connection response:

The tracker responds in a similar format including the clients transaction_id and a new generated connection_id. When receiving a connection response, these steps should be followed:

1. Receiving the packet, it should have the following format:

Offset	**Size**	**Name**	**Value**
0	32-bit integer	action	0 (=connect)
8	32-bit integer	transaction_id	(chosen in request)
12	64-bit integer	connection_id	(chosen by tracker)

2. Checking whether at least 16 bytes are included.
3. Comparing the transation_id with the one chosen in the connection request.
4. Checking whether the value of the action field is equal to zero.
5. Are all the conditions in the previous steps fulfilled, storing the connectio_id for further use, but this ID will be valid for not more than two minutes.

Announce request:

The connection response we obtain contains a new connection_id which allows us to send to the tracker an announce request by guiding through these steps:

1. Choosing a new randomly generated transaction_id.
2. Filling an announce packet according to the rules below. A proper BitTorrent client has all needed information for a request.

Offset	**Size**	**Name**	**Value**
0	64-bit integer	connection_id	(randomly chosen)
8	32-bit integer	action	1 (=announce)
12	32-bit integer	transaction_id	0
16	20-byte string	info_hash	(.torrent info hash)
36	20-byte string	peer id	(ID of the peer)
56	64-bit integer	downloaded	(bytes downloaded)
64	64-bit integer	left	(bytes left)
72	64-bit integer	uploaded	(bytes uploaded)
80	32-bit integer	event	0 (=default)
84	32-bit integer	IP address	0 (=default)
88	32-bit integer	key	(unique client key)
92	32-bit integer	number wanted	1 (=default)
96	16-bit integer	port	(port of the peer)

3. Sending the packet to the tracker.

Announce response:

As far as there are any peers available, the announce response contains at the end a list of IP numbers followed by the associated port number.

1. Receiving packet of the format below.

Offset	**Size**	**Name**	**Value**
0	32-bit integer	action	1 (=announce)
4	32-bit integer	transaction_id	(chosen in request)
8	32-bit integer	interval	(waiting interval)
12	32-bit integer	leechers	(number of leecher)
16	32-bit integer	seeders	(number of seeder)
20+6*n	32-bit integer	IP address	(list of IP addresses)
24+6*n	16-bit integer	port	(List of port numbers)

2. Checking whether the packet is at least 20 bytes.
3. Comparing the transaction_id with the one chosen by the client.
4. Checking whether the value for action is set to 1.
5. Remembering to wait for the next request as long as indicated in the interval value.

Although, the number of wanted peer IP addresses can be specified, the tracker returns not more than 50.

2.5 Related Work

BitTorrent has a great impact on the P2P history. Through modelling, simulation and experimentation, people have studied the BitTorrent protocol over a long time. In this part, an overview on already existing and especially meaningful research topics will be given.
An analysis over eight months trace has been done by Poulwelse, et al. [10]. It offers a detailed analysis about four performance measurements of the BitTorrent network, as integrity, flash crowd influence, availability and performance of the system. They noticed that only a few peers tend to stay in the system after they have finished downloading. They exposed as well that peer arrival and departure in BitTorrent networks is not describable with a Poisson distribution, which has been assumed in modelling work.
Zhang et al. tried in a similar approach to describe the entire public BitTorrent system. They developed a high-performance crawler that concurrently crawls thousands of trackers. In the end, they crawled five of the most popular torrent index sites over a period of nine months. Based on their findings, the popularity of BitTorrent content is sensitive to its age and distribution of Users on different index sites is disproportional [11].
In contrast to the work of Poulwelse et al. and Zhang et al., this thesis will focus on a shorter time period and more on the behaviour of BitTorrent users regarding different torrent content.
In another more theoretical approach, Guo et al. [12] used mathematical modelling and equations that gave a good estimate for the lifespan of a particular torrent regarding the number of successful peers and the number of peers that failed due to a lack of seeds. They showed that the performance of small torrents fluctuated widely.
Pitatek et al. provided an interesting study about direct and indirect peer monitoring techniques and gave empirical evidence approving that at least some law enforcement agencies are using the indirect monitoring [13]. Indirect monitoring is the same approach as the implementation in this work uses. Further, they showed that IP blacklists are absolutely ineffective with current identification techniques.

Chapter 3

Solution Design and Implementation

This chapter primarily reveals the application concept and gives explanations for selected design choices. The first section will outline the design specifications and gives answers on how the solution tries to provide the main requirements. The subsection 3.2 explains the collaboration of the components and their structure. The next subsections focus on the decoding of tracker addresses, the configuration of starting parameters, and explains the implemented logging mechanism.

3.1 Design Specifications

According to the main goal of this thesis, which is acquiring a set of data about peer behaviour from trackers, an application has to be designed that provides the requirements flexibility, extensibility, good documentation and deployment.

- Flexibility must be provided regarding the application's domain. There is a huge number of different tracker implementations connected to the global BitTorrent network. Hence, tracker behaviour may be unpredictable and hardly reproducible. Different trackers need to be announced in different intervals and the time to receive a response varies.
  This tool approaches this issue by providing as many reasonable starting configuration parameters as possible, in addition, sending announce requests for single or multiple torrents is supported. A further part of this chapter will explain this approach in detail.

- Extensibility, because BitTorrent offers beside the already discussed topics a wide range of other functionalities. For example instead of downloading .torrent files manually, magnet-links [14] offer a service to acquire the meta-data information automatically. It is assumed that the BitTorrent protocol will be extended or changed in the future. Any newly designed tool related to BitTorrent has to take this fact into consideration.

By clearly separating the work area of the application, as conceptually shown in Figure 3.1, proper extensions as adding new functionalities should be provided.

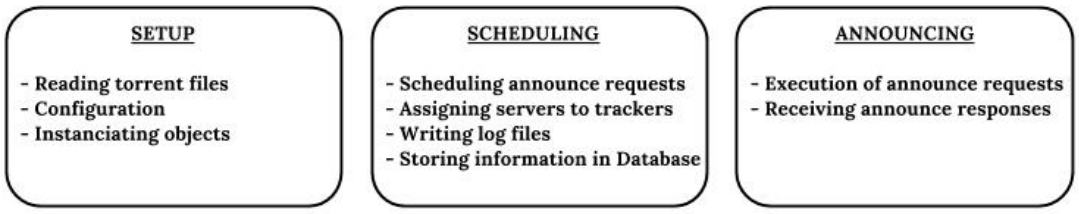


Figure 3.1: Separation of different work areas

- Good Documentation, because regarding the extensibility requirement and the resulting code modification work, an easily readable and quickly comprehensible documentation is required. All code files have to be properly documented and further serves this thesis as another approach to explain and illustrate the application's functionalities.
- Due to the fact that acquiring a huge amount of data from a tracker needs several announce request steps, it is possible that a tracker blocks a client. Deployment of the application is an important aspect. Splitting the announce requests to a wide range of different servers avoid them from being suspended. But still is a flexible distribution a crucial aspect of this approach.

3.2 Architecture Design

The global architecture has been split into three major components, Master Server, Slave Server and Database. The first and most important is the Master Server which acts as a primary component. In contrast to the Slave Servers is the Master Server not directly connected to the trackers. Since one or ideally several distributed Slave Servers are sending announce requests to the trackers, banning will not be an issue due to too many requests. As a final component acts the Database to perform the task of storing the announce results. Figure 3.2 gives an overview of the deployment of these components.
The Master Server communicates with the Database Server using Java Database Connectivity(JDBC). JDBC consists of a set of interfaces and classes which allows the application to connect to a database. The Master Server sends the necessary parameters like tracker addresses and the specific torrent info hashes to to the Slave Servers using Java Remote Method Invocation(RMI) which supports direct transfer of serialized objects. Slave Servers are using either a HTTP connection to send announce requests to HTTP trackers

or equivalently UDP for UDP trackers. These major components are organised in two Java packages, one package includes the Master Server and the Database components, the second package the Slave Server. The following subsections of this chapter will have a closer look at each of these different components.

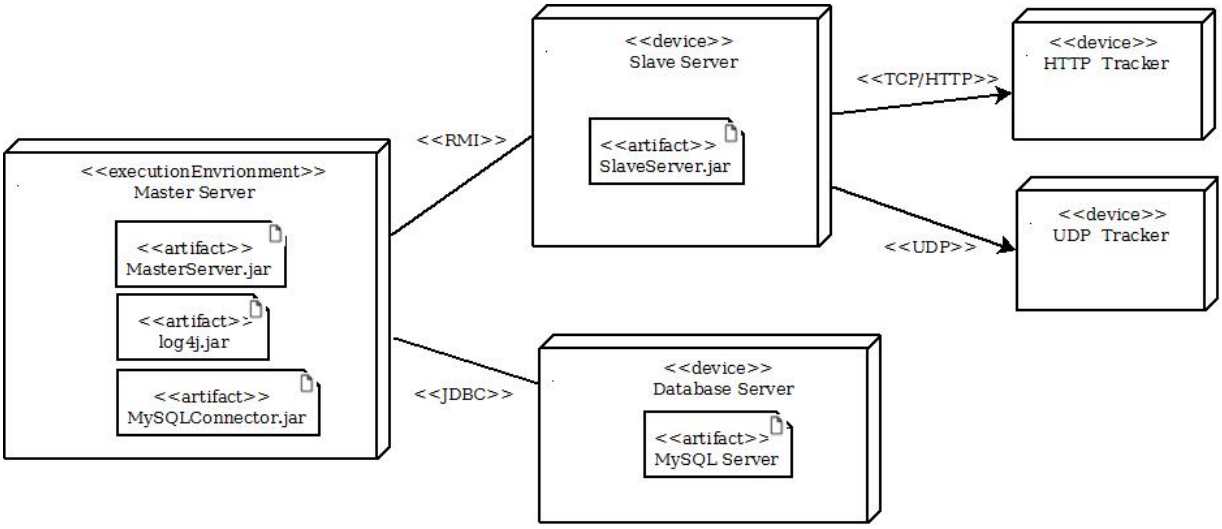


Figure 3.2: Application deployment

3.2.1 Master Server

The Master Server has been designed by separating four major tasks to achieve the main objectives. Figure 3.3 shows the class diagram of the Master Server. The first one is the *Setup* class, it extracts all information from a configuration file, connects to the Database Server and creates the tables. It gets from the configuration file all available Slave Servers, creates instances and stores them in a list. In a next step, the*Setup* class reads all .torrent files from a folder and accordingly creates for each of them an instance of a *Torrent* object. As a next important task decodes the *Setup* class for each .torrent file all included trackers and creates instances for each of these trackers. The *Torrent* objects are stored in a list, each entry contains another list of *Tracker* objects. As a class for additional encoding and decoding algorithms serves the abstract class *Helperfunctions*.
The second component of the Master Server is the scheduling. The detailed mechanism will be explained in a next subsection. The *Scheduler* class receives the lists of *Torrent* objects including their trackers and instances of all available Slave Servers. The class keeps track of the current announce interval and a timer mechanism takes care of the correctly given announce orders. A queue storing all Slave Server instances helps the scheduler to assign slaves to trackers in a reasonable order.
A third component is the combination of tracker and torrent objects. Torrents keep track of additionally needed announce requests for each of their trackers, which allows the scheduler to decide whether another Slave Server has to be assigned or the status can be changed to *done* for this announce interval. The detailed mechanism is shown in the next chapter as well.

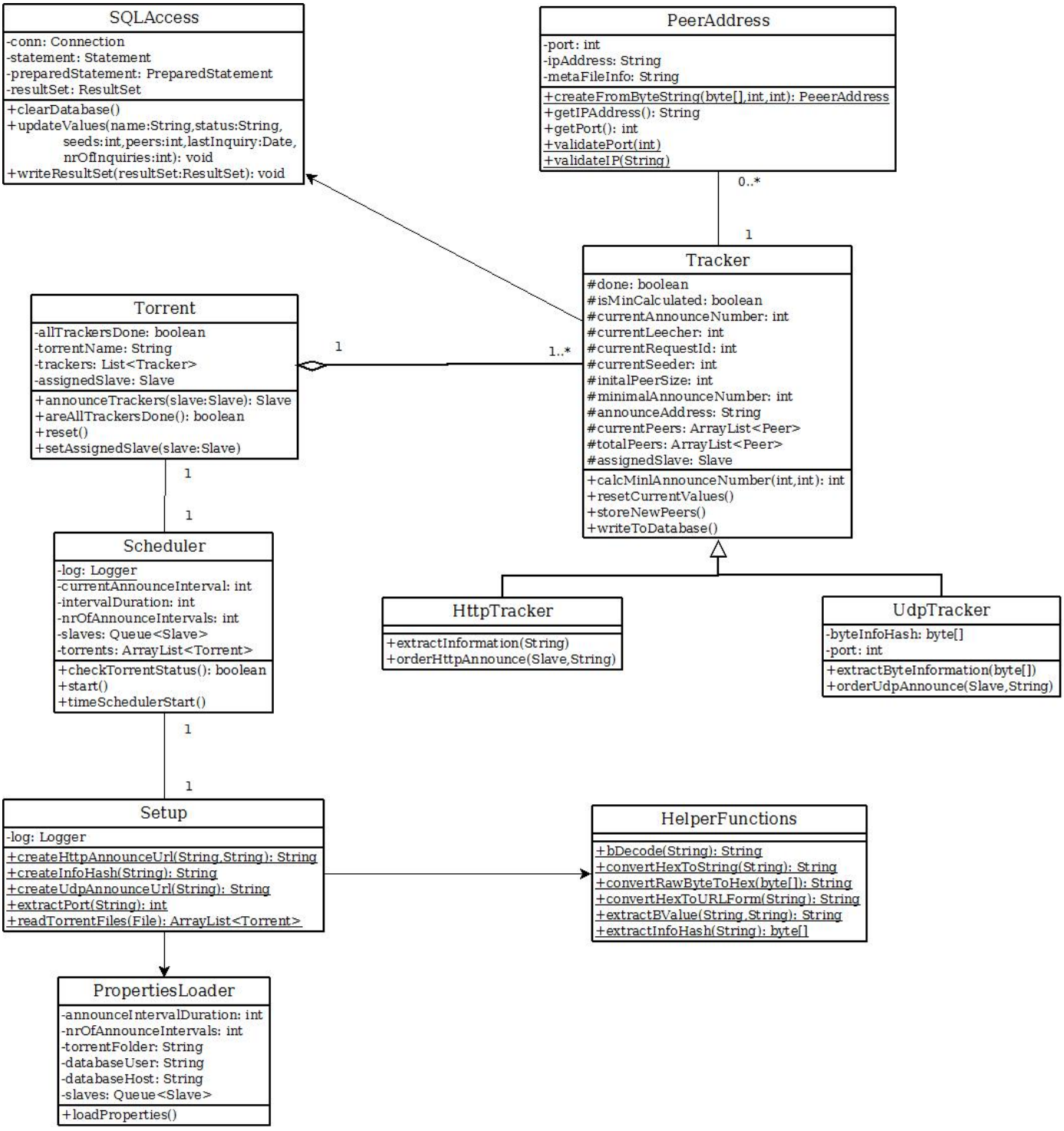


Figure 3.3: Class diagram Master Server

PeerAddress stores the IP address and the port number for a peer and the classes BDecoder and *BValue* are necessary to convert the BEncoded values from tracker responses and meta-data files into a readable format . As a final component serves the class *SQLAccess* which contains all methods to create and insert parameters into the actual database which is located on a MySQL server.

3.2.2 Slave Server

The Slave Servers have been designed as simple as possible. Their main objective is to send announce requests to the trackers and to receive the responses. All encoding and decoding work, beside building UDP byte messages, reside on the Master Server side. The responses received from the trackers are forwarded in an encoded form back to the Master Server. In addition to the classes supporting the RMI communication, there is only an announce class, which has the function to receive orders including all necessary parameters to send an announce message to a given tracker. Two functions are first combining the parameters to either a UDP or a HTTP announce message and forward them in a second step. Figure 3.4 shows the class diagram of a Slave Server. The *HelperFunctions* class contains some methods to assist the composing of the UDP byte announce message.

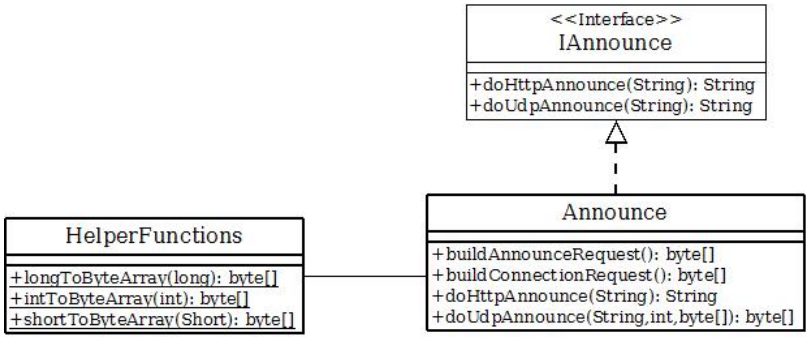


Figure 3.4: Class diagram Slave Server

3.2.3 Database

The Database Server is only connected to the Master Server. It is a MySQL server containing one database with two tables. One of them is storing all information about a particular announce interval, as for example the tracker and the torrent name or information about connected peer numbers. Another table shows for each of the announce requests all IP addresses and port numbers. Timestamp stores the time and the date of an announce procedure. Figure 3.5 shows the schema of the two tables.

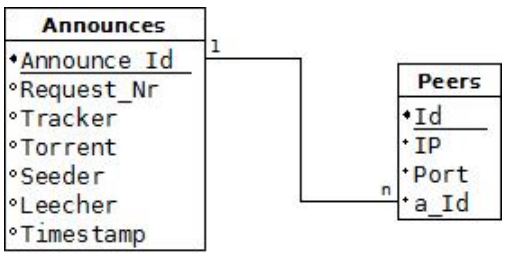


Figure 3.5: Database schema

Announce_Id in the Announces table and *Id* in the Peers table are serving as a primary keys. The field *a_id* is a foreign key in the peers table and identifies a particular announce request.

3.2.4 Scheduling

Another part of the design is the scheduling. The user needs be able to choose the interval time between acquiring two sets of peers from a tracker. Additionally, the user must be able to define how many times in total all peers are collected from a tracker and at which time these acquiring procedures start. The fact that only 50 peers are received in each announce response indicates that multiple requests need to be sent concurrently if a tracker's peer list contains more than 50 peers. All announce messages would need to be received by the tracker at the exact same time, which is technically impossible. Peers are connecting and leaving trackers in undetermined time intervals, this circumstance makes an absolutely accurate and error-free set of peers for a point in time impossible. Nevertheless, distributing the announce requests on different instances of Slave Servers and trying to send them as simultaneously as possible reduces this lack of accuracy. The formula

$$\sum_{i=0}^{p-1} \frac{p - (i \mod 50)}{p - i}$$

to approach the Coupon Collector Problem [15] has been used to calculate the needed number of announce requests to receive a descent number of peers. In this formula stands 'p' for the total number of peers connected to the tracker. Chapter 4 will show and try to analyse the results obtained using this approach. Based on this formula, for a set of 100 peers a minimum of 7 announce requests is needed and accordingly 500 peers need a minimum of 64 requests. The problem arises if the tracker's network consists of larger numbers of peers. For example need more than 1000 peer already more than 150 theoretically concurrent announce messages. Depending on the tracker and the results, receiving a response for an announce message takes a specific time. It is expected that sending 150 announce messages in a row would not only adulterate the real peer set due to the long announce period, it would affect the users choice in interval duration as well. Local tests with one Slave Server have shown that 150 announce requests take approximately 40 minutes The exact duration of announce periods with large peer numbers using this formula will be evaluated in chapter 4.
In the next part is an overview of the design regarding the assignment of Slave Servers to trackers given. Due to previously explained reasons, multiple announce messages must be sent as concurrent as possible. An obvious design solution would be sending all needed announce requests for each tracker in a row. However, most trackers request clients to wait a certain time interval between two announce requests. Moreover would the non-availability of a tracker for a few seconds result in series of failure messages, the outcome of this is an extensive time delay for the whole announce procedure. In previous test announce procedures it has become apparent that especially failed connections are one of the main reasons for response delay. Instead of the previously mentioned scheduler design solution, a different approach has been chosen. The pattern shown in 3.6 is simplified but typical possible initial situation for a start of a new acquiring procedure using this approach.
There are four Slave Servers available and they need to send announce messages to six different trackers. All these trackers were extracted from three different BitTorrent metadata files. This means in this example, trackers T1, T2 and T3 are aware of all peers downloading right now the actual file content from .torrent A. Accordingly has .torrent

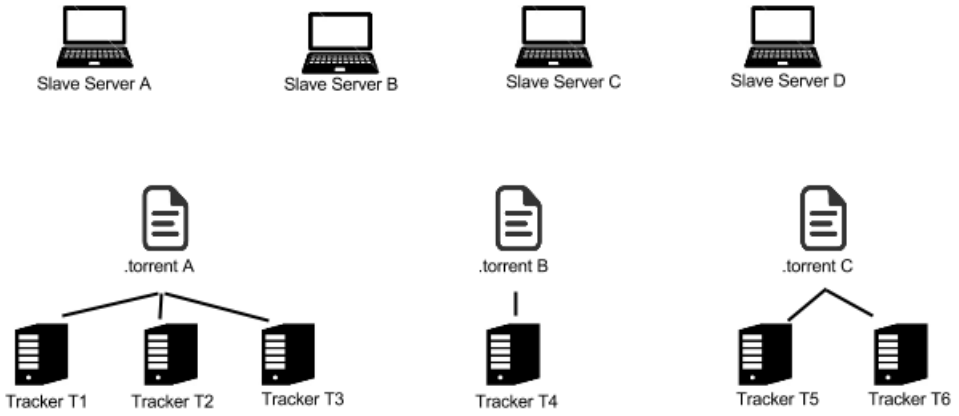


Figure 3.6: Initial situation

B only one tracker and .torrent C two trackers that keep track of peers downloading a specific file. Regarding the design, it was important to assign to each Slave Server in a reasonable order trackers. Another aspect that had to be taken into account was that a Slave Server is not able to send announce messages to the same tracker in quick succession. To assure this condition, all slaves have been stored in a Queue data structure. As far as not the same number of Slave Servers and torrent meta-data files are available, it is nearly impossible to assign a slave to the same tracker multiple times in a row. For a better overview, the assignment procedure for a single .torrent file is explained in a flowchart on Figure 3.7.
In this example, as soon as the tool has to start gathering all peers connected to to the trackers, the following assignments will be made: Slave Server A will be assigned to .torrent A, Slave Server B will be assigned to .torrent B and Slave Server C will be assigned to .torrent C. Slave Server C remains in the queue and will be assigned as soon as a new slave is needed. Each slave assigned to a .torrent sends announce messages to all the connected trackers. That means, Slave A will send to trackers A, B and C announce messages. As soon as all responses for this particular .torrent are received, the Slave Server object is returned to the queue. Once all .torrent objects have ordered announce messages to all of their trackers, the scheduler checks whether additional announce requests are needed for a tracker in one of the .torrents. If there are still requests needed to obtain a decent number of peers, the scheduler assigns to each .torrent that contains one or more of these trackers a new slave. This procedure goes on until all .torrents report that they do not contain any trackers which need to be announced another time. As soon as a new announce interval starts, the whole procedure restarts all over again.

3.3 Implementation

This chapter will introduce the ideas and key concepts of the implementation. The first two subsections will focus on the decoding of BEncoded values and the transformation into easily usable format. Next, the configuration and logging mechanism will be explained.

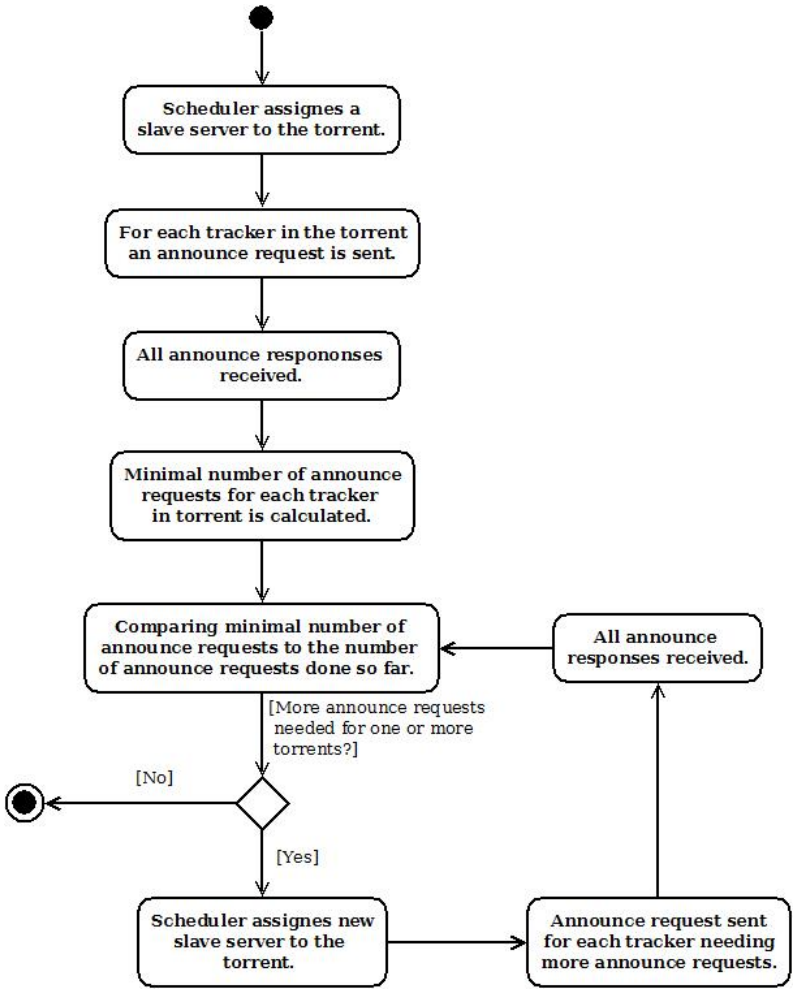


Figure 3.7: Server assignment

3.3.1 Conversion of BEncoded Values

Torrent meta-data files and HTTP Tracker responses are generally encoded in a BEncoded format. This format is less efficient than pure binary encoding as used in UDP tracker implementations, but it is unaffected by endianness, which assures cross-platform communication. Although, BEncoding is an easily readable information representation, handling and processing this encoding can become exhausting. Due to this fact, another way of storing announce responses has been chosen. Especially regarding the extensibility of the tool, a different encoding seems to be a good choice. BValues are used in other BitTorrent related implementations, but are still not an official way to translate and store BEncoded values. Figure 3.8 shows the transformation of a BEncoded value to a BValue format using the *bDecode* method.

complete: Number of peers who have completely downloaded the file.

downloaded: Number of peers downloaded the file and share the file (i.e. seeders).

incomplete: The number of non-seeding peers (i.e. leecher).

peers: The value for this key is another list of dictionaries.

interval: The number of seconds a user should wait for the next announce.

peers: A binary coded IP/port list.

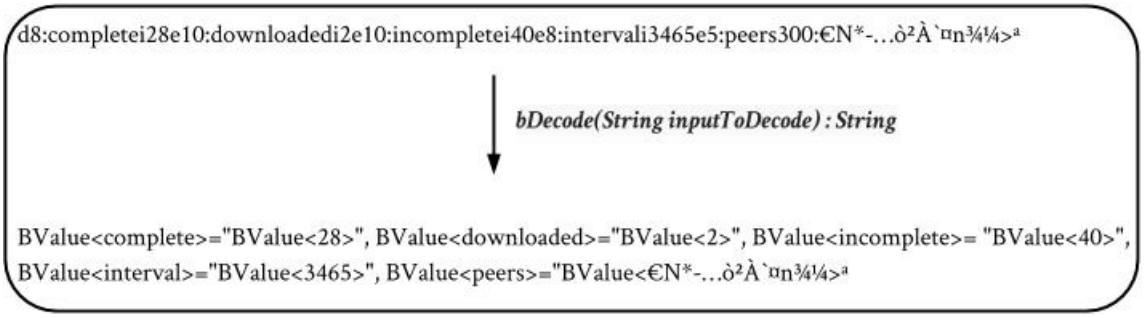


Figure 3.8: Method *bDecode* decodes BEncoded HTTP response to BValue format

The key/value pairs of the BEncoded dictionary are translated into another key/value dictionary by ”BValue” separators which results in a more comfortable, handlebar and even readable format. There is already a vast number of different efficient implementations available on the internet. Due to this fact, it has been decided to adopt and slightly modify Chris Oudman’s elegant solution [16] for the *BDecoder* and *BValue* class.

3.3.2 Decoding of Tracker Addresses

In this subsection a short overview of the implementation will be given and the used methods for the first decoding of a .torrent and the ensuing building of the final announce will be explained. For a tracker using the HTTP protocol and a tracker using the UDP protocol an example will be shown to explain the functionality of each used method in detail. The start of each URL building process is the decoding of the .torrent files.

1) private static ArrayList<Torrent> readTorrents(File torrentFolder)

The constructor of *Setup.java* called at the application start creates a new list of *Torrent.java*. The method *readTorrents* receives as a parameter the file location which has been loaded from the class *PropertiesLoader.java*.

2) privat static String bDecode(String input)

For each file found in *torrentFolder* the method *bDecode* is called to convert the file entry to a string. The method returns a string containing the BEncoded values converted to a dictionary of BValues as explained in subsection 3.3.1. The next step in extracting tracker addresses from the in BValues encoded string has simply be done by searching for suitable substrings. Each available tracker address either starts with

”http://” or ”udp://”, depending on their protocols. Each address which has been found is stored as a string *trackerAddress*. Conversion into BValues allows to easily identify start and end position of the needed substring. In a further step, the address is compared to the already collected tracker addresses. Based on previous observations regarding the time performance of announce requests, the HTTP tracker protocol is chosen if an identical UDP tracker address is available. Some trackers are supporting UDP and HTTP connection. In contrast to that observation, there are different trackers using different but very similar URLs to identify two independent trackers, sometimes it is difficult to distinguish between an already added and a new tracker. An example are the two tracker addresses: http://fr33domtracker.h33t.com:3310/announce and udp://fr33dom.h33t.com:3310/announce. It is nearly impossible to avoid double entries of the same trackers which is actually for the announce process not severe. Before a new tracker is added to the final list of available trackers, the announce URL has to be created. The following method will focus on HTTP trackers, steps in creating an announce URL for a UDP tracker will be shown afterwards.

HTTP: *3) private String createHttpAnnounceUrl(String trackerAddress, String metaFile)*

The meta-data file contains a byte info hash of the .torrent file. The method *createHttpAnnounceUrl* receives the .torrent file in string format. By calling itself the function *createInfoHash* an info hash already converted to a URL friendly format by *convertByteToHexString* is returned. The method *createHTTPAnnounceUrl* adds finally the URL conventional parameters ”?info_hash=” before appending the received info hash. The announce URL is in a next step returned to the *readTorrents* function and stored in the according tracker instance. UDP trackers use different announce addresses to handle a request. Initially, all UDP trackers are stored in the .torrent files in a format similar to udp://tracker.openpittorrent.com:80/. To send a UDP announce request, the address has to be changed using the following two methods.

UDP: *3) private String extractPort(String trackerAddress)*

UDP: *4) String createUdpAnnounceUrl(String trackerAddress)*

By removing the protocol name ”udp://” and the port, the initial tracker address has already been converted in a suitable UDP announce address. The port number as well as the announce URL is stored in an according tracker instance.
After extracting for each torrent all the trackers, the list of torrents is delivered to the *Scheduler.java* class.

3.3.3 Configuration

Choosing the correct starting configuration is an important criterion to acquire relevant information. The file *configuration/configuration.properties* serves as a document to specify the initial starting parameters. The class *PropertiesLoader* extracts all properties defined

by the user as soon as the application has been started. The constructor of the class *Setup* is used to instantiate the values. As an example returns the static method *PropertiesLoader.getTorrentFolder()* the location of the .torrent files. The user must specify each parameter carefully to guarantee a proper functionality. Wrong or missing properties can result in unexpected errors. The following part explains the not trivial configuration parameters.

announceInterval: Defines the interval duration between two acquiring processes of peers for each tracker. According to the design, only the minimum time is specified. Needs the Scheduler more time to complete all announce requests, the time for this particular announce interval will be extended.

nrOfAnnounceIntervals: The total number of times the peer acquiring process must be performed. It defines how many announce entries are inserted into the database.

maxNrOfConcurrentRequests: Maximum number of requests in an announce interval. The calculated number of requests during an interval depends on the initial number of peers. This allows the user to limit the number of requests. A '0' entry indicates unlimited, which means, as many announce requests will be done as calculated before.

database: Defines whether beside the log file a database is used. '1' indicates the use of a database, '0' indicates no database support.

torrentFolder: The location of the .torrent files. The user has to take care that this folder is located in the same directory as the *MasterServer.jar* file or in a subdirectory.

port: Defines the accessible port of a particular slave.

user: Indicates the user of a Slave Server, an arbitrary string, used to register in RMI.

host: Represents the IP number of the Slave Server.

All Slave Server properties must be defined by adding '.X', where X represents the number of the slave. For example port.3 represents the port of slave number 3. The user is advised to enumerate the slaves properly from 1 to the number of slaves which the user intends to use. Missing numbers will cause errors.

3.3.4 Logging

During the implementation and test process an extended debugging mechanism with the ability to write a logging file has been proven beneficial. Error recognition and the possibility to reproduce announce results require more than database records. Due to this demand, a basic logging mechanism has been implemented for the Master Server by

using Apache Log4j [17]. The external library log4j-1.2.17.jar allows the tool to support logging functionalities. The configuration of the mechanism is stored in */configuration/log4j.xml* and *log4j.dtd*.A static instance of the Logger is created by *LogManager.getLogger(Scheduler.class)*. As soon as the application starts, the folder *MasterLog* is created in the same file path as the *MasterServer.jar*. Into *MasterLog* the output logging file *MasterLog.log* is added. The mechanism logs in a first step all successfully added .torrent files and their associated trackers. Additionally, all Slave Servers are listed. The following logging messages give an overview of the announce procedure configuration, including approximate duration, number of announce requests for each tracker and interval duration. The next logging messages contain for each sent announce request the following information:

[response status] - ([current] / [needed]) [tracker] for [torrent] : [Slave]

response status: Indicates whether an announce message was successful. A failed first announce request will result in a second attempt. If a tracker has been unsuccessfully tried to be announced for two times, he will not be considered any more for this announce interval.

current: The current announce request number for this interval.

needed: The needed number of requests as calculated for the first announce or defined as the maximum number, which has been specified in the configuration.

tracker: The address of the tracker.

torrent: The torrent associated to the requested tracker.

slave: The name of the Slave Server which has sent the request.

Another type of logging entry tells the user whenever a Slave Server was not connectable. Further indicates the logging mechanism the end of an announce interval and the end of the announce procedure. The fact that the number of needed announces is calculated based on the number of peers received in the first request can cause irregularities in the number of actually received and the expected number of peers. A higher number of received peers compared to the expected number of peers may indicate that more new peers have announced to the tracker than left during this interval.

Chapter 4

Evaluation

This chapter will show and explain the results of the announce procedures in a real life environment. In a first part, an introduction to the test environment Emanics Lab is given. Emanics Lab allowed to deploy the Master Server and the slaves on different nodes all over Europe. The second section examines the tool's performance regarding the needed announce request time and analyses a factor which has an impact on the delay of the tracker responses. The section 4.3 will evaluate the acquired data from a one week announce process of four torrent files.

4.1 Test Environment

To test the designed application and to acquire the requested data about one week's peer behaviour, a distributed approach had been chosen. An ideal condition to collect the data offered the deployment of several different slaves in a spatial separated environment.

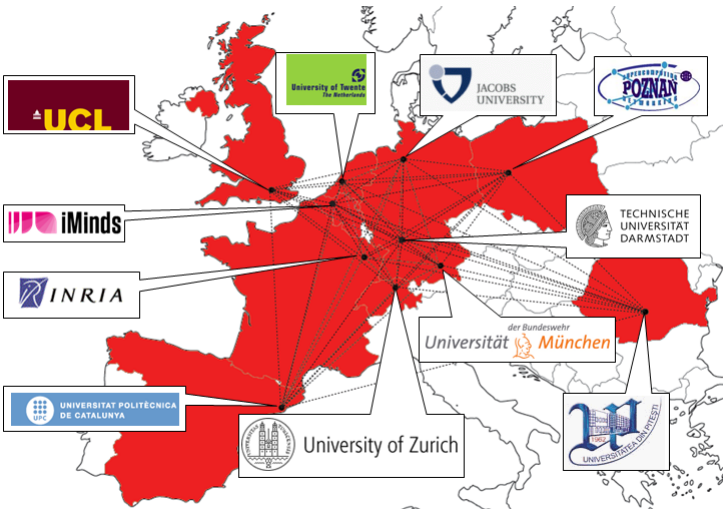


Figure 4.1: Distribution of Emanics Lab sites [18]

Planet Lab is a large collection of servers spread around the world for distributed system research. It is composed of 1090 nodes worldwide. The concept of Planet Lab is to assign to each research project a slice, and each slice has access to a subset of all available nodes. Similarly to Planet Lab is Emanics Lab based on myPLC, the backend management infrastructure of Planet Lab. Emanics Lab consists of 11 sites, which refer to different research institutes and 22 nodes spread all over Europe (Figure 4.1) and the functionality is comparable to Planet Lab, but in a smaller scale.
Emanics Lab provided 20 nodes for the experiments described in sections 4.2 and 4.3. On one node was the Master Server located and on other nodes were Slave Servers running. For the one week acquiring process only 10 nodes have been used at the same time to react on the eventual loss of the Master Server or several slaves at the same time which had caused the failure of the whole experiment. Retaining the other 10 nodes had offered the possibility to react on any unpredictable incident. Furthermore is regarding the design the number of slaves primarily important to prevent trackers from banning a slave, this is not expected to be an issue with 10 or with 20 slaves.

4.2 Scraper Evaluation

Prior tests have already suggested that it is possible to acquire a certain number of peers from different trackers with the designed application and its minimal announce number calculation. But in terms of trackers or torrents with thousands of peers might force the user to extend the announce interval into a length that does not offer any more a satisfying accurateness. An experiment will try to observe this issue by monitoring in particular bigger sized torrent swarms.
Several previous local tests have indicated that the successive and not concurrent announces might delay the announcing process, especially if a tracker is not responding. A number of announce results will try to statistically evaluate whether a correlation between announce duration and failed tracker responses exists.

4.2.1 Experiment Design

Regarding the first part of this experiment, a collection of 12 .torrent files have been gathered from different BitTorrent index sites. All of them have been designated from the index sites as torrents which are shared by more than 2000 peers. A focus has been placed on the number of calculated announces and the resulting announce procedure duration. In this respect, the previously indicated and the actual received number of peers have been compared. A growing aberration of the expected and received number of peers with respect to the announce duration would have indicated that the application's design approach was not appropriate for bigger torrent swarms.
The correlation between announce time and failed connection has been analysed by evaluating 48 announce requests from the dataset received during the one week announce period. A closer explanation of the .torrent files and the results of the evaluation of this experiment will be given in Section 4.3.

4.2.2 Results

According to the BitTorrent index sites, a number of more than 2000 peers have been available for each torrent. In fact, only 8 of 12 torrents and their associated trackers responded with a cumulated number of at least 2000 peers. Table 4.1 gives an overview of the received results. The 12 torrents have been announced as many times as necessary according to the formula mentioned in chapter 3, shown in the column Announces. The number of failed announce requests due to connecting problems to the trackers as well as received peers in relation to the primarily indicated number of peers are shown in the next two columns. The last two columns indicate the duration in seconds for each announce process and the average number of seconds needed for each individual announce response. A particularly striking observation is the fact that the previously indicated number of peers for a particular torrent and the finally received number differ widely. Comparable differences have not been observed for smaller torrent swarms. This suggests that either the actual number of connected peers does not agree with the indicated number of peers from trackers or that the fluctuation of peers is very high. High fluctuation as sole explanation implies that the differences on a percentage basis of these two peer numbers grow in respect to the the duration, which can not be observed.

Torrent	Announces	Failed	Peers	Duration	Duration / Announce
Futurama	2869	51	23762 / 16410	1598	0.557
Graceland	888	13	8049 / 5818	433	0.488
Top Gear	619	13	6499 / 3716	719	1.162
Wilfred US	550	10	5128 / 3734	300	0.545
The Daily Show	422	7	4124 / 3033	215	0.509
Game of Thrones	331	12	5058 / 2184	323	0.976
The Big Bang Theory	329	6	3884 / 2117	225	0.684
King and Maxwell	300	11	3133 / 2170	224	0.747
The Newsroom	267	12	3136 / 1776	236	0.884
Americas Got Talent	189	8	2120 / 1547	207	1.095
Falling Skies	157	13	2548 / 1337	200	1.274
Continuum	86	24	1381 / 693	526	6.116

Table 4.1: Announce results for trackers with more than 1000 peers

It is obvious that the number of sent announce messages for a particular torrent has a significant impact on the duration of its whole announce procedure. But the comparison of the average announce request time suggests that an individual message tends to be strongly influenced by the fraction of failed announce messages. This observation will be evaluated in the now following second part of the Scraper evaluation.
Figure 4.2 shows graphically the relation between the duration and the number of failed

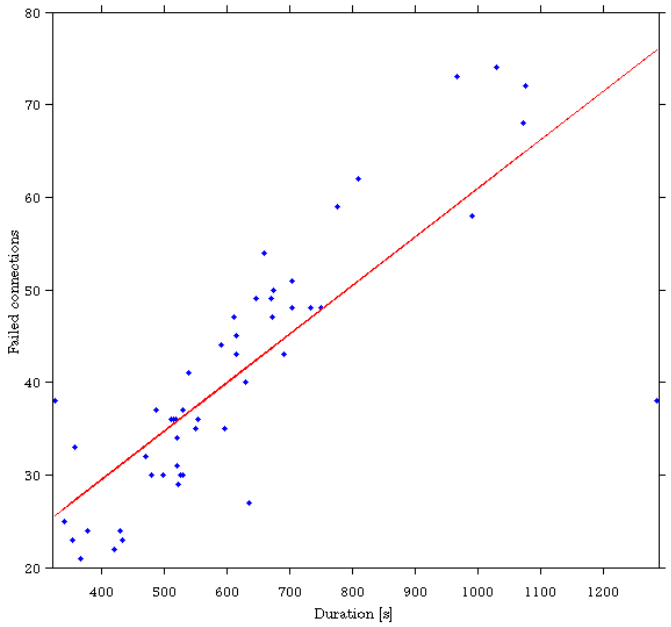


Figure 4.2: Announce procedure duration and failed announce requests

announce messages for 48 requests. As expected, a clear correlation is observable. Thus, it appears that either handling failed announces differently would be beneficial or a completely different design approach would improve the announce duration.

4.3 Evaluation of acquired Data

The next two subsections will show the results of the seven day announce period.

4.3.1 Experiment Design

This experiment analyses the behaviour of peers in a swarm connected to a particular torrent and the peer interaction regarding different torrent files. To collect the data from a real life environment, meaningful starting parameters had to be chosen. Due to this condition, a careful selection of .torrent files was crucial. Choosing BitTorrent content from absolutely independent topics did not offer expectations for any relations between peers. Based on this fact, a collection of different popular TV shows has been chosen. Using BitTorrent to download the newest episode of a TV show is widespread all over the world which promises a decent number of peers. Based on these findings and based on previous experience with this application regarding its performance, four different .torrent files have been selected. Each torrent with a number of approximately 1000 peers.

Defiance S01E11 720p HDTV, .torrent uploaded on 02.07.13

Falling Skies S03E07 HDTV XviD-AFG, .torrent uploaded on 15.07.13

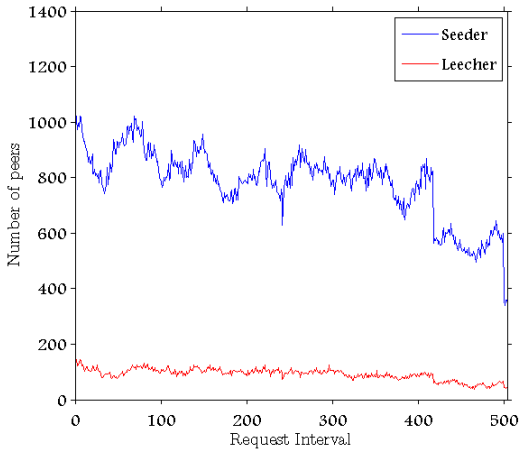
Orange Is The New Black S01E10 720p WEBRip, .torrent uploaded on 14.07.13

Under the Dome S01E04 480p HDTV x264, .torrent uploaded on 16.07.13

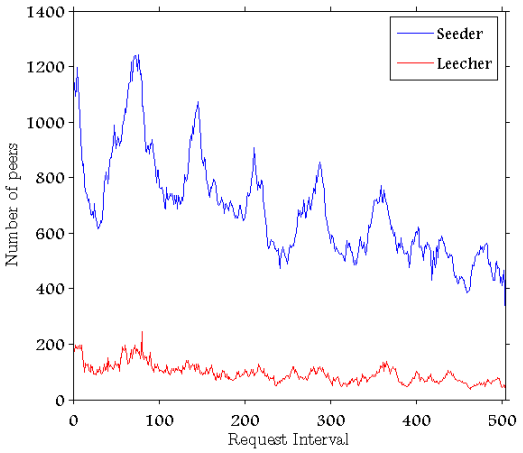
All torrents beside one have been uploaded on a BitTorrent index site a few days before the experiment started. To observe a possible impact of the upload time on the peer number development, the torrent "Defiance S01E11 720p HDTV" has been chosen, which has been uploaded roughly two weeks before the others. All these series are still running on TV, at least in the United States, and are quite successful. The target audience of the shows are similar and therefore particularly suitable for this experiment.
The priority of this experiment is to describe, beside the torrent specific, the individual peer's behaviour based on the four different torrent files as exact as possible. Based on this precision criteria, the intervals between two announce processes had to be chosen as small as possible. It has been tried to acquire an accurate collection of involved peers in intervals of 20 minutes which means, a total of 504 announce procedures for each tracker. Regarding the announce interval duration based on number of trackers and peers, 20 minutes duration seemed to be appropriate to receive all responses in time. Accordingly, the sum of all peers connected to a torrent has been monitored three times an hour during one week.
The experiment started at 19:12 on July 21, 2013. To one Emanics Lab node was the Master Server assigned, while 10 other nodes acted as slaves, handling the announce requests. The evaluation of this experiment is separated into two parts, the first tries to describe the peer behaviour regarding fluctuation of seeder and leecher numbers during the whole week. The second part will focus on the number of peers which were downloading more than one of the given torrents during the experiment duration, it will especially be taken a closer look on when peers downloaded two different torrent files, whether they did it concurrently or with a time lag.

4.3.2 Results

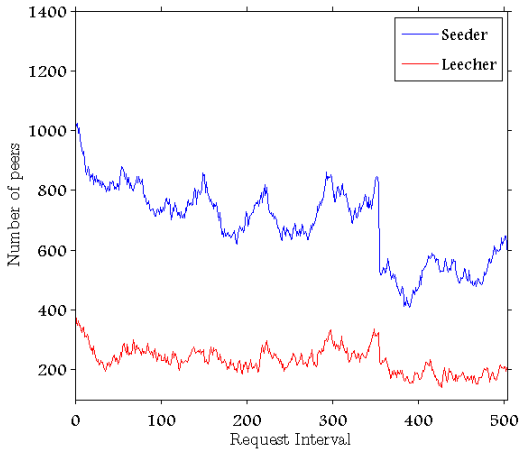
Unfortunately, due to technical problems on a particular Emanics Lab site, one of the slave servers did not respond at all. The remaining 9 nodes were collecting the seeder and leecher numbers from all trackers during a total of 168 hours. Figure 4.3 gives an overview on all four torrent files regarding their peer numbers.
An interesting aspect is that the fractions of peers which were acting as seeder have been for all torrents and during the whole week clearly higher than the fraction of leecher. Another finding is the periodic recurrence of peaks in all four diagrams for the number of seeder, clearly shown in Figure 4.3b. The peaks reappear approximately after 72 announce intervals, which equals to 24 hours. The experiment started at 19:12, shortly before the first occurrence of a peek. The recurrences indicate a periodic user behaviour regarding the use of BitTorrent clients. In addition to this observation about the seeder, the distribution of leecher numbers is less periodic, but still some similarities observable, especially in Figure 4.3c. In contrast is the fluctuation of leecher numbers in Figure 4.3a



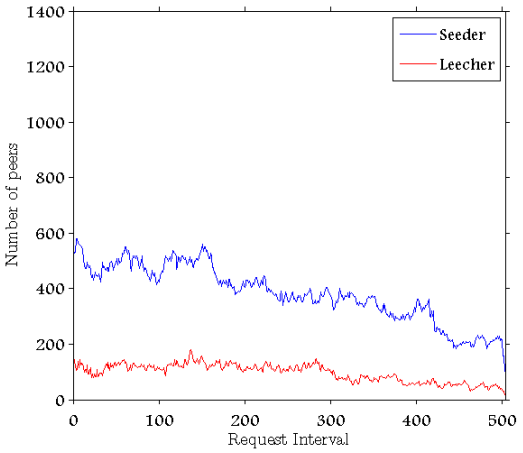
(a) Defiance S01E11 720p HDTV



(b) Falling Skies S03E07 HDTV XviD-AFG



(c) Orange Is The New Black S01E10 720p WEBRip



(d) Under the Dome S01E04 480p HDTV x264

Figure 4.3: Change of peer numbers

less significant compared to its seeder numbers. This might be explained by the fact, that this torrent has already been uploaded two weeks earlier and the demand less intense.
An overall observable similarity for each torrent is the fact that all peer numbers have decreased. At first sight it seems that the fraction of seeder have lost in percentage more during the experiment. But as shown on Table 4.2, the leecher have overall lost for each torrent in percentage more clients. This indicates that the interest in downloading these particular torrent files has decreased over the course of the week. Although, the TV show Defiance has been uploaded around two weeks before the others, there is no indication that it was losing faster peers than the other three. It appears that two weeks of upload time difference had no impact on the peer decrease compared to newer torrents.

The following final part of this chapter will focus on peers which have been downloading two or more of the monitored torrents concurrently in the same announce interval or at

	Defiance			Falling Skies		
	Peers	**Seeder**	**Leecher**	**Peers**	**Seeder**	**Leecher**
1st Interval	1175	1035	140	1338	1148	190
504th Interval	410	367	43	313	276	37
Δ	-65.1%	-64.5%	-69.3%	-76.6%	-75.9%	-80.5%
	Orange is the new Black			**Under the Dome**		
	Peers	**Seeder**	**Leecher**	**Peers**	**Seeder**	**Leecher**
1st Interval	1398	1008	390	700	541	159
504th Interval	781	580	201	116	96	20
Δ	-44.1%	-42.6%	-48.5%	-83.4%	-82.2%	-87.4%

Table 4.2: Change of peer numbers during the experiment

least in any time interval during this week. Due to the fact that these TV shows are popular today and all of them addressed a similar audience, it was expected to discover a descent number of users.

	Falling Skies	**Under the Dome**	**Defiance**	**Orange is the new Black**
Falling Skies	-	92 (0.100%)	4 (0.001%)	460 (0.079%)
Under the Dome	92 (0.051%)	-	2 (0.001%)	194 (0.033%)
Defiance	4 (0.002%)	2 (0.002%)	-	20 (0.003%)
Orange is the new Black	460 (0.256%)	194 (0.212%)	20 (0.005%)	-
Different Peers	179944	91223	335230	585102

Table 4.3: Peers concurrently connected to 2 torrents

Despite these facts, as apparent in Table 4.3, less than one percent of all users who have downloaded one of these torrents, have downloaded at the same time another one as well. The last row presents the total number of different peers which have been monitored downloading a particular torrent content. This means, for example for ”Falling Skies”, nearly 180’000 different IP addresses have been collected during seven days. Regarding the fact that concurrently only 1’400 peers have been connected to all available trackers for ”Falling Skies”, the fluctuation per day must be very high.

	Falling Skies	**Under the Dome**	**Defiance**	**Orange is the new Black**
Falling Skies	-	258 (0.283%)	15 (0.004%)	1215 (0.208%)
Under the Dome	258 (0.143%)	-	10 (0.002%)	411 (0.070%)
Defiance	15 (0.008%)	10 (0.011%)	-	49 (0.008%)
Orange is the new Black	1215 (0.675%)	411 (0.450%)	49 (0.015%)	-
Total Different Peers	179944	91223	335230	585102

Table 4.4: Peers connected to 2 torrents during experiment

The Table 4.4 shows the number of peers which have downloaded two of these torrents during the whole week, either concurrently or time delayed. There are still only a few clients observable which have downloaded two torrent contents. The small number of peers might be explained with the fact that there are several different .torrent files available for one particular show. They differ in resolution, sound quality or even language. Which means, if a client was downloading one of the selected torrents, the user might was also downloading one of the TV shows from another .torrent source that has not been monitored in this experiment.

Chapter 5

Summary, Conclusions and Future Work

5.1 Summary

The overriding purpose of this thesis was to acquire and evaluate a set of data about peer behaviour in a BitTorrent environment. Due to evaluating and validating purposes of BitTorrent applications, a realistic set of input data is beneficial. To accomplish this task it became necessary to reach some prerequisite goals. Determining how information about peer swarms can be collected and how the BitTorrent protocol works in detail. In several iterations of design and implementation a suitable application called Scraper has been designed which is able to collect the necessary data from one or several BitTorrent meta data files. It offers the user different configuration possibilities. An approach has been chosen which allows the Scraper to gather the data distributed from different Slave Servers and collects the overall information in a database coordinated by a centralized Master Server.
The evaluation has been separated into two parts, the first included the evaluation of the Scraper application and identified issues regarding trackers with large peer numbers. Another problem regarding the time delay of tracker responses has been detected and evaluated. The second part analysed the collected data about peer behaviour which has been acquired during one week. It has been focused on the overall peer behaviour and on the concurrent downloading of more than one of the monitored torrent files.

5.2 Conclusions

The evaluation of the Scraper application has shown that smaller numbers of peers can be easily acquired. Although, there is no guarantee that the received peer numbers and the individual IP addresses are correct or the acquired data complete. Regarding torrents with larger peer numbers, an accurate set of peers is difficult and not possible to acquire with the chosen design. Especially, time delay caused by not responding trackers have a

significant impact on the announce procedure.
The monitoring of seeder and leecher numbers of four .torrent files during one week has revealed a periodic behaviour of BitTorrent users. The comparison of peer numbers in this thesis have revealed similar results in all torrent files, in general, the peers collected from all four monitored files have shown similarities in their behaviour.

5.3 Future Work

The work performed in this thesis provides a basis for further future research in different fields.
The observed peer numbers in the one week acquiring process showed a periodic behaviour of BitTorrent users. One possible direction of research is investigating the geographical location of different peers. As mentioned in chapter 4, the monitored TV shows are particularly popular in the United States. A comparison of clients from different locations might reveal more insights into peer behaviour.
Only a small number of peers have been discovered which have downloaded two torrents concurrently. An additional experiment including all collectable .torrent files with the same content may give a more meaningful result.

Bibliography

[1] Cisco visual networking index: Forecast and methodology, 2012-2017. Available: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360_ns827_Networking_Solutions_White_Paper.html, July 2013

[2] Ankur Gupta, Lalit K. Awasthi: Peer-to-Peer Networks and Computation: Current Trends and Future Perspectives, Model Institute of Engineering and Technology, 2012

[3] Bram Cohen: The BitTorrent Protocol Specification. Available: http://www.bittor-rent.org/beps/bep_0003.html, July 2013

[4] BitTorrent Inc.: Trackers. Available: http://www.bittorrent.com/intl/fr/help/manual/appendixa010502, July 2013.

[5] EBU TECHNICAL: Peer-To-Peer Technologies and Servicies. Available: http://tech.ebu.ch/docs/techreports/tr009.pdf, July 2013

[6] Bram Cohen: Incentives Build Robustness in BitTorrent. Available: http://www.ittc.ku.edu/~niehaus/classes/750-s06/documents/BT-description.pdf, July 2013

[7] Vuze Bittorrent Client. Available: http://www.vuze.com, July 2013

[8] uTorrent Available: http://www.utorrent.com, July 2013

[9] Olaf van der Spek: UDP Tracker Protocol Specification. Available: http://www.bittorrent.org/beps/bep_0015.html, July 2013

[10] J.A. Pouwelse, P.Garbacki, D.H.J. Epema, H.J.Sips: The BitTorrent P2P file-sharing system: Measurements and Analysis, Delft University of Technology

[11] Chao Zhang, P. Dhungel, Wu Di, K.W. Ross: Unraveling the BitTorrent Ecosystem, Parallel and Distributed Systems, IEEE Transactions on , vol.22, no.7, pp.1164,1177, July 2011

[12] Guo, Lei, et al.: Measurements, Analysis and Modeling of BitTorrent-like Systems. Berkeley, CA : ACM, 2005. Internet Measurement Conference

[13] Piatek, Michael, Kohno, Tadayoshi and Krishnamurthy, Arvind: Challenges and Directions for Monitoring P2P, 2008, University of Washington

[14] Magnet-URI Project, Available: http://magnet-uri.sourceforge.net/, July 2013

[15] Weiyu Xu, A. Kevin Tang: A Generalized Coupon Collector Problem, Cornell University, 2011

[16] Chris Oudeman, 2011, ubertorrent, Available: http://code.google.com/p/ubertorrent/source/browse/trunk/ubertorrent/src/main/java/com/bitsfromspace/?r=7, July 2013

[17] Apache Software Foundation, Apache Log4j, Available: http://logging.apache.org/log4j, July 2013

[18] EmanicsLab Steering Committee, Emanics Lab, Available: http://www.emanicslab.org/

Abbreviations

C/S	Client / Server
CGI	Common Gateway Interface
JDBC	Java Database Connectivity
HTTP	Hypertext Transfer Protocol
P2P	Peer-To-Peer
RMI	Remote Method Invocation
SQL	Structured Query Language
TCP	Transmission Control Protocol
URL	Uniform Resource Locator
UDP	User Datagram Protocol

Glossary

.torrent A .torrent file is a synonym for a BitTorrent meta-data file. A BitTorrent user downloads the .torrent file in general from a BitTorrent index site which allows the client to connect to trackers.

announce interval The announce interval specifies the time duration between two processes of acquiring all peers from a tracker.

announce URL The announce URL is the specific tracker address which is used to send an announce request.

announcing By announcing or sending an announce request informs a BitTorrent client a tracker that he is connecting to the particular torrent network for first time or that he needs more peers.

BEncoding BEncoding is an encoding used by BitTorrent related applications to store information in meta data files or HTTP tracker responses.

BValue BValues are used in several BitTorrent related applications to store the BEncoded values in a well readable format.

Database The Database is part of the Scraper application, it stores the results received from the Master Server.

info hash Each .torrent file has a unique info hash which ensures the complete, uncorrupted download of a content.

leecher Leecher are a part of a BitTorrent swarm. In contrast to a seeder, uploads and downloads a leecher content of a specific torrent.

Master Server The Master Server is a component of the Scraper tool with the main task to schedule and order the announces.

meta-data file A meta-data file is the same as a .torrent file. A BitTorrent user downloads the meta-data file from a BitTorrent index site which allows the client to connect to trackers.

peer A peer is either a seeder or a leecher. A BitTorrent swarm consists of a given number of peers.

peer-to-peer Peer-to-peer is a network architecture which allows the system to distribute the cost of file transfer and organization on different clients. The result is higher scalability.

RMI Remote Method Invocation is a Java API which has similar functionalities as remote procedure calls. It provides communication between programs written in Java.

scraping Scraping or sending a scrape message is done by a BitTorrent client to receive the currently connected number of peers to a BitTorrent network.

seeder A seeder is a client which shares file pieces with other BitTorrent users. In contrast to a leecher has a seeder already downloaded the complete content.

Slave Server Slave Servers are components of the Scraper application. Slaves are communicating with the trackers and are forwarding the results to the Master Server.

swarm A BitTorrent swarm consists of the total number of peers connected to a tracker regarding a particular torrent.

tracker Trackers store information about one or many BitTorrent networks.

List of Figures

List of Tables

Appendix A

Installation Guidelines

Slave Server

Each Slave Server needs to be defined in the configuration.properties file in the Master.jar. Each instance of Slave.jar has to be started as follows

- *Slave.jar [Slave host] [Slave port]*
  e.g. Slave.jar Scraper 14000

Using exactly the same port, user and host as indicated is crucial to guarantee functionality.

Master Server

Before executing the Master.jar, all the Slaves need to be started. It is important that all entries in the configuration.properties file have been adjusted. The location of the BitTorrent meta data files which need to be monitored must be specified as well in the configuration file. The logging file is created next to the Master.jar. The Master Server can be started with

- *Master.jar*

The application will immediately start with the announce procedure and logs the relevant information to the logging file.

Appendix B

Contents of the CD

Evaluation:

DataEvaluation.zip - Contains all files from the evaluation of the one week peer acquiring process.

ScraperEvaluation.zip - Contains all files from the evaluation of the Scraper tool.

Source:

Master.jar - An executable Master Server file.

Slave.jar - An executable Slave Server file.

Source.jar - Project source file.

Thesis:

Zusfg.pdf - A German version of the abstract.

Presentation.pdf - A copy of the presentation slides.

RelatedWork.zip - Copies of related work papers.

Thesis.pdf - A copy of the written thesis.

Thesis.zip - Source files of the written thesis.

Abstract.pdf - A copy of the abstract.