School of Engineering & Design
Electronic & Computer Engineering

MSc Distributed Computing Systems Engineering

Brunel University

Minimizing Free Riding in BitTorrent Networks with Game Theory

Ansgar Hoffmann

Supervisor: Dr. Maozhen Li

07/2010

A Dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science

School of Engineering & Design
Electronic & Computer Engineering

MSc Distributed Computing Systems Engineering

Brunel University

Minimizing Free Riding in BitTorrent Networks with Game Theory

Student's name: Ansgar Hoffmann

Signature of student:_____

Declaration: *I have read and I understand the MSc dissertation guidelines on plagiarism and cheating, and I certify that this submission fully complies with these guidelines.*

Acknowledgement

I owe my deepest gratitude to my wife Xiaosong who had patience, provided motivation, discussed with me and pushed me to give my best at all times. This thesis would also not have been possible without my family and colleagues who were there to do my work while I took holidays to write up this document. I would also like to thank Matthias Kienle for giving me the freedom to take free time whenever I needed it and Carsten Timm for doing my work during these times. Thanks also to Stephan Pressler, Patrick Cervicek and Rolf Laible for proof-reading and providing improvements. Further thanks to my colleagues Micha Koller and Tobias Langjahr for the discussions about mathematics.

My gratitude also goes my supervisor Dr. Li for unbelievably fast email responses and for not only providing the topic of this thesis but also for the useful suggestions after reading my thesis.

I would also like to thank my grandmother who died during the writing of this thesis. She showed me that with enough determination, one can reach any goal.

Contents

List of Figures

Glossary/Abbreviations

BitTorrent

Bitfield The bitfield is sent to other clients to advertise which chunks a peer has. The field contains as many bits as there are chunks in the torrent. A bit has the value 1 if the peer possesses the chunk and 0 otherwise.

Choke When a peer chokes another peer, then it stops sending data to this other peer.

Chunk A chunk is a piece of the torrent to be downloaded. In BitTorrent, all data is split into chunks of a certain size. Each chunk is identified by an ID. This ID corresponds to the position within the bitfield of a peer.

Downloader A peer that has not yet finished its download. Downloader are sometimes also refered to as leechers but downloader will be the preferred term as leecher is sometimes also used as a synonym for free-rider. To avoid misunderstandings, a downloading peer is downloading and uploading. A free-rider downloads only.

Free-rider A free-rider is a downloading peer that does not contribute any-

thing to the system. Free-riding is understood differently throughout the literature. Please refer to the introductory chapter for a list of references. Within this dissertation, free-riding will be understood as never contributing useful data to the system. The only upload done by a free-rider is the data that is necessary to fulfil the protocol's requirements. According to this defintion, each peer that uploads useful data of any amount to another peer, is not to be considered a free-rider anymore, although it might still draw more resources from the system than it gives.

Interested If a peer is interested in another peer's chunks, then it sends a message to this peer to inform it of its interest. The receiver of the interested message will set the interested flag for the sender and consider unchoking it.

Leecher This term shall not be used within this dissertation as it is understood in different ways and may have a negative conotation to some.

Optimistic Unchoking The mechanism used by a peer to explore the swarm of which he is a part of. It means that the peer unconditionally sends data to a randomly chosen partner to initiate reciprocation.

Peer Any client that is connected to the tracker is called peer.

Torrent-file Also: .torrent-file. A file containing meta information like checksums of the chunks and the tracker's address. If a user wants to publish a file, then he will use a BitTorrent-client to create a torrent-file. The user has to provide some of the data, e.g. the tracker's address. Data like the checksums will be generated by the BitTorrent-client.

Scrape A scrape request is sent by a BitTorrent-client to the tracker. The result is sent back by the tracker. This result contains information about the tracker e.g. the tracker status and the number of connected peers.

Seed A seed is a peer in possession of all chunks. Seeders only contribute to the system while not downloading anything. Seeds are an essential part of BitTorrent. The initial distributor of a torrent will stay in the system as a seed to bootstrap other peers. Afterwards, seeds are often responsible for a substantial portion of the available bandwidth in the system.

Share ratio Most clients display a ratio to tell the user when he has reached a balanced upload/download ratio. Users are told to upload at least as much data as they download. The share ratio has no effect on the download rate of the clients. The ratio is calculated by the client for each torrent.

Swarm A swarm is comprising a number of peers connected to a tracker to share a certain file.

Torrent The data to be downloaded is referred to as a torrent. This data can either consist of files or folders. One torrent is downloaded using the information from a torrent-file.

Tracker A tracker is a server that keeps a list of peers and sends subsets of the list to enquiring peers. Peers register at the tracker. A tracker is responsible for one torrent.

Unchoke Unchoking is the action of a peer to allow another connected peer to request data.

Game Theory

Player Every participant in game theory is called a player.

Nash Equilibrium The Nash equilibrium is a strategy combination, in which no player can gain a higher payoff by a unilateral switch to another strategy.

Dominant Strategy A weakly dominant strategy results in an equal or higher payoff for a player, regardless of what the other players do. A strictly dominant strategy results in a higher payoff.

Payoff (synonym: Utility) The payoff or utility describes the value of the result of playing a strategy combination to a player. Anything that has an influence on the choice of a player has to be modelled into the payoff.

Rationality If a player is rational, then this means that it will always choose the alternative that yields the higher payoff.

Chapter 1

Introduction

1.1 Intention and Scope

This master dissertation will investigate how the existing BitTorrent mechanisms can be improved to mitigate the problem of free-riding.

This introductory chapter will make use of terminology that has not yet been explained. The concepts used in this dissertation will be briefly introduced. Chapter 2 contains more information and a glossary of the technical terms for reference.

BitTorrent is a widely used protocol for file sharing. Currently, it is the most popular protocol for file-sharing because of its high performance. "In Australia, the eDonkey and BitTorrent traffic volumes are particularly extreme with only 14% eDonkey and as much as 73% BitTorrent."[ipo07]. In a system that can be anonymously used, users will try to avoid having to contribute to the system in order to gain more resources. One paper that researched the

free-riding problem within the Gnutella network found, that "70% of Gnutella users share no files, and 90% of the users answer no queries". [AH00]. It can be assumed that the users don't behave differently when they switch to a new technology like BitTorrent. As soon as there is a way to exploit the system with a acceptable effort, it will be done. It has been stated and proven that BitTorrent can be exploited easily by [LMSW06, HP05, JA05, Moo06]. Even an easy to use free-riding client called BitThief [LMSW06] has been implemented, which allows any user to free-ride without any technical knowledge or effort. There exist two different definitions of free-riding in the literature.

The first definition defines free-riding as using more resources than contributing to the system. This can be summed up as *the downloaded amount of data is greater than the sum of the peer's uploaded data*. This definition is given by [JA05, MPM+08, BAS03] as "free riders; that is, they contribute little or nothing."

The second definition is more strict and states that a free-riding peer does not contribute any resources to the system. For BitTorrent, this means that the free-riding peer does not send any data chunks except for the protocol messages to other peers. This definition is the one predominantly used in the literature [LYWM07, LMSW06, QS04, BH05, LLKZ07, MPES08, Teo07, AMCB04] and will be used within this dissertation.

Because the BitTorrent system does not hold any persistent state information on a central server that could be used to ensure fairness. This means that fairness can only be enforced during the time of download. Enforcing complete fairness as postulated in the first definition would harm the system's

performance. Most Internet connections nowadays are asynchronous, meaning that their download rate is higher than their upload rate. To ensure complete fairness would require to limit a peer's download rate to its upload rate. The whole system is designed in a way that peers first download at a high rate and later stay in the system to contribute their resources to the system. But because of the missing central server to store the contribution information, there is no incentive for a peer not to quit the network after its download is finished.

The aforementioned BitThief-client represents a free-rider of the second definition. This dissertation will also follow the latter definition that a free-rider does not contribute any payload data to the system and only sends protocol information. A slow uploader or a peer that quits the network without having uploaded as much as it has downloaded is not considered a free-rider according to this definition.

The system of BitTorrent consists of several parts: A central server called tracker that acts as a peer registry. If requested, the tracker sends peer lists to a peer. This enables the peer to find neighbours. The clients connected to the tracker are the second part of a BitTorrent-system. A webserver for hosting the torrent-files containing the meta-data can also be seen as part of the BitTorrent-system, although it is not a vital part of it and is not even required for BitTorrent to function. It is only one way of obtaining the torrent-files that contain the meta-data. A user could also receive torrent-files by email or FTP. Therefore the webserver will be excluded from the analysis. It will be assumed that all peers are in possession of a valid torrent-file so that they can

connect to the tracker and compare the received chunks against the checksums within the torrent-file. Although the tracker plays a vital role, it will also not be modified. Its function is only to give out lists of peers to clients who need to find new neighbours. The tracker has absolutely no way of knowing whether a certain peer is a free-rider or not because no data transfer is routed through the tracker. The tracker can only limit the size of the peerlist that it gives out and the frequency in which a peer is allowed to obtain a list of new peers. This can help to fight the "Large View Exploit"[SHRY07], which aims at getting the complete list of peers in order to increase the chances of being optimistically unchoked. If a tracker does not allow the peer to request a large list and bans peers for asking in a too high frequency, then the effect of this exploit can be minimized. This is already done in real world systems so it will be assumed that free-riding peers will not violate the limits set by the tracker software. The analysis in this thesis will focus on the peers as they participate in the exchange of data chunks and are able find out whether another peer behaves like a free-rider or not. It will be further assumed that the data sent to others is valid and not corrupt. Only data that can be validated by the peer itself shall be used. This way, no trusted third party has to be introduced.

1.2 Motivation

Although BitTorrent has proven very resilient against cheating so far, cheating occurs anyway. It has not yet resulted in the expected *tragedy of the commons* [Har68], which has been anticipated by [LFSC03, LLSB08, MLLY04, JA05]. But with the increasing ease with which BitTorrent can be exploited, this will

probably change. Free-riding BitTorrent has become very easy since BitThief has been released to the public. As free-riding is used at a wider scale, more and more users do not see why they should contribute their resources to free-riders. No technical knowledge is necessary for the user. If the user can choose between contributing resources and not contributing, contribution has to be rewarded or free-riding has to be punished. In a game theoretical analysis, peers can be seen as rational players. Rational players always choose the strategy that offers the greatest reward for them. In this case, they would not choose to contribute. There may of course be altruism but the majority of users will choose the strategy that promises the best outcome, not caring about others.

Since rational users would always choose to maximize their profit, the rules must provide an environment in which it is the best option for the user to choose a strategy that is also profitable for the system. This is normally achieved by incentives. The incentive mechanism of BitTorrent is called tit-for-tat. Tit-for-tat is a game theoretic strategy that came out as winner in a computer tournament held by Robert Axelrod[Axe84]. A rough translation of tit-for-tat is *equal retaliation*, meaning that the strategy will always retaliate bad behaviour by imitating it in the next round. This way, a cheating opponent can only cheat once. Afterwards, the opponent will face retailation. If the opponent is playing fair again, then the tit-for-tat player will imitate the fair behaviour in the next round. This simple strategy defeated all other strategies that were part of the tournament of Axelrod.

But the BitTorrent incentive-mechanism is not completely tit-for-tat be-

cause it does not include all parts of the system. This is agreed on by several sources [LLSB08, LMSW06, Moo06, LYWM07]. Although BitTorrent provides incentives to contributing peers, there are two situations to which tit-for-tat does not apply. One is a situation in which a seed[1] and a peer interact. Because the seed has no ability to discern free-riders from contributors, it will upload data to any peer. By definition, a seed cannot apply tit-for-tat as it does not need any data chunks. The second situation is when a peer tries to find a new peer by uploading data to a random partner without the need for reciprocation. This is called optimistic unchoking. Those two exceptions from the tit-for-tat scheme can be exploited by free-riders and lead to the situation that free-riders can download a file as fast as or even faster [LMSW06, LYWM07] than users using the original client. The users of the original client also have the disadvantage that they cannot use their resources while contributing. This provides users with a reason to divert from their altruistic strategy and switch to free-riding as well.

This dissertation aims at reducing BitTorrent's vulnerability for free-riding by addressing those situations, in which the tit-for-tat scheme of BitTorrent can't be applied. A suitable modification will be developed. The proposed mechanism should be able to be rolled out while the current mechanism is still in use. The effectiveness of the proposed mechanism shall be studied and analysed for a mixed population, i.e. together with peers that use the current mechanism, as well as in a population of peers that only use the modified mechanism.

[1] A peer that already possesses all parts of the torrent

1.3 Aims and Objectives

The aim of the master dissertation is to develop a novel algorithm to mitigate the free-riding problem of the BitTorrent system. This includes the following objectives:

1. Background Information: Introduce the concepts of BitTorrent and game theory.

2. Survey of Related Work: Review existing approaches to alleviating the BitTorrent free-riding problem and describe their advantages and disadvantages.

3. Alternative Concept: Develop a novel mechanism to tackle the problem of free-riding. Compare the modification to the original BitTorrent algorithm.

4. Simulation Implementation: Implement a simulation of the new algorithm to investigate the effectiveness of alternative concept.

5. Measurement, Analysis and Comparison: Analyse the simulation results in order to verify the suitability of the algorithm modification to tackle the free-riding problem. Identify strengths and possible weaknesses of the novel concept

1.4 Project Management

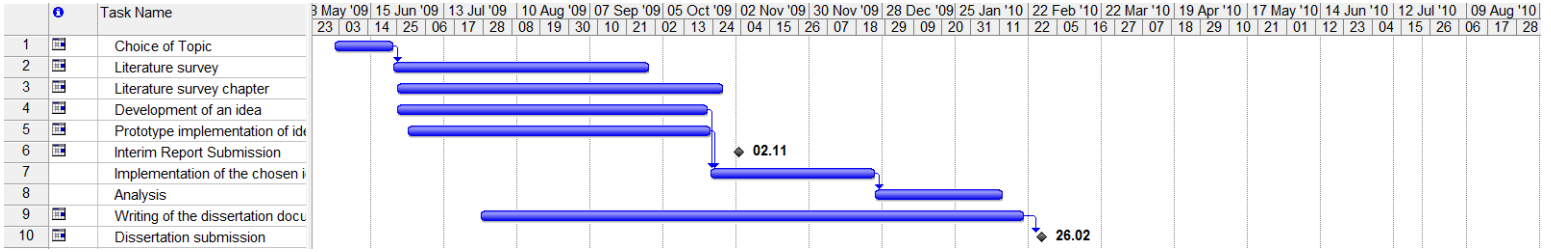


Figure 1.1: Estimated time schedule

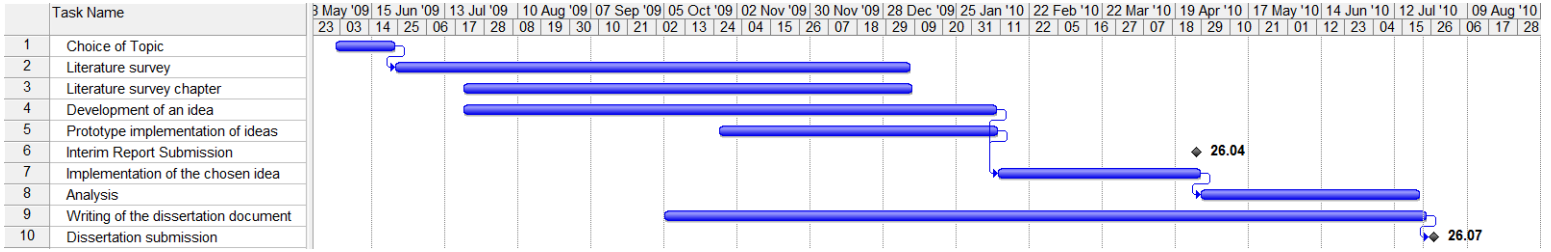


Figure 1.2: Actual time schedule

Figure 1.1 shows the time schedule as it was estimated at the time of the interim report. Figure 1.2 shows the actual time schedule of the project for comparison.

In the original time schedule, the implementation should start in October and results should be available for analysis at the beginning of December. This did not work out quite the way it had been planned due to the fact that a family member had to be taken care of at home and the later death of that person. Hospital visits and day and later home care prevented nearly all progress for about two months.

Another thing to note is that the *Prototypical Implementation of Ideas* did not start together with the *Development of an Idea*. In the original time plan both tasks started simultaneously. An idea did of course not come on day one. It took some time to develop something that could be tested using a prototype implementation.

The schedule assumed that eight hours can be used on weekend days and another four hours spread across the week, totalling in 20 hours a week for the work on the dissertation or simulation runs. Unfortunately, the experience showed that the time after work can't be used effectively. This together with long working hours and moving into another apartment has lead to a delay in submission.

The removal into another apartment delayed the tasks from the beginning of 2010. Writing the dissertation and renovation of the new apartment had to be done in the free time. Because of this, all the following tasks did not have as much time available as planned.

The final deadline at July, 31st can still be met. All work packages have been successfully completed. All Aims and Objectives have been reached.

Chapter 2

BitTorrent and Game Theory

The background chapter is split into two parts. First a section about BitTorrent, second a section about game theory. The content of this chapter is intended to help understand the underlying concepts, technologies, and mechanisms.

2.1 BitTorrent

This background section introduces the BitTorrent protocol, and some of the basic concepts of BitTorrent.

2.1.1 Protocol

This subsection gives a brief overview of how BitTorrent works and how the peers interact during the lifetime of a torrent.

Figure 2.1 shows the creation of a torrent-file by a user who wants to

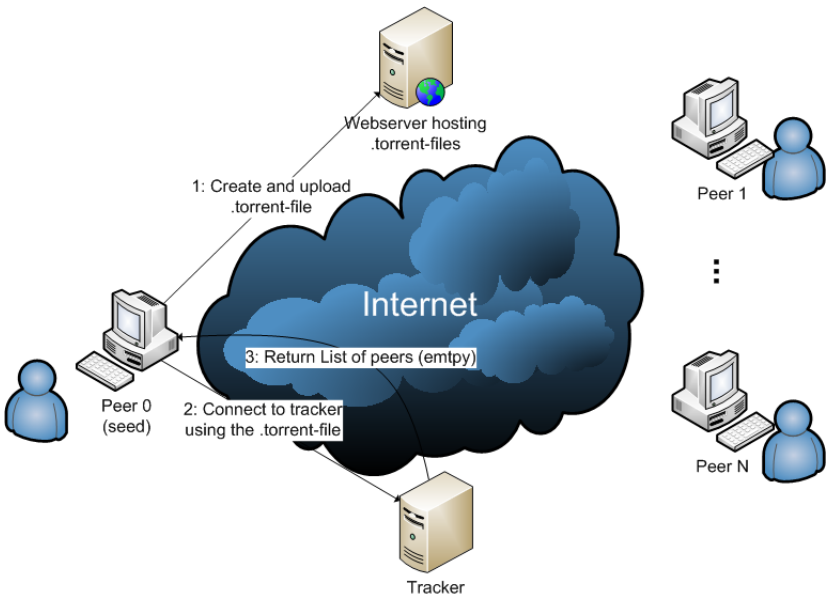


Figure 2.1: Torrent creation and initial connection with the tracker

publish something. A .torrent file is created using a BitTorrent client. A user selects a file that he wants to publish and the client will create a so-called .torrent-file containing metadata. This metadata amongst others consists of a tracker address, the number of chunks and the checksums of the chunks. This file is then published. Publication usually happens by uploading the file to a website that hosts .torrent files and where the file can be found using a search engine. The BitTorrent protocol itself does not incorporate any facility for content search. Searching and hosting of .torrent-files is mostly done with the help of search engines on the webservers that host the torrent files. Torrent files could just as well be exchanged by email, FTP or even other file sharing protocols. After creation of the torrent, the publishing user will open the .torrent-file to connect to the tracker and wait for enquiringn clients, that want to download the torrent. The publisher acts as a seed in order to bootstrap

the process.

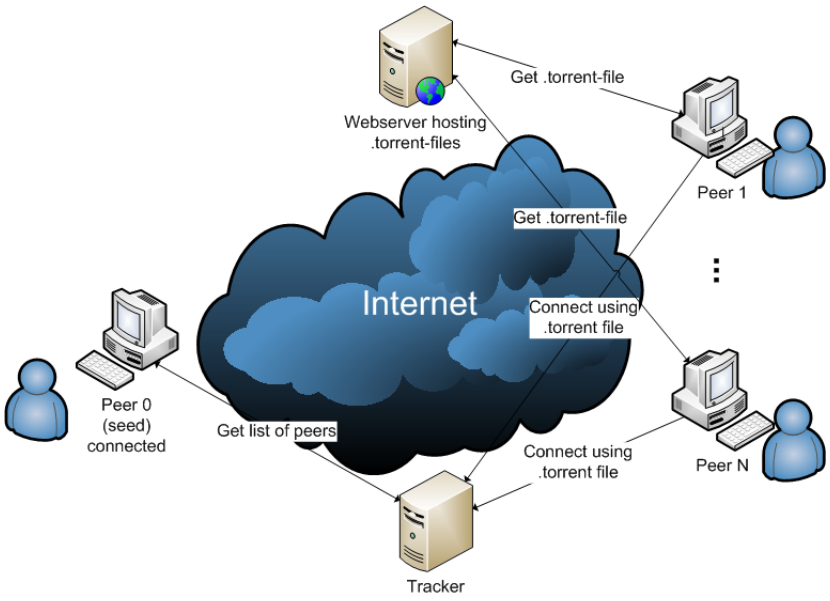


Figure 2.2: Interaction of peers, webserver, and tracker

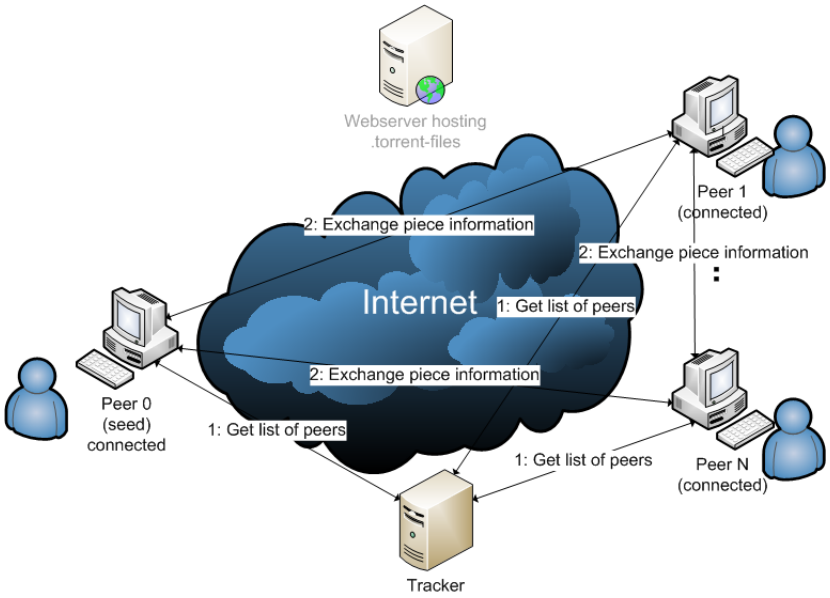


Figure 2.3: Peers obtaining a peer-set and connecting to neighbours

A user who wants to download the torrent obtains the torrent-file from

the webserver and opens it using a BitTorrent-client. The BitTorrent- client parses the .torrent-file and extracts the necessary information to connect to the tracker. The tracker is the central server which is responsible for holding a list of all the connected peers. The client will ask for a list of peers from the tracker using a peer list request. The tracker will respond to this request with a random list of peers. This list is received and stored by the client. The process of obtaining the torrent-file from a webserver and connecting to the tracker can be seen in Figure 2.2. Figure 2.3 shows how the peers obtain a peerset from the tracker and exchange handshake messages with their newly found neighbours. To obtain the peerset, each peer sends a request to the tracker to ask for a list of peers. The tracker answers it by sending a random set of peers. Upon reception of the list, the client iterates through the list and connects to the peers contained in the list. A TCP[1] handshake is initiated with the other peer upon which bitfield messages are exchanged. From this moment on the peer knows which pieces his neighbour does possess and whether he is a seed or another downloader.

Figure 2.4 shows how peers interact after connection and handshake are completed. Peer 1 compares his chunks to the bitfield he received from Peer 0. If he lacks a chunk, then an *INTERESTED*-message is created and sent to the peer 0. *INTERESTED*-messages do not contain a chunk-ID, they merely indicate a general interest in the other one's data. If Peer 1 is selected by Peer 0's choking mechanism (see section 2.1.1), then Peer 0 will send an unchoke-message to Peer 1. After receiving the unchoking message, Peer 1 will ask for the needed chunk by generating and sending requests for subpieces of the

[1] **T**ransmission **C**ontrol **P**rotocol

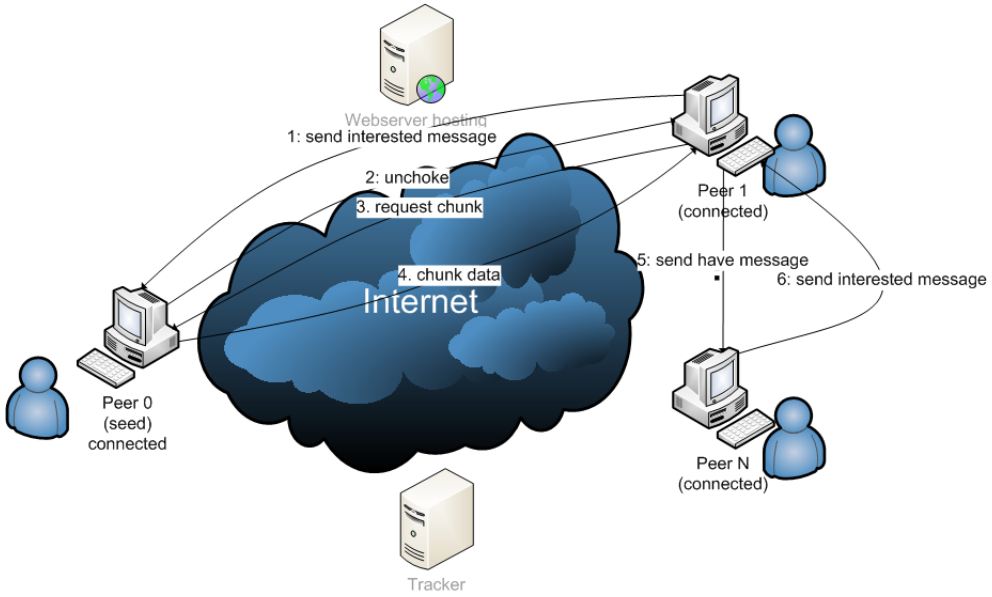


Figure 2.4: Data exchange between peers

chunk. Peer 0 will send the data to Peer 1 either until the chunk is uploaded completely or until Peer 1 is choked. If the chunk is uploaded completely, Peer 1 will generate and send a have-message to all his neighbours to inform them of the change. Peer 2, who receives the have message then compares the chunk-ID to his bitfield and sends an interested message if it does not own the chunk yet.

Seeds

Seeds will unchoke a random peer for an arbitrary amount of time depending on the implementation. Earlier versions chose the downloader with the highest download rate from the seed. Later clients abandoned this strategy because peers with a high download rate were preferred. This way, a free-rider with an exceptionally high download speed could bind all seeds to himself. Other

implementations use a simple round robin strategy to distribute their upload evenly among all the peers.

After the last chunk has been received, the peer changes his status and becomes a seed. This means that he will only contribute to the system and does not need any pieces from other peers. Seeds are expected to stay in the system until their share-ratio reaches one but there is no way to ensure this. At the moment a peer becomes a seed, all pieces of the file are put together on the harddisk and the file is ready to use.

Choking mechanism

BitTorrent's choking mechanism decides which peer will be allowed to download next. This decision is recalculated every ten seconds in the standard implementation. The algorithm is as follows:

- Compare the optimistic unchoke slot to each upload slot (the comparison is based on a twenty seconds rolling average of the download rate from that peer).
- Send a *CHOKE*-message to the uploading peer with the least upload if the peer at the optimistic unchoke slot uploads at a faster rate.
- Take over optimistic unchoke slot to an upload slot (data transfer here is now bidirectional).
- Randomly choose another, not yet utilized peer to fill the vacant optimistic unchoke slot.

Optimistic unchoking

Each peer has an optimistic unchoke slot over which he will send data to a randomly chosen client for a given amount of time (usually 30s) and will wait for reciprocation. The choking mechanism may decide to exchange the optimistic unchoking slot into the normal uploading slot if the reciprocation is higher than the one of the peers that are already uploading. Using optimistic unchoking, a peer can explore the network and find peers that provide faster download speeds for it.

2.2 Game Theory

Game theory is a relatively young field in mathematics. Although problems that were related to it were discussed as early as in 1713, it did not emerge as a unique field in mathematics until 1928, when John von Neumann, a mathematical genius who was a member of the manhattan project and who also gave the name to the von Neumann computer architecture, published papers like "Zur Theorie der Gesellschaftsspiele" (Eng.: On the theory of parlour games). Game theory is similar to decision theory but while decision theory is concerned with decisions in a static environment, game theory also takes the reaction of the environment into account. The subject of game theory is the analysis of strategic decision situations, e.g. situations, in which the result depends on the decisions of several players in such a way that none of the players may decide the result alone.

Game theory is an extremely versatile field and has been applied to nearly

all disciplines such as economics, engineering, politics[Gut97], psychiology, social science, biology, computer science et cetera. There is nearly no situation involvingn humans decisions that cannot be understood better when analyzing it using game theory.

When analyzing a situation with game theory, the payoff for the players has to be defined. The payoff or utility is a value that represents how much the situation's outcome is worth to the receiving player. This can be very easy when looking at a situation that can be described mathematically. Situations like this may involve monetary payoffs in games where it is easy to say that a player may obtain £10 if he chooses to play strategy A but has a chance of winning £100 if playing strategy B. But most of the time, it involves "human factors" like greed, preference, and the value of things to an individual that cannot be put into real numbers. A *rational player* in game theory will always decide to play the strategy that will provide the maximum payoff. The importance of the "human factors" has to be modelled into the utility function.

2.2.1 Visualisation of games

Games can be represented in several forms. The most common is the strategic form (or normal-form). In the matrix of the normal-form each cell contains the players' payoff when playing a certain strategy combination. In the simplest form this payoff is written as a number. For the decision of a rational player only the relations of the numbers are of importance. All players play at the same time, this means that they decide on a strategy without knowing the other players' decisions. If a player can win five gold coins when playing a

strategy or a glass of water when playing another strategy, the payoff may not be given as 5 gold or 1 water but rather as a number that indicates the usefulness of the items to the winner. This utility function to calculate the payoff will model all the conditions into the game. If the game would be held in a desert, a glass of water may have a higher value for the player than five gold coins. The amount of effort a player has to spend to gain victory can also be modeled into the game. If the strategy for winning five gold pieces requires the player to work for several years while the glass of water just requires a button to be pressed, then this has to be included into the equation as it will certainly affect the player's choice of strategy.

A game may also be represented in extensive form. In this form, they are normally represented as trees. Each vertex represents a choice of a player. The vertex is annotated by a number specifying the player who will decide. The payoff of the strategy combination is written at the end of the path. In an extensive form game, players can react on the choice of the opponent.

Both forms of visualisation are not suitable for more than three players, although they can of course be used to model games with more players. Normal form games will have one dimension of the matrix per player. If there are more than two players, combining the opponents can help reducing the game to a one on one game with the player playing against the rest of the players.

2.2.2 Nash Equilibrium

The Nash equilibrium is a very important concept in Game theory. It is defined as a situation where no player has an incentive to deviate from his strategy

because it would always decrease the expected utility. The Nash equilibrium will lead to a preference order as shown in equation 2.1. Let S be the set of possible strategies, $s^{*i} \in S$ the chosen strategy of a player i, $s^i \in S$ a strategy that the player i could choose and $s^{*-i} \in S$ the strategy combination chosen by the opposing players.

$$s^{*i} \succeq_{i|s^{*-i}} s^i \tag{2.1}$$

The strategy combination chosen by the player must have a higher conditional order of preference to all the other strategy combinations. From this follows the definition of the Nash equilibrium in equation 2.2 [Sch04].

$$\left(s^{*i}, s^{*-i}\right) \succeq_i \left(s^i, s^{*-i}\right) \forall\, i \in A, s^i \in S^i \tag{2.2}$$

If only one player has an incentive to deviate from the strategy combination, then it is not a Nash equilibrium. Given the other players play s^{*-i}, strategy s^* is preferred over all other strategies s^i if it is a Nash equilibrium.

2.2.3 Prisoner's Dilemma

The prisoner's dilemma is one of the well known situations that can be studied using game theory. The cover story of the prisoner's dilemma is that there are two suspects in police custody. They have no ability to communicate with each other. An officer gives them the choice to admit the crime or keep silent. The payoff if both keep silent is high[2]. If one breaks the silence and the other one refuses to give evidence, the one who talks gets a very low sentence and

[2] A high utility value means a low sentence in this case

the one who says nothing will be punished more severe. If both admit the crime, they are both given a medium sentence. The situation can be seen in normal form in table 2.1.

		Player A	
		Admit	Conceal
Player B	Admit	(10/10)	(5/20)
	Conceal	(20/5)	(15/15)

Table 2.1: Payoff matrix of the prisoner's dilemma.
Note: Higher payoff means lower sentence

The best decision, according to the payoff matrix, would be for both to stay silent. Then both would get away with a moderate sentence because the police could only prove a minor crime. But the definition of the game requires that there is no communication between the players whatsoever. If player one compares the payoff of silence to the one if he would admit it, then even if the suspects had sworn not to say anything before they were caught, he would have an incentive to admit and get the higher payoff. But because both players are rational beings, they can already imagine that their counterpart may not keep word. Then the only rational choice is to admit. If one is silent, then he risks being betrayed. If he talks, he has the chance of a big payoff in case the other one keeps silent and does not risk to be betrayed. If both players decide to admit, the system is in equilibrium because no player can gain an advantage by unilaterally changing his strategy. This point, at which no player may increase the utility by a solitary change in strategy is called the Nash equilibrium[3]. If all parties act rationally, then this state will be kept forever. If external influences may occur, then they would have to be modelled

[3] Named after the person who proposed it: John Forbes Nash

into the payoffs.

Looking at the prisoner's dilemma in the more generalized form in 2.2, the following things have to be true for the values of T, R, P and S to make the situation a prisoner's dilemma: $T > R > P > S$.

		Player A	
		Admit	Conceal
Player B	Admit	(P/P)	(S/T)
	Conceal	(T/S)	(R/R)

Table 2.2: Payoff matrix of the Prisoner's Dilemma. P = Punishment, S = Sucker's Payoff, T = Temptation, R = Reward.

Chapter 3

Literature Review

This chapter will give an introduction into the literature regarding the topic of free-riding in BitTorrent and introduce the analyses and results of other groups or individuals who have put effort into resolving the issue. The effectiveness of their approaches as well as their practicability will be discussed. After reviewing the current situation and solutions to the problem, a novel solution shall be developed that takes into the account the shortcomings and the advantages of the reviewed material.

Material that could be of interest for the development of a new or synthesized solution will be listed among possible solutions and results done by others.

The first section will give an overview over the ways of cheating in a file sharing system as defined by [LZL08].

Another section will look at situations when contribution to the system happens involuntarily by the use of technology such as firewalls.

Successful strategies, as defined by [Axe84] will show which attributes a strategy should have to be successful. Axelrod performed computer tournaments and analyzed the competing strategies and their properties.

Finally, the last part of this chapter will introduce the existing approaches to the free-riding problem and discuss their effectiveness.

3.1 Ways of Cheating

The paper [LZL08] classifies cheating into two categories: Cheating on private information and cheating on buffer information. Both things cannot be used to free-ride in BitTorrent. Free-riding is done by exploiting the two kinds of peers to which tit-for-tat does not apply — peers that are optimistic unchoking and seeds. Among others, [LMSW06], [Moo06] and [QS04] state that optimistic unchoking and seeding violates tit-for-tat and can be exploited by free-riding peers.

3.1.1 Cheating on Private Information

If information has to be reported by the client software, then certain data of it may be used to cheat the system. If a system relies on peers reporting whether or not someone had sent them data, then this information could be sent for a peer who didn't send anything or suppressed to make another peer look bad to the system. This may not enable free-riding but it may give the cheater an advantage over others. Examples would be the amount of downloaded data (piece set), the upload capability, et cetera.

3.1.2 Cheating on Buffer Information

A client program has to report which pieces it already possesses so that others can request them. By reporting a lower number, a client may reduce the number of requests by others. BitTorrent is good in preventing this by replacing peers that don't have necessary parts with better peers (i.e. peers that upload to them with a faster rate). Under-reporting to seeds makes no sense because seeds will upload to anyone, even if the one has all pieces but one. The paper [LLSB08] shows how under-reporting can be used to keep the neighbouring peers interested by only showing those pieces which are most common but which the neighbour does not yet possess. Over-reporting, on the other hand, can be detected and punished by neighbouring peers because after being unchoked, the requesting peer will decide which piece to request. If the request isn't answered, then the peer can decide to ban the over-reporting peer.

3.1.3 Free-riding

Free-riding is the term used for the action of using a common good without paying or exchanging for it. This draws resources from the system. BitTorrent relies on the peers sharing their bandwidth. Like in every closed system, the sum of the upload bandwidths equals the sum of the download bandwidths. If a peer does not contribute its upload bandwidth while using the bandwidth contributed by others, it will harm the system in two ways. First by binding the resources to itself and second by refusing to give something in return.

As mentioned in this section's introduction, BitTorrent offers two weak points that can be exploited by free-riders. One is the optimistic unchoking mechanism and the other one are the seeds. Although BitTorrent uses a tit-for-tat strategy to prevent free-riding, this tit-for-tat strategy does not apply to those two cases. While peers are exchanging data, they know that the one who reciprocates is not a free-rider. They can act accordingly by only sending data to peers that reciprocate. Other peers will not be taken over into the regular upload slots. Seeds do not receive any upload and are thus not able to know who is a free-rider and who isn't. Furthermore, if all peers would wait for the reciprocation of their partners, they would wait indefinitely because one of them has to do the first step. One of the peers has to start by uploading data and waiting for reciprocation. This mechanism of discovering new partners is called optimistic unchoking. When optimistic unchoking, the peer sends data to a random partner for a certain amount of time and hopes for reciprocation. If the other peer does not react, then the upload is stopped. But no retaliatory action is taken nor is the peer remembered as not being cooperative. This way, a free-rider can obtain data without uploading anything.

3.2 Network Problems - Why some users cannot contribute

There are also not only malicious free-riders. The paper [MPES08] shows problems of users that are firewalled or use NAT and therefore cannot accept

incoming connections. This may make them look like free-riders to other peers in cases when a peer tries to connect to them. Firewalled peers can only upload to connectable peers but not to other firewalled peers. This means that a firewalled peer is not really free-riding but rather only uploading to a subset of other peers. This has to be taken into calculation when punishing peers. A peer that looks like a free-rider to one peer, may reciprocate to another peer.

Certain settings and technologies may prevent a user from reciprocation. These include:

- **NAT**: NAT[1] is a popular default setup for broadband users. One public IP address is assigned to the router. Traffic initiated from the inside to the outside passes the router. The router translates the internal source address to the external public address. Responses are retranslated to the internal address. But the problem is: It does not know to which internal address to route an incoming connecion to the public address. Therefore a user behind a NAT in its default configuration can not receive incoming requests but can initiate a connection which can then be used for data transmission in both directions.

- **UPnP**: With UPnP[2], users can open ports to the internet, resulting in security threats. The answer to these threats are firewalls. This shifts the problem from NATs to firewalls as the source of not being able to contribute.

[1] Network Address Translation
[2] Universal Plug and Play

- **Firewalls**: Many firewalls are configured not to allow incoming connections. Especially in corporate environments, most ports are blocked and the user himself has no means of opening the necessary ports. Firewall puncturing may help to reduce this problem but it does not always work and would require heavy modification of the protocol.

3.3 Successful Strategies

When analysing the strategies in the iterated prisoner's dilemma, Robert Axelrod[Axe84] provides a list of characteristics that make up a good strategy. Those characteristics will be listed and briefly explained. Afterwards, tit-for-tat, the winning strategy of Axelrod's computer tournament, will be introduced. [Axe84] lists the following characteristics:

Nice means that the strategy is not the first to defect.

Retaliatory means that a strategy will punish when the other player is defecting. Players that are not retaliatory can be easily exploited by the other side.

Forgiving means that the punishment has to end at some point to give the other side a chance to react to the punishment and hopefully cooperate.

Clear means that the reaction of the strategy to a behaviour should be deterministic. A strategy with a high clarity benefits from this because others know how to handle it. If an opponent is punished by the strategy, it can

know that it happened because of its actions and act appropriately in a future encounter.

Tit-for-tat came out as the winner of the tournament and it has all the charateristics of a good strategy. It will be nice in the first round. It will punish another player for a defection (retaliatory) and will do it in the next encounter (clear). After that, it will imitate the behaviour of the other player. This means that if the other player acts nice in the next round, the tit-for-tat strategy will be forgiving the player and play nice as well, thus fulfilling the last required characteristic of the list.

3.4 Evolutionary Aspects of Swarms

In the paper [HP05], the authors state that BitTorrent proves to be so resilient not only because of its tit-for-tat-mechanism but to a great extent because there are several swarms for one and the same file. Each swarm has a different composition of peers, seeds, and free-riders. Peers will try to maximize their download rate by choosing the swarm with the highest number of seeds. A swarm with a small number of seeds or a low availability does not look promising and will be left. If the availability is high but there are many free-riders in the swarm, peers will leave it because of its unsatisfactory download rate. This will eventually leave the free-riders alone in the swarm and the swarm dies out. In this evolutionary process, only the fittest swarm will survive.

3.5 Solutions to the free-riding problem

In this section, solutions to the problem will be introduced and discussed.

3.5.1 A central server to store contribution information combined with proportional share allocation

The paper [LYWM07] confirms that seeds have a high impact on free-riding and that within a system with many seeds, the original tit-for-tat mechanism does not work very well. The paper also states that optimistic unchoking has not a very high impact on free-riding. That means that seeds are the main source of bandwidth for the free-riders and that they have to be taken into consideration when finding a solution to the free-riding problem.

In the paper, the authors introduce a central trusted third party server which is responsible for keeping track of the fairness of the peers. The central server in their idea is using Eigentrust or DRBTS to ensure that the reports are truthful. Eigentrust, according to the authors, is very resilient to false reports. The paper [KSGm03] shows that over 70% of the peers in a network must collude maliciously to be able to fake the data.

Discussion

In their algorithm, Li, Yu, and Wu let each seed share its bandwidth according to the contribution of a peer to the system. The contribution values are retrieved by the seeds from the central server. The authors show that their proposed mechanism works and that there exsists a Nash equilibrium for their

scenario. The use of a proportional share model is very advantageous because there is not only the incentive to give just enough but an incentive to give as much as possible because more contribution will result in more utility from the system. A proportional share algorithm has also been used by [LLSB08, MLLY04].

In their work, the authors of [LYWM07] assume, that a mechanism such as Eigentrust is preventing any kind of false reporting of the peers. If all peers report to the central server, then this would indeed be the case. But when looking at the situation game theoretically, then there is an incentive to not report at all because if one reports that another peer has sent data, then this will only make the peer whose contribution has been reported more attractive to the seeds than the reporting peer. The central server does not know about the transactions because they are not routed through it so it can only manage information it gets from the peers and does not know whether a message has been withheld. Left out reports won't go into the calculation of contribution. This could be especially a problem in a time when the system is switching to the new solution and old and new clients should work together.

Another drawback of their approach is, that they do not include optimistic unchoking peers into their considerations. Although they show that the amount of data obtained by optimistic unchoking is significantly less than the amount of data received from seeds, they are nevertheless a source that should be handled.

Furthermore, the idea of a central server would increase the requirements for the servers which would have to handle many requests more than before.

The tracker, the central server of a swarm, already has to handle a lot of traffic due to peer list requests and scrape messages. Peer lists can only be obtained at a certain frequency set by the tracker's administrator. This interval is sent with the peer list and not obeying to it may result in being banned from the tracker.

[Wik10]: "... the tracker is scraped many thousands of times for that torrent alone, even if the swarm is not very big. The tracker can usually handle this number of requests. However, if there are more requests than strictly necessary, this can destabilise the tracker and put it offline." Vuze[3] does scrapes "in order to determine whether or not to send an announce requesting more peers. Sending a list of peers is usually more bandwidth consuming than sending a scrape result."[Wik10]

This means that the scrapes are already a measure to reduce the load of the trackers.

If a seed would always ask the central server before deciding which peer to unchoke, the number of requests would exceed the number of scrapes and would certainly be a problem for some of the servers that currently serve as trackers.

Another problem with this idea is that a central server would be necessary for all the swarms that should profit from the improvement. And for the central server to work, the peers in the swarm would have to implement the new algorithm. If a seed would still run an old version of the client, it would also upload to free-riders. If most of the peers would run old clients, then

[3] A popular BitTorrent client, formerly known as Azureus

a small number of malicious peers could outnumber the peers using the new mechanism and manipulate the Eigentrust secured trust data.

The advantages of the mechanism are: First, it takes the seeds into account. Second, the incentive for peers is thus that uploading more to the system will result in better service from the seeds. This encourages peers not only to upload a little as not to be marked as free-rider, but really to upload as much as they can in order to get a higher download rate.

The disadvantages are: The central server which would have to be able to handle a high number of requests resulting in high cost for the owner due to increased traffic and hardware requirements. The switch from the old mechanism to the new mechanism would best be done for all clients at once because if only one client would find itself in a swarm together with some malicious peers and a number of peers implementing the old mechanism, then the malicious peers could fake messages to make the central server believe that they have contributed a lot to the system. Only if a high number of peers including the seeds would use the new mechanism and would be connected to a swarm that is enabled with an additional server, the new mechanism would work satisfactory.

3.5.2 BitTorrent as an Auction

Another similar solution is introduced by [LLSB08]. The authors of the paper use a proportional share auctioning model to improve BitTorrent's incentives. Each peer "bids" for bandwidth of another peer. This mechanism is meant to replace the current optimistic unchoking of BitTorrent. One problem of

their solution is the bootstrapping of clients that have no pieces to offer in an auction. This problem is solved by a special bootstrapping piece exchange in which two clients use a third client as a proxy to exchange an encrypted piece. After verification of the successful transfer, the key is sent and the bootstrapping is done. The peer can now bid in an auction.

Discussion

While proportional share offers incentives for every client to give as much as possible, the bootstrapping problem proves to be difficult to solve in a solution that expects mutual exchange. In mixed population environments, where original and modified clients interact with each other, proxy peers may not be easy to find. While the switch to the new system would eliminate the chance of free-riders to exploit optimistic unchoking, seeds are ignored. [IUKB+04] found that 40% of their downloaded bytes were uploaded by seeds. Seeds play an important role not only for the download rates of normal peers but also for free-riders. The auction model takes care of the exploitation of optimistic unchoking. Li et al. [LYWM07], however, state that "free-riders do not impose a major impact through optimistic unchoking".

3.6 Other information

Another paper, [QS04], analyses the performance of BitTorrent and uses a simple fluid model for analysis. Qiu and Srikant identify optimistic unchoking as one of the sources from which free-riders can obtain service. They formulate equations that can be used to model the current BitTorrent implementation.

[LLSB08] analyse BitTorrent and agree that BitTorrent is not tit-for-tat and that BitTorrent is not fair. Because fairness is defined differently by different authors, they argue that BitTorrent is unfair because a downloader will always only receive an equal part of the bandwidth of a peer, even if it might upload much more than another downloader connected to the same peer. This can be exploited by peers that minimize their upload to such an extent that they are not replaced by others but do not have to sacrifice their whole upload bandwidth.

The authors of the paper see BitTorrent as an auction and propose an algorithm to alleviate the problem of free-riding called "proportional share". To replace optimistic unchoking, the authors suggest that a peer should first "... give an arbitrarily small amount, e.g., a single block. In the subsequent round, a prop-share client will ramp up its allocations ..."[LLSB08]

3.6.1 Piece revelation

The paper [LLSB08] analyses the effect of strategic piece revelation, i.e. prolonging the interest of other peers in oneself by hiding pieces that are of interest to them and only showing those which they are interested in but which are most common among the peer set. The download time decreased when using this method in a mixed population test. This shows that peers can benefit from under-reporting.

Discussion

While it can clearly be an advantage for peers to under-report in the way the paper suggests, it also becomes obvious that this is an exploit at the cost of the normal peers. Once all standard peers use the piece revelation strategy, the rarest first policy does not work properly anymore and the availability is reduced. This would result in a reduction of efficiency that could not easily resolved. After all peers are using the piece revelation strategy, there would be no incentive to switch to the old implementation because then one would get exploited.

Under-reporting would only be an advantage if the under-reporting would not be used to exploit but rather to encourage reciprocation.

3.7 Prisoner's Dilemma

The prisoner's dilemma is one of the most famous cover stories for a game theoretic situation. Several papers have modelled BitTorrent as being a prisoner's dilemma [LFSC03, JA05, HP05].

3.8 Simulation

[NS2] can be used to simulate BitTorrent networks. There exists a working simulation [EHBK07] for an older version of NS2 that may be modified and can be used as a basis for a simulation of a new algorithm. The simulation includes the standard BitTorrent algorithms as a C++ implementation.

Chapter 4

Algorithm Design and Analysis

This chapter will present the proposed modification to the BitTorrent mechanisms and explain how they differ from the original.

4.1 Proposed Modification

A modified BitTorrent client will not reveal all of its chunks at once to a neighbour. This strategy of withholding some of the chunks will prevent the neighbour from downloading the whole file. To ensure that all peers withhold the same pieces to a certain peer, all use the same algorithm to calculate the piece set. This algorithm will be described later on. BitTorrent's choking mechanism works in a way that the requester can choose which piece it wants to download. If the piece is not sent, then this means a violation of the protocol and the requester will ban the peer. Therefore, the piece withholding must happen earlier. The bitfield, which is sent just after a connection to a new neighbour has been established, mustn't contain the chunks that are withheld.

Have messages, which normally inform the neighbours about a peer's newly obtained chunk, have to be kept back as well. This way, a peer will look like not having those chunks. Moreover, seeds will look like peers which have all pieces but those that are withheld. Of course, this has an effect on the availability and the interest of the peers. This will also be discussed in chapter 4.2. If a peer receives a chunk from another peer, then it will randomly choose one of the withheld chunks. A have message is sent out at once if the peer already has the chosen chunk. If the chunk is not yet in its possession, then the have message will be sent out as soon as it obtains it. Assuming a population in which all peer use the proposed mechanism, a newly joined peer will see only the pieces that are not withheld from it. This will make the torrent seem dead. But as the peer participates in the exchange of data, have messages for the withheld pieces will arrive and the availability will increase. For a free-rider, this situation will not change. Because a free-rider does never send any payload data, withheld pieces will also never be revealed to it. This way, a free-rider will always see an incomplete torrent that is very unattractive. A free-rider that downloads all the chunks that are available will still steal resources from the network but it can't finish its download until it at uploads at least enough chunks to reveal all the available chunks. This will either enforce a minimal amount of fairness or provide an incentive to the free-rider to leave the swarm in search for a vulnerable one.

Figure 4.1 shows the process starting from the point when a peer (n) receives a chunk from one of its neighbours (i). After a piece has been received, the bitfield corresponding to the chunks of the sender (B_i) is updated. Then the proposed algorithm checks if the set of withheld chunks (W_i) for this

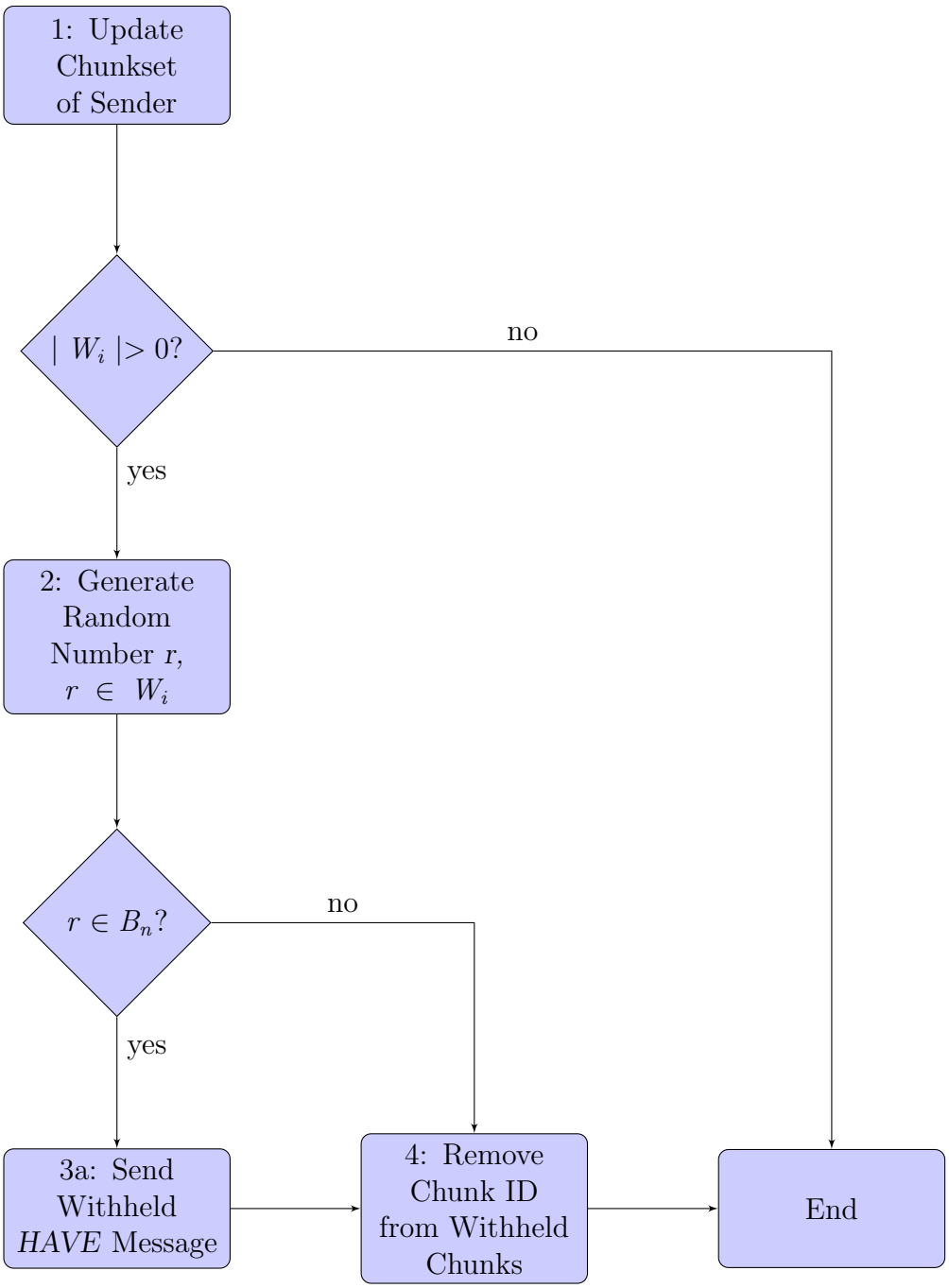


Figure 4.1: Modified behaviour after chunk has been received

neighbour contains chunk ids. If there are no chunks in W_i ($W_i = \emptyset$), then the peer is treated normally. If $W_i \neq \emptyset$, then there are still chunks hidden from i. In step 1, the receiver will generate a random number $r \in (1.. \mid W \mid)$ which corresponds to a chunk id in the set W_i. If peer n possesses the randomly chosen chunk ($r \in B_n$), then a *HAVE* message is sent to peer i (step 2). Subsequently, the chunk is removed from W_i (step 3). If peer n does not yet have the chunk, then it will only be removed from W_i. As soon as peer i receives the chunk with the ID r, it will send a *HAVE* message to peer n. The sending of *HAVE* messages is visualized in Figure 4.2. *HAVE* messages are only sent if the chunk p is not contained in the withheld set W_i. If a *HAVE* message has been withheld in this way, then it will be sent out as soon as the peer i sends a chunk and the chunk id is chosen by the algorithm (see Figure 4.1).

4.1.1 Calculation of the Withheld Chunk Set

Figure 4.3 shows the creation of the withheld chunk set W_i for a peer i. A chunk set of n chunk ids is created.

To calculate the chunks that are withheld, an algorithm has to be used which yields the same result on every client. The withheld chunk set is calculated for a peer upon connection. To calculate the chunk set, a peer uses the long value of the IP address of the peer for which it wants to generate the chunk set. Using this IP value, it computes a four byte SHA-1 digest. The resulting hex value is converted back into a long value. From this long value, a chunk id is generated by a modulo operation. This modulo operation will

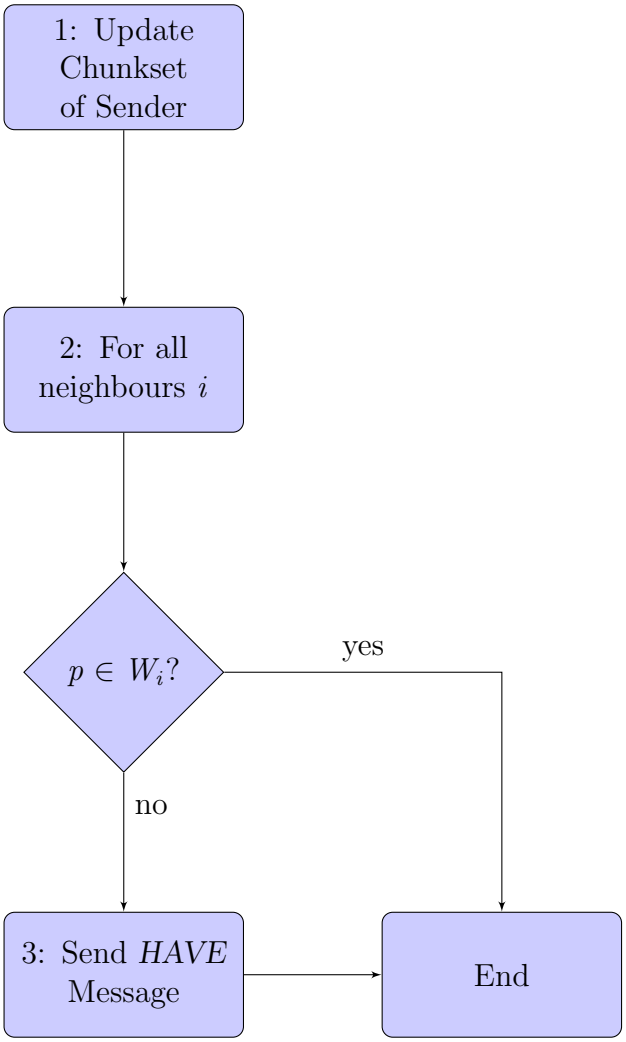


Figure 4.2: Modified behaviour after chunk has been received *HAVE* messages to neighbours

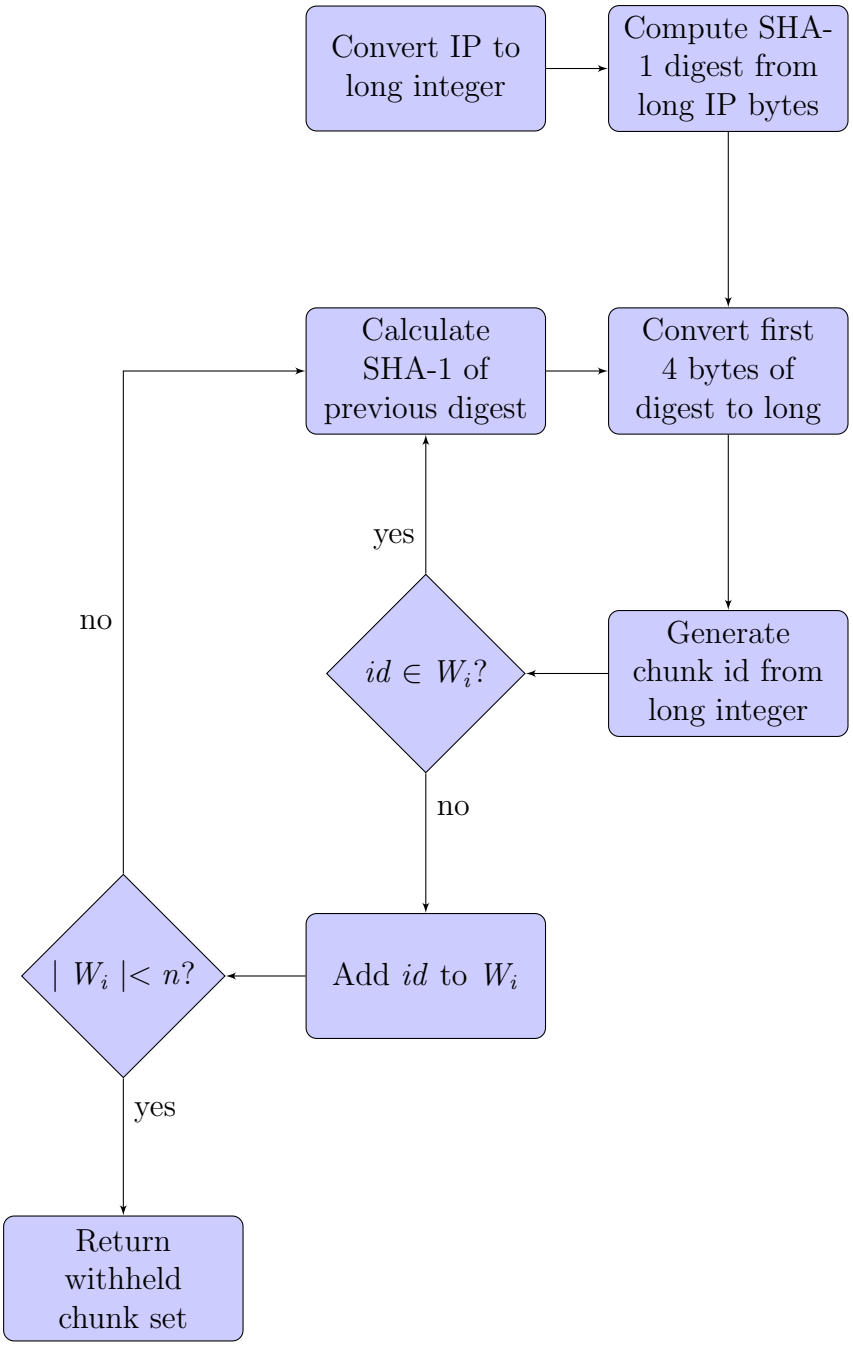


Figure 4.3: Generation of withheld chunk set for a peer i

limit the resulting number to a range between 1 and $max(chunkid)$.

The four byte SHA-1 digest is then fed back into SHA-1 to generate a new digest. For this digest, the process is repeated to compute a chunk id. If the chunk id is already present within the withheld chunks set, the process is repeated as well. After the necessary number of chunk ids has been created, the result is saved to form the chunk set that is withheld from this peer.

4.2 Analysis

Several authors have related BitTorrent to the prisoner's dilemma (see chapter 2.2.3). The solution for the prisoners in this situation is to admit because it will be the best answer to all strategies, their opponent could choose. From the point of view of the police, this is a great situation because the prisoners admit because it is reasonable to do so. For policing BitTorrent, one must also create a situation in which it is reasonable for peers to contribute. This chapter will analyse the proposed mechanism and discuss its ability to provide an incentive for contribution.

A simple game theoretic model can be used to analyse the effectiveness of the current BitTorrent algorithm. The payoff equations will be kept as simple as possible. Download is what peers want, so it will be a positive value. Upload is something that does not help peers so it must be negative. If a player does both at the same time, then the payoff is less than when only downloading. Although there is a subjective component in the model because one has to estimate how negative a peer values uploading.

If one models BitTorrent this way, the resulting payoff matrix (table 4.1)

shows a discrepance to the prisoner's dilemma. In the prisoner's dilemma, the variables must have the following relation: $T > R > P > S$[1]. The original BitTorrent mechanism violates this requirement. T must be greater than R to provide a prisoner's dilemma like situation. In the simple BitTorrent model, $T \widehat{=} D$ $R \widehat{=} (D - U)$, $\forall\, U > 0 : D - U < D$ means that every player has an incentive to deviate from the contributing strategy and switch to free-riding. But when one peer is free-riding and the other is contributing, then there exists an incentive to the contributor also to switch to no contribution. Ultimately, this will lead to the tragedy of the commons as it is described by Hardin [Har68].

To modify the model in order to make the desired strategy combination "Contribute/Contribute" more attractive to the players. This is done by modifying the conditions and rules of the games, thus increasing the payoff of using the "Contribute" strategy. A rational player will choose the strategy that provides the highest payoff.

		Player A	
		Contribute	Not Contribute
Player B	Contribute	(D-U/D-U)	(-U/D)
	Not Contribute	(D/-U)	(0/0)

Table 4.1: Payoff matrix of BitTorrent. D = Download, U = Upload

Free-riding occurs, when a player never uses the strategy "contribute" during the whole time of downloading the torrent. This means that in principle a single case in which a peer is forced to choose the "Contribute" strategy would suffice to make him a non free-rider by definition. The more times a peer can

[1] P = Punishment, S = Sucker's Payoff, T = Temptation, R = Reward.

be forced to use the "Contribute" strategy, the fairer the system will become. A player should therefore be forced to choose the strategy "contribute" at least at a certain time. If the player would have to contribute each time, then complete fairness would be reached but the performance of the system would decrease. Seeds and optimistic unchokes, which are an essential part of BitTorrent, could not be utilized. To be able to use them but still enforce a certain level of fairness, the withheld chunks are introduced. This concept will not change the equations within the payoff matrix of table 4.1 but rather the way D and U are calculated.

For a homogeneous system, [LYWM07] defines the download of a peer as being

$$d(t) = \mu \cdot [(1 - \rho(t))\eta x_n(t) + (1 - \kappa(t))y(t)] \tag{4.1}$$

with μ as the upload rate of a single peer, $1-\rho(t)$ as the bandwidth assignment criteria of non-free-riders, η as the efficiency of file sharing [QS04], $x_n(t)$ the number of non-free-riding peers at time t, $1 - \kappa(t)$ the bandwidth assignment criteria of seeds to non-free-riders, and $y(t)$ as the number of seeds at time t.

While the bandwidth assignment stays the same, seeds are not recognizable to a free-rider anymore. Therefore, $y(t)$ will not spread its upload capacity uniformly among all peers as described by [LYWM07]. At one point, there are no pieces left for the free-rider to be interested in. This means that the seed part of equation 4.1 needs another component which will be called ψ ($\psi \in [0..1]$) and which will be the probability of seeds having chunks that the

peer does not possess.

$$d_n(t) = \mu \cdot [(1 - \rho(t))\eta x_n(t) + (1 - \kappa(t))\psi y(t)] \tag{4.2}$$

$$d_{fr}(t) = \mu \cdot [\rho(t)\eta x_n(t) + \kappa(t)\psi y(t)] \tag{4.3}$$

η provides a measurement for the effectiveness of filesharing, ψ is necessary to reflect the changes made to the behaviour of the seeds. [QS04] define η as $\eta = \mathbb{P}\{\text{downloader } j \text{ needs no piece from downloader } i\}$. η for the proposed mechanism can be calculated using equation 4.15 (see chapter 4.2.2).

Using the modified algorithm, seeds are now not necessarily of interest to a downloader. Thus, the assumption of [LYWM07] that the bandwidth of seeds is evenly shared to all the connected peers has to be modified to fit the new situation. The bandwidth of seeds is equally shared among the peers that are interested in the chunks that are visible to them. The new factor ψ incorporates the chances of a seed being of no interest to a downloader into equation 4.2.

$$\psi = \mathbb{P}\{\text{downloader } i \text{ needs pieces from seed } j\} \tag{4.4}$$

Equation 4.4 can also be calculated using equation 4.16 see chapter 4.2.2). If a free-rider would now join the system and download chunks, then it could only do that as long as it is interested in other peers. With the withholding mechanism in place, the withheld chunks would only be revealed to it if it would upload chunks to its neighbours.

If there are no new chunks available for the free-rider to see, then the situation will be ok until it the free-rider has downloaded all the chunks that are visible to it. When the free-rider has obtained the last chunk, then ψ will be zero, because seeds will always only show the chunks that are not part of the withheld chunk set for the free-rider. At the same time, the free-rider is not interested in any of the other peers' chunks because if it has not sent any data in the past, the peers will also only show the not withheld chunks to the free-rider. Although the other peers might be interested in the free-riders chunks, they have nothing to offer and so they will also not unchoke the free-rider. This means that at this point, the download rate of the free-rider is zero and does not increase until the free-rider contributes at least a chunk to a downloader. This would make the free-rider a downloader by definition and means that the mechanism was effective. The payoff matrix of this situation can be seen in table 4.2.

		Free-rider A	
		Contribute	Not Contribute
Downloader B	Contribute	(-/-)	(-/-)
	Not Contribute	($D_f - U$/D)	(0/0)

Table 4.2: Payoff matrix of a free-rider when all but the withheld chunks are downloaded

Table 4.2 shows that Downloader B can't contribute to Free-rider A because A does not need any chunks. The decision in this case is only the free-rider's. If the free-rider contributed, then it will receive the payoff $D_f - U$ where D_f is an expected download rate in the future. B will receive download rate D.

For a situation in which a downloader A and a downloader B interact

using the proposed chunk withholding mechanism, the payoff matrix can be modelled like in table 4.3. The peers do not only receive a download rate because the other player contributes. They also receive a future download rate D_f for their contribution. D_f will be zero after the downloader has obtained all the chunks that are within the withheld chunk set but until then, it serves as an incentive for the downloader to contribute as long as $D_f > U$. That $D_f > U$ can be assumed because the payoff describes the use of the outcome to the player. The player wants to obtain a full copy of the torrent's contents, therefore download is valued higher than upload. So the assumption for D_f is, that $D_f > U$.

		Downloader A	
		Contribute	Not Contribute
Downloader B	Contribute	$(D + D_f - U / D + D_f - U)$	(D/-U)
	Not Contribute	(-U/D)	(0/0)

Table 4.3: Payoff matrix of two downloaders using the proposed chunk withholding mechanism

Looking at table 4.3, one can see that the strategy combination (Contribute/Contribute) is a Nash equilibrium because as long as $D_f > U$, there is no incentive for any of the players to switch to another strategy. According to the definition of the Nash equilibrium (see equation 2.2 in chapter 2.2.2), there is no strategy that would offer a higher payoff if the peer would switch to it. The strategy "Contribute" is even a strictly dominant strategy for each player if $D_f > U$.

Now the situation is analysed if a peer would only upload to one peer to get all the chunks revealed. This would mean that the mechanism would ensure at least a share-ratio of $\frac{|W|}{S}$, $|W|$ being the number of withheld chunks and S

the number of chunks of the torrent. For 250 withheld chunks of a 400 chunk torrent, this would ensure that the exploiting downloader would have to reach a share ratio of 0.625. If a peer would try to exploit the system in this way, it would also risk that the peer to which it sends all the data will leave the system before it has sent all the chunks to the exploiter. This risk serves as an incentive to spread the contribution among several peers. Another property of the chunk withholding mechanism has the same purpose: the random chunk revelation. If a peer A uploads a chunk to a neighbouring peer B, then B removes a random withheld chunk from the withheld chunk set for peer A. If the chunk is in B's possession, then a *HAVE* message is generated and sent. If not, then B will send it out once it obtains the chunk. This insecurity which chunks are in possession of the neighbour prevent that an exploiter can upload chunks to a peer until there are no new chunks revealed and then go to another peer because it knows that the first peer does not possess any more chunks. Waiting too long to upload chunks does also not provide a good performance because once the point has been reached at which there are no chunks visible, a peer has to upload a chunk for each chunk it wants to download.

4.2.1 Availability

Influence of the new algorithm on the availability. The availability is a measurement in a BitTorrent system to see how many copies there are. Because chunks are withheld by the new algorithm, this will have a serious impact on the availability of chunks. In the original BitTorrent implementation, the availability is measured as can be seen in equation 4.10. To calculate the avail-

ability for a peer j, the chunk sets S of all peers are joined to the multiset[2] A as seen in equation 4.5.

$$A_j = \biguplus_{i=1}^{I} S_i \tag{4.5}$$

$$B = \{1, \ldots, N\} \tag{4.6}$$

Given that the torrent is the set B consisting of the chunks as shown in equation 4.6, the number of full copies that a peer sees can be calculated as seen in equation 4.7. The number of full copies is the minimum of occurrences of all chunk numbers. If a chunk is not contained within the multiset A, then no full copy exists.

$$\forall\, x \in B : f = \min\left(A_j \sharp\, x\right) \tag{4.7}$$

To calculate, how much of the chunk set B is contained within the incomplete copies, a set M is defined as the complement of the full multiset A and a multiset gained from union of f times the chunk set B. The chunk ids that are left are the ones not contained in full copies M (see equation 4.8). $M = \emptyset$ if there are only seeds in the system. $M \subset B$.

$$M = A_j \biguplus \left(\biguplus_{y=0}^{c} B_y \right) \tag{4.8}$$

The multiset M is the assigned to a normal set M_{unique} to remove duplicates. Equation 4.9 shows the fraction u of the chunk set B contained in all chunk

[2] In contrast to a set, a multiset can contain the same element more than once

sets of the peers which have not finished the download yet.

$$u = \frac{|M_{unique}|}{|B|} \tag{4.9}$$

Finally, the availability a_j can be written as the sum of the number of full copies plus the fraction of incomplete chunks.

$$a_j = f + u \tag{4.10}$$

In the normal BitTorrent system, f is just equal to the number of seeds in the system. But with the new algorithm, seeds do not necessarily look like seeds if a peer has entered the system. Seeds can only be recognized if an exchange happened while the seeds were still downloading. A seed will otherwise look like a peer that has every chunk except for the chunks that are withheld.

With the chunk withholding algorithm, equation 4.5 now has to be extended to reflect the fact that some peers are not showing all their chunks. The resulting equation is:

$$A_j = \biguplus_{i=1}^{I} (S_i \setminus W_{ij}) \tag{4.11}$$

All chunks of all peers are put together into the peer j's multiset A_j. Equations 4.6, 4.7, 4.8, 4.9, and 4.10 can be used without modifications.

In a system where all peers are running the original algorithm, the availability for a new peer can be easily calculated using the abovementioned equa-

tions. Each connected seed will add one to the availability and peers will add their share.

An example: A swarm comprises one seed and three peers, each of them running the original BitTorrent client. The torrent consists of ten chunks $B = \{A, B, C, D, E, F, G, H, I, J\}$[3]. Peer 1 has four chunks in its chunk set ($\{A, B, C, J\}$), peer 2 has already obtained five chunks ($\{B, C, D, E, F\}$) and peer 3's chunk set contains the chunks $\{D, E, F, G, H, I, J\}$. The seed is in possession of all ten chunks.

If a new peer joins a swarm, the availability for the original BitTorrent can be computed using the equations developed before. First of all, the chunk sets are combined to a multiset A (see equation 4.5).

$$A = \{A, B, C, J, B, C, D, E, F, D, E, F, G, H, I, J, A, B, C, D, E, F, G, H, I, J\}$$

Equation 4.7 will compute the number of full copies within this multiset by getting the minimum number of occurrences of all pieces within the multiset. Chunk A is contained twice, B three times, et cetera. If one chunk is not in the multiset, then it occurs zero times. This would mean that there is no full copy visible to the peer whose availability has to be computed. In the example, the minimum number of occurrences is two for any chunk $\in B$, hence $f = 2$. One full copy is the one held by the seed, the other one is spread across the peers. The two forms the number before the decimal point. Now equation 4.8 is used to remove the full copies from the multiset A. The complement of f times the

[3] Letters instead of numbers are used for simplification. A in this example represents the chunk with the id *1*, B represents *2* and so on.

full set B is computed from A. This way, the full copies that have already been computed, are not counted. This reduces A's contents to $M = \{B, C, D, E, F\}$ [4]. The fraction of incomlete copies can now be calculated using equation 4.9: $u = \frac{5}{10} = 0.5$ Finally, the availability value is computed by adding the two values according to equation 4.10: $a = 2 + 0.5 = 2.5$.

If the same swarm would use the proposed modification, the availability for a new peer would look much different. Two factors would play a role: The IP address of the peer in question and the number of withheld chunks. If the above example would be changes such that the withheld chunk set W_i for peer i would contain three chunk ids $W = \{A, D, E\}$, then the availabiltiy for peer i would be calculated the following way. First, as before, the union of all chunksets is computed using the modified equation 4.11. Then the full copies are counted with equation 4.7. In this case, because peer i has not yet uploaded any data to its neighbours, each peer is withholding chunks A,D and E. Thus, no full copies can be seen ($f = 0$). This makes the use of equation 4.8 superfluous because there are no full copies to be removed $M = A = \{B, C, J, B, C, F, F, G, H, I, J, B, C, F, G, H, I, J\}$. M is the assigned to $M_{unique} = \{B, C, F, G, H, I, J\}$. Equation 4.9 yields the result $u = \frac{7}{10} = 0.7$. Because there are no full copies visible to peer i, the resulting availabilty is $a = 0.7$. This only changes, once peer i sends chunks to its neighbours. They in return will send withheld *HAVE* messages for chunks they have and a will increase accordingly.

[4] In this case, all elements occur just once. $M = M_{unique}$

Maximum Number of Chunks to Withhold

A torrent can only be downloaded if each chunk can be obtained by one of the peers within the swarm. C is the set of chunks of the file. c is a single chunk of the set. W_i is the set of pieces hidden from peer i. G_i is the set of chunks that can be gained by a peer i ($G_i = C \setminus W_i$).

$$\forall\, c \in C : \exists\, c \in \bigcup_{i \in I} G_i \tag{4.12}$$

For all chunks in the set C, the chunks must exist in the union of the parts that may be obtained by the single peers within the system. No chunk shall be hidden from all the peers at the same time. Assuming that the withheld chunk sets are disjoint, the minimum number of peers in the system is $p \geq \lfloor \frac{|C|}{|W|} \rfloor$. A greater $\mid W_i \mid$ can still provide a stable system in a real life situation where peers frequently leave and join.

With $\mid p \mid$ peers in the system W can contain $|W| \leq |C| - \lceil \frac{|C|}{|p|} \rceil$ chunk ids. The number does not reach the maximum because the visible chunks for the peers needn't be disjoint.

4.2.2 Analysis of Interest

A peer is interested in another peer if this peer has at least one chunk that the first peer does not have. Once a peer discovers that it lacks a chunk, it will send an *INTERESTED* message to its neighbour to inform it of its interest. The *INTERESTED* message does not contain which chunk it is interested in. It is just a general expression of interest.

After implementing the chunk withholding mechanism, peers will initially see less chunks. This will have an influence on the amount of peers they see as being of interest and therefore also have an effect on the chance of getting an optimistic unchoke slot as peers are allocating the optimistic unchoke slot randomly among the peers that are interested in them. If the peer is not interested in anything because it cannot see the chunk in question, then it will also not be an optimistic unchoke candidate.

Interest in an Original BitTorrent System

To find an equation for the amount of peers that are of interest to a peer P_i, first an equation for the original algorithm will be formed. P_i is interested in P_j if the chunk set C_j of P_j contains one or more elements that are not contained in C_i. This means that for two peers P_i and P_j with given defined chunk sets C_i and C_j, P_i is only interested in P_j if $C_j \setminus C_i \neq \emptyset$. The interest of two peers can only be stated as yes or no if the contents of both chunk sets are given.

If one wants to look in a general way at the interest of peers with a chunk set containing a given amount of chunks without defining which chunks the chunk set contains, then a true/false statement can't be given for the case that P_i's chunk set contains more or equally many chunks as that of P_j. This will be discussed below.

If $|C_j| > |C_i|$, then P_i will be interested in P_j. The chunk sets do not have to be looked at because even if all $|C_i|$ chunks are equal to the chunks contained within C_j, there will be at least one chunk $\{c|\,(c \in C_j) \wedge (c \notin C_i)\}$.

For the case that $|C_i| \geq |C_j|$, the interest can only be given as a probability if no specific chunk set is analysed. To calculate the probabilty of interest for a given peer P_i with a chunk set of $|C_i|$ chunks, another equation will be formed. Let $n_{ci} = |C_i|$ and $n_{cj} = |C_j|$. For any peer P_i, having n_{ci} chunks in its chunk set, the probabilty of another peer P_j having chunks that are not yet in its possession, can be calculated as follows. First, the possiblity $p_{P_j notinterested}$ of P_j drawing only chunks which are already in P_i's chunk set is computed. This can be seen as P_i drawing n_{ci} out of the total number of available chunks N without putting them back. Then P_j draws n_{cj} chunks from the chunks drawn by P_i. The result is then subtracted from one to calculate the probablity of the case in which P_j draws at least one chunk that wasn't drawn by P_i.

$$p_{P_j notinterested} = \frac{\binom{N}{n_{ci}} \cdot \binom{n_{cj}}{n_{ci}}}{\binom{N}{n_{ci}} \cdot \binom{N}{n_{cj}}} = \frac{\binom{n_{cj}}{n_{ci}}}{\binom{N}{n_{cj}}} \tag{4.13}$$

Equation 4.13 can be subtracted from one (see equation 4.14) to compute the probabilty that a peer P_i is interested in P_j.

$$p_{P_j interested} = \begin{cases} 1 & , n_{cj} > n_{ci} \\ 1 - p_{P_j notinterested} & , n_{cj} \leq n_{ci} \end{cases} \tag{4.14}$$

Interest in a Modified BitTorrent System

For the modified mechanism, there are new variables that decide whether a peer can be interested in another peer's chunks or not. For each peer, there exists a withheld chunk set which decides which chunks the neighbour peer withholds from him. The neighbouring peer may or may not have these chunks in its possession. If one wants to check the interest of peer P_A to peer P_B, these withheld chunk set must be considered as well. Let W_{BA} be the withheld chunk set of P_B for P_A. If P_A looks at P_B's chunk set, it can only see those chunks that are not in the withheld chunk set W_{BA}. Thus, P_A will only be interested in the chunks it can see from P_B that it can see. For this, the set is split into two parts and looked at seperately. The visible part of the chunk set of P_B contains $n_v b$ of $N - W_{BA}$ chunks. P_A's interest in P_B can now be expressed in a similar way as in 4.14. P_A's chunk set will be split in the same manner as P_B's because the chunks P_A cannot see will not influence the interest. The resulting equations can be written as follows:

$$p_{P_b notinterested} = \frac{\binom{N - |W_{BA}|}{n_{va}} \cdot \binom{n_{vb}}{n_{va}}}{\binom{N - |W_{BA}|}{n_{va}} \cdot \binom{N - |W_{BA}|}{n_{vb}}} = \frac{\binom{n_{vb}}{n_{va}}}{\binom{N - |W_{BA}|}{n_{vb}}} \tag{4.15}$$

$$p_{P_b interested} = \begin{cases} 1 & , n_{vb} > n_{va} \\ 1 - p_{P_b notinterested} & , n_{vb} \leq n_{va} \end{cases} \tag{4.16}$$

Equations 4.15 and 4.16 are nearly the same as before, except for the changed set sizes $N - |W_{BA}|$. If P_A sends a chunk to P_B, P_B will reward this by removing one of the elements of its withheld chunk set. This means that W_{BA} will be decreased by one. If it owns the chunk that was removed from the set, then the withheld *HAVE* meassage is sent out. If the chunk is not in its chunk set, then the chunk id is only removed from W_{BA} and the *HAVE* message is sent out once P_B obtains the chunk. Now looking at the invisible part of the chunk set, one can calculate the increase of probability of interest $pP_b interested$ if W_{BA} is decreased by one. For this, the number of chunks within the invisible part of the chunk set n_{ib} and n_{ia} will be looked at. If P_B reveals one position of its withheld chunk set W_{BA} to P_A, then the probability that the position is a chunk that is in P_B's chunk set is shown in equation 4.17:

$$p_{brevealchunk} = \frac{n_{ib}}{|W_{BA}|} \tag{4.17}$$

Only in case that P_B has the chunk, P_A's probability of interest will rise. The possibility that the revealed chunk is actually of interest to P_A is calculated using equation 4.18.

$$p_{revealedinterest} = 1 - \frac{\binom{n_{ia}}{1}}{\binom{|W_{BA}|}{1}} \tag{4.18}$$

If peer P_A has sent a chunk to P_B, then the probablity of being interested will rise. This rise can be calculated by combining equation 4.17 and equation

4.18 to form equation 4.19.

$$p_{newinterest} = \frac{n_{ib}}{|W_{BA}|} \cdot \left(1 - \frac{\binom{n_{ia}}{1}}{\binom{|W_{BA}|}{1}} \right) \tag{4.19}$$

The first fraction of equation 4.19 represents the probability that the revealed position contains a chunk that is owned by P_B and the second part will decide whether the newly revealed chunk is of interest to P_A because it is not the same as the chunks that P_A already possesses.

4.2.3 Whitewashing

This section will address the problem of whitewashing. In a system that is based on a temporary attribute, it is important to know how easy it is to change this attribute and if the mechanism is still effective after a peer has whitewashed itself. The reduction of the effectiveness of the proposed mechanism shall be discussed below.

How Whitewashing Works

In the case of the mechanism as it has been proposed in this dissertation, the IP address of a peer serves as the attribute to identify a peer and to compute its withheld chunk set. If a peer uses a dialup connection and redials, a new IP address will be assigned to it and its withheld chunk set will differ from before. This means that it can download chunks that were not available to it

before.

Desired Attributes to Prevent Easy Whitewashing

The effect of whitewashing can be reduced if:

1. whitewashing requires time or effort
2. the whitewashing has to be repeated multiple times to be effective

The time or effort to disconnect a dialup connection, obtain a new address and reconnect to the BitTorrent network are not extremely high but a user has to be present to initiate it if the program does not have the administrative rights to do it on its own. In a larger network, it may even be impossible because several users share one external IP address and the BitTorrent client would have no access to the internet gateway. If effort and time are low, then it would help prevent whitewashing if the process of reconnection had to be repeated several times until a complete download can be obtained.

Withheld Chunk Set Size and Whitewashing

The size of the withheld chunk set has an important effect on whitewashing. If more than half of the chunks of the torrent are contained in the withheld chunk set then one disconnection is not enough to download the whole torrent, even if the free-rider has the luck that two distinct withheld chunk sets are computed from its IP addresses. In reality, the withheld chunk sets may overlap, thus reducing the chances of obtaining a full copy further. If a torrent contains 400 chunks and 250 of them are assigned to a peer's withheld chunk set, then in

the best but very unrealistic case the peer would have to reconnect two times to obtain a full copy of the torrent.

Chapter 5

Evaluation and Results

This chapter will first introduce the design of the simulation and then go on to present the results.

5.1 Design

This chapter will introduce the software design of the chosen solution. Parts of the design were already given by the architecture of the simulator and the basic BitTorrent simulation[EHBK07] on top of it. When designing the simulation extension, the main goal had to be simplicity and to change as little as possible of the original code. The simulation should also be able to simulate original peers, peers using the new algorithm as well as free-riding peers.

Because this is a prototypical proof-of-concept implementation, no refactoring of the existing code has been done, except if the new functionality demanded it or the piece of code had to be rewritten.

NS-2 has been used as the basis of the simulation. It simulates the TCP

network connections between the peers with all its parameters. Methods of the simulation are called, once data has been received. This data is then handled by the simluation. The architecture of this simulation shall be the topic of the following paragraphs.

5.1.1 Piece Withholding

The piece withholding code resides mostly within the `RandomPiece` class. `RandomPiece` uses another class called `SHA1` to provide its functionality. When a new peer is added to the peer list, a modified client will also calculate the withheld chunk set. The chunk ids contained within this set are obtained by calling the `GetPieces()` method of the class `RandomPiece`. Most of the flow control is coded into the `BitTorrentApp` class as it is deeply connected and woven into the BitTorrent protocol functionality. All peers are of the class `BitTorrentApp`. Variables decide whether or not they are seeds or free-riders. These variables can then be used to decide whether or not to call the code that provides the modified behaviour within the `BitTorrentApp` program flow. If one looks at the block diagram in Figure 5.1, the single blocks can also be found within the design class diagram seen in Figure 5.2. Only the most important classes are shown to allow a better overview. The extended and modified `BitTorrentApp` class corresponds to the *BitTorrent Simulation* in the block diagram. *Algorithm Modifications* in the block diagram is partly realized by the `BitTorrentApp` class and partly by the aforementioned `RandomPiece` and `SHA1` classes. Reporting as it can be seen in the block diagram will be the subject of the next section.

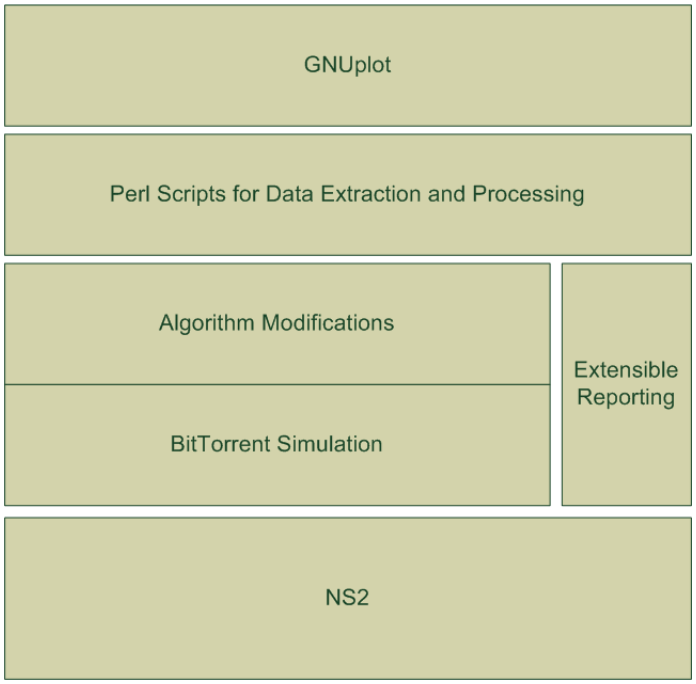


Figure 5.1: Block diagram of the experimental setup

5.1.2 Reporting

The reporting framework that shall be responsible for collecting peer data during a simulation run was designed from scratch and had extensibility as its main requirement. In the beginning of a research, it is not always clear which parameters may become of interest while the research goes on. New measurements might be necessary and it may become obvious that others are not longer needed. Therefore, an extensible reporting framework had to be designed that could be easily adapted to fit the requirements at any time.

Dependency inversion[Mar96] has been used to decouple the high level reporting class from the low level implementations for specialized reports. To do this, an abstract base class `IReporter` has been implemented of which new reporters can be derived. In the beginning, a `TimeReporter`, a `PieceReporter`,

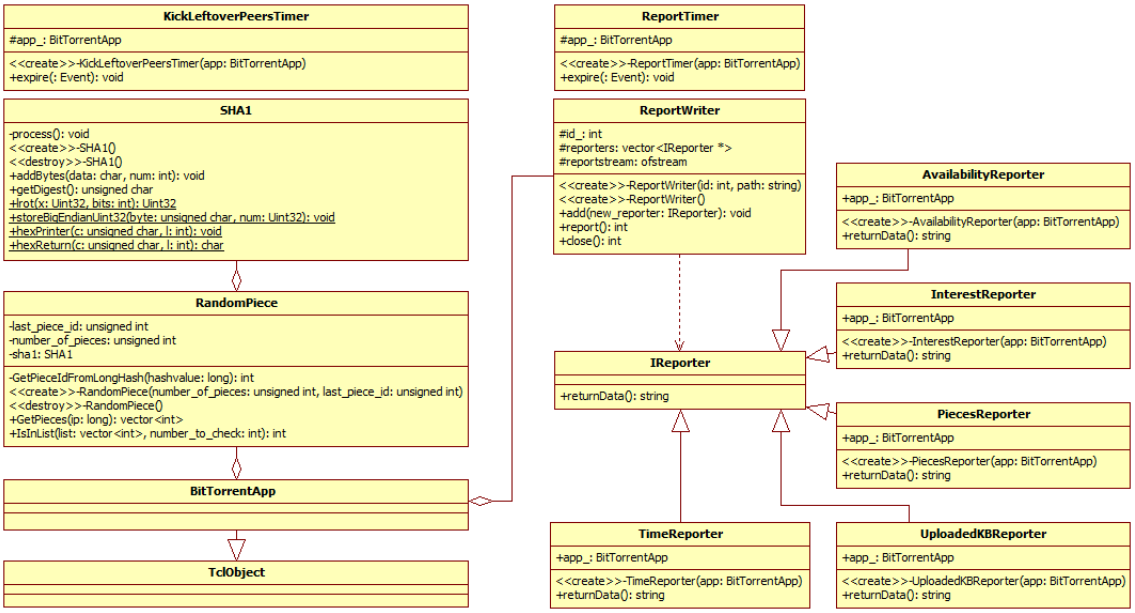


Figure 5.2: Class diagram of the simulator with extensions

and an `InterestReporter` were implemented to be able to retrieve the simulation time, the pieces of a peer, and the number of peers that a certain peer is interested in. Each of the subclasses implements a `returnData()` method in a way that it returns the necessary data as a string. Later in the process of writing this dissertation, an `AvailabilityReporter` and an `UploadedKBReporter` have been added. All without having to change code of the parent class (see Figure 5.2 for more details).

`ReportWriter` is the class responsible for reporting. The `BitTorrentApp` instanciates an object of it and initializes it. The `ReportWriter` object aggregates objects of the type `IReporter` which can be registered by its `add()` method. According to the Liskov Substitution Principle, all derived classes of `IReporter` can now take the place of the parent class. Therefore all kinds

of specialized reporter classes can be aggregated. If new types of reporters are implemented, they can be added just like any other object of the type `IReporter`.

To trigger a report, a timer `ReportTimer` has been put into place. It derives from `TimerHandler`, which is a class of the simulator. The timer overwrites the `expire()` method of its parent which is called after the timer has counted to its desired limit. Within the `expire()` method, the timer now calls the `report()` method of the `ReportWriter` object. Now, all aggregated reports' `returnData()` methods are called and their results written into a CSV file on the harddrive.

5.1.3 Example Simulation Run

A text file is created containing the parameters of the simulation. These include the number of peers, a random seed, the upload capacity of the peers, the number of free-riders and the number of peers using the original algorithm for mixed population tests.

This text file is read by a shell script which runs the simulation script using the parameters. The simulation script will then create simulator objects, set up the network environment of NS-2 and create the peers according to the parameters.

Once the simulation runs, a `BitTorrentApp` object is instanciated. A variable decides what kind of peer it is: seed, downloader or freerider. Another variable decides whether or not the modifications are active. The object will connect to the tracker object that has been instanciated as well, register and

obtain a list of neighbours. The `BitTorrentApp` object will then do a TCP handshake and exchange its bitfield with the neighbours. If it is a mofified peer, then a withheld chunk set is created[1] for each peer it connects to. The sent bitfield will not contain the withheld chunks. In the following time, downloaders will exchange data. Once a modified peer receives a chunk[2] from another peer, it will remove a chunk id from the withheld chunk set of that peer. The choking mechanism is triggered by a timer and decides which peer is allowed to request data. Once a peer has obtained all chunks, the `seed` variable is set to one and it will stay in the system to serve as a seed to others. If all peers have the `seed` variable set to one, the simulation ends. If the modification prevents peers from finishing, a timer called `KickLeftoverPeersTimer` will remove the unfinished peers after the simulation time has beeen exceeded. Data about the download is now written into a CSV-file. The objects of type `IReporter` have a separate timer and continuously write their data into CSV files (one file for each peer).

The shell script that ran the simulations in the background will poll the directories for the simulation CSV files. Once all files are present, the script will continue to call the Perl scripts which calculate additional information such as mean values. All of this data will then be used by another script which calls Gnuplot. Gnuplot in turn then generates plot images in PNG format which can be included in the dissertation. To generate all the plots for the single peers, a special Gnuplot file has been created that contains placeholders like `!!!PLOTFILE!!!` which are replaced by the filename of the source CSV file.

[1] For a flow chart of the cration see Figure 4.3 of chapter 4.1
[2] For a flow chart about how a received chunk is handled see Figure 4.1 in chapter 4.1

If one wants to plot all peers' interest values over the time, then the following shell script is used to generate all images:

```
for f in ‘ls report*‘;
do for i in $f; do
cat ../plotinterest.plt | sed "s/!!!PLOTFILE!!!/${i}/g" | gnuplot;
done;
done.
```

This script will loop over all reports, feed the plot commands into Gnuplot while replacing the placeholder with the filename.

5.2 Results

This section will introduce the results of the simulation. For each result, the conditions and parameters will be introduced. Afterwards, the results are presented, explained, and discussed.

Different measurements have been conducted to analyse the effectiveness of the proposed mechanism and to compare it to the unmodified mechanism. In chapter 4.2, two aspects have been identified and discussed which are affected by the proposed modification: Availability and interest. These will now be discussed with the help of simulation results. Furthermore, measurements of metrics are presented. Among those are the download time and the amount of uploaded bytes of a peer. Tests were done for pure population (only original peers or only peers using the modified algorithm), pure population plus free-riding peers, and mixed population in which old and new peers are put together in one swarm. The mixed population tests also include free-riders.

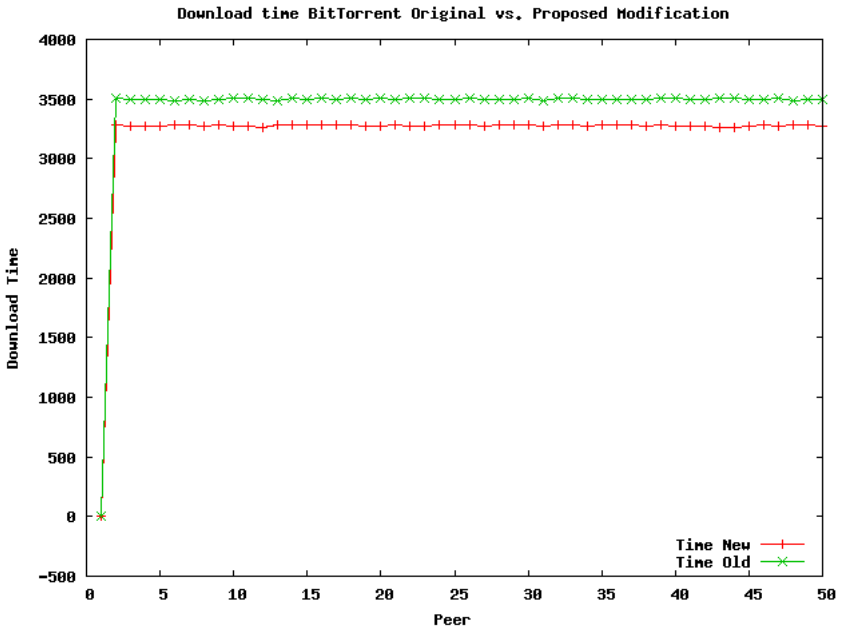
5.2.1 Pure Population

This section will present the measurement results of the pure population simulations. Pure population means that there are either only new or only original peers in a swarm. Free-riders will be added to show the effect they have on the metrics of the other peers.

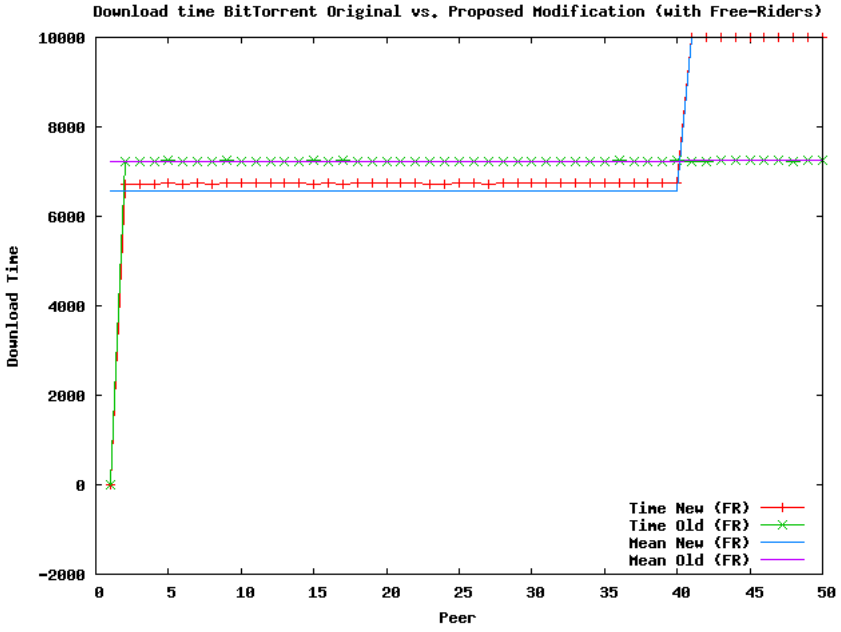
The first simulation had the following scenario: 50 peers download a file of 100MB size (400 chunks). Peer 1 is a seed. All peers join the network at the same time in a so-called flash-crowd. This is the worst possible situation because the availability is only provided by the initial seed. The proposed mechanism has an influence on the availabiltiy and in this situation it can show that it can also overcome this worst case. In reality, peers come and go during the process of downloading. Each peer has an upload rate of 512 kBit/s, which is a normal rate for nowadays asynchronous DSL[3] connections. The download rate is not limited. 50 chunks are withheld by the proposed mechanism. Because the modified algorithm is designed to prevent the free-riders from finishing the download, the simulation will stop after 10,000 seconds.

Figure 5.3a and Figure 5.3b show the simulation results of the original BitTorrent mechanism compared to the modified mechanism with the parameters described above. Figure 5.3a shows a swarm in which there are no free-riders while 5.3b was created as a result of 40 normal peers and 10 free-riders. Both, Figure 5.3a and 5.3b, show one line for the original algorithm (green) and the modified algorithm (red). Figure 5.3b contains two more lines that show the average for each kind of peer: free-rider and downloader. The X-axis shows

[3] **D**igital **S**ubscriber **L**ine



(a)



(b)

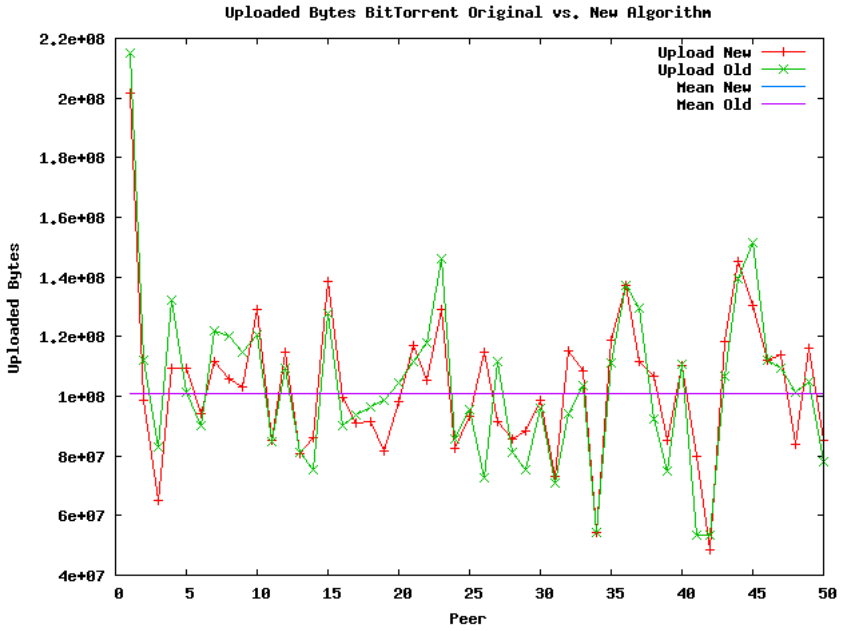
Figure 5.3: Download time: Pure population Original vs. Modified without free-riders (a) and with free-riders (b) (50 withheld)

the peer ids. Peer 1 in all plotted lines is a seed. The highest ids are assigned to the free-riders. In the simulation with free-rdiers this means that the peers with ids 41 to 50 are free-riding.

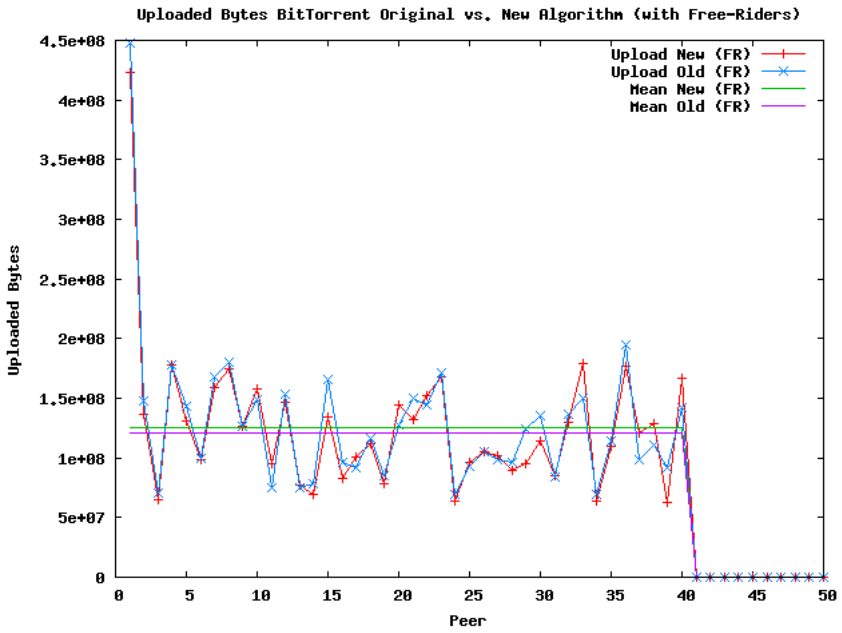
In Figure 5.3a, one can see that all peers finish their download at approximately the same time. Peers using the modified algorithm as it is proposed within this master thesis show a shorter download time than those using the original algorithm. This can be explained by the situation in which the measurement is taken. Only the seed has the whole file and thus the availability is bad. For such a situation, the original BitTorrent uses a feature called *super seeding* in which the seed will first show only a part of its chunks to each peer so that the chunks are not randomly chosen and a whole copy is quickly spread within the system. This mechanism shares a similarity with the proposed mechanism in that it reduces the chunks that a peer can see. The proposed mechanism will also result in a better initial spreading of chunks. It is important to notice is that the use of the new mechanism does not negatively influence the download time of peers.

When looking at Figure 5.3b, it can be seen again, that the modified clients beat the old clients in download time. Also, free-riding is not punished by the original clients. The peers with ids 41 to 50 finish together with the original peers. The modified clients on the other hand show a lower download time. A part of the bandwidth of the original clients is "stolen" by the free-riders. In the population containing only new peers and free-riders, $\frac{1}{8}$ of the upload is not wasted to the free-riders. Also the free-riders' download time is cut off at 10,000 seconds when the simulation was stopped. No free-rider has finished

its download at this time.



(a)



(b)

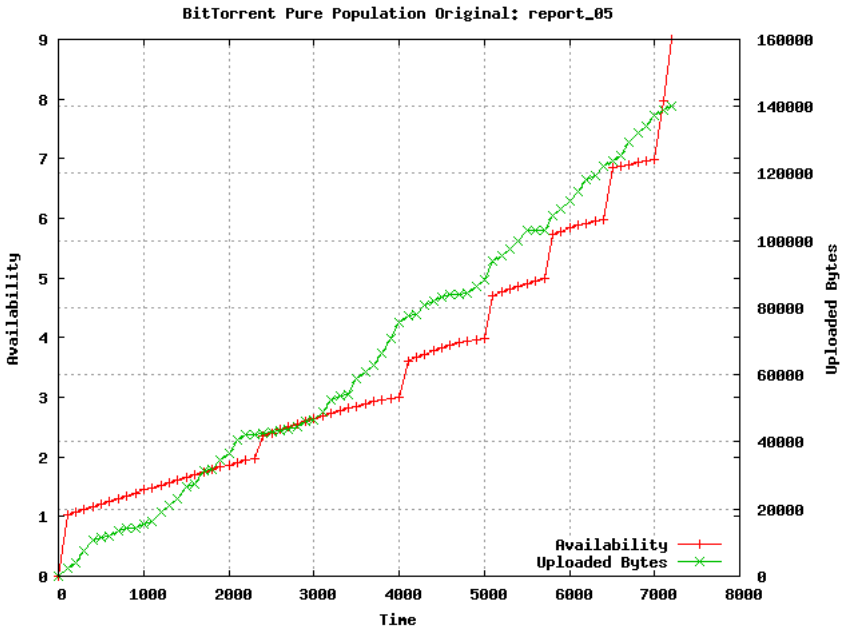
Figure 5.4: Uploaded bytes: Pure population Original vs. Modified without free-riders (a) and with free-riders (b) (50 withheld)

Figure 5.4a and Figure 5.4b show the uploaded bytes of the original and the modified clients in the simulation. The colours correspond to the colours in 5.3. Both images include a mean value to allow for a better comparison.

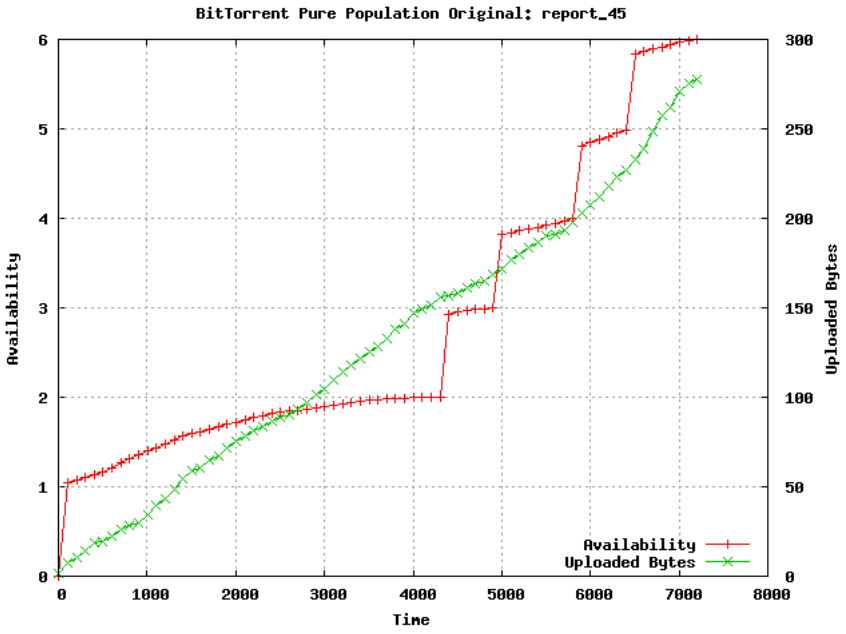
In Figure 5.4a, it can be seen that in average, both kinds of clients upload the same amount. This is because BitTorrent is a closed system and like in physics, where the law of conservation of energy applies, every upload to the system must also be the download of someone. Because the downloaded bytes are the number of bytes of the torrent times the downloaders, the total upload must also equal the total download. Hence the average values are equal. The new algorithm shows a slightly higher average because the seed is not counted. In Figure 5.4b, the average of the new clients is higher. The seed wastes less of its bandwidth to the free-riders. As a consequence, the uploaded bytes of the downloaders rises.

An important criterion for a torrent is its availability. The availability is different for each peer and is calculated according to equation 4.10 (see chapter 4.2.1). Because in the modified algorithm, some chunks are hidden from a peer, the availability is influenced. To illustrate this, the availability from the point of view of a peer has been recorded during the simulation. An example of the availability for an old client is shown in Figure 5.5a

Figure 5.5a shows the availability and the uploaded bytes over the time from the point of view of an original client. Figure 5.5b shows the same metrics from the point of a free-riding peer in the same simulation. Both kinds of peers have a similar availability. The free-rider, of course, has no upload at all (not even 300 bytes).



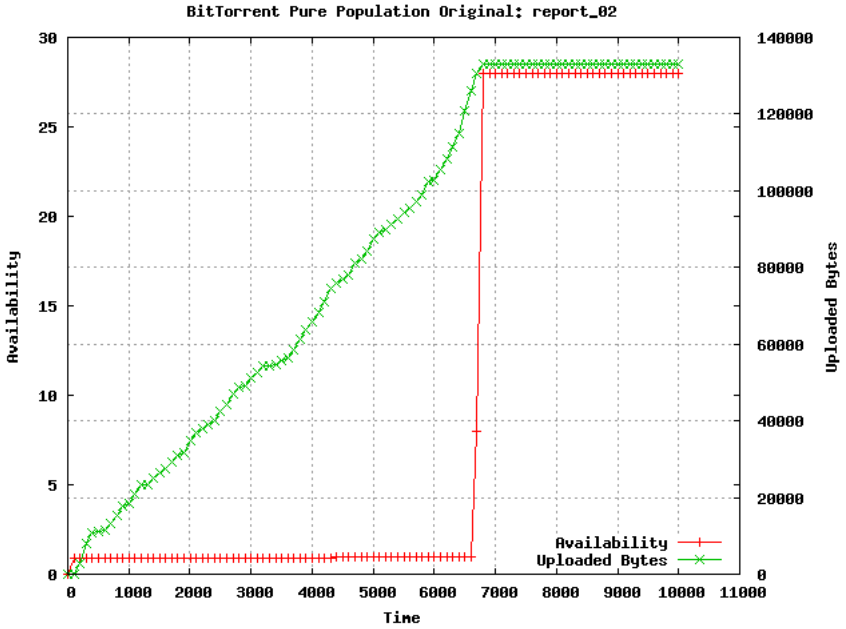
(a)



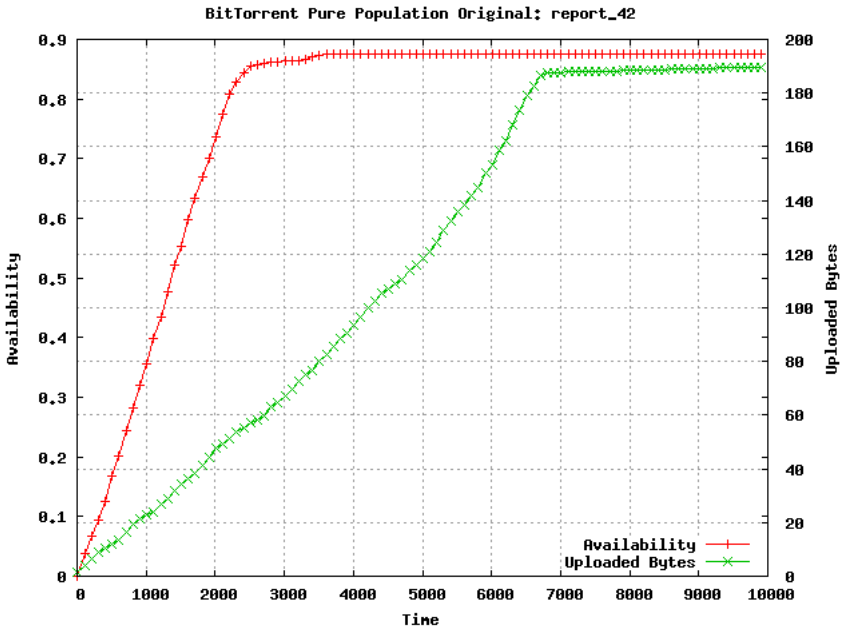
(b)

Figure 5.5: Results of availability and uploaded bytes of an original client (a) and a free-rider (b) (50 withheld)

The next images (Figure 5.6a and 5.6b) show a modified peer and a free-rider.



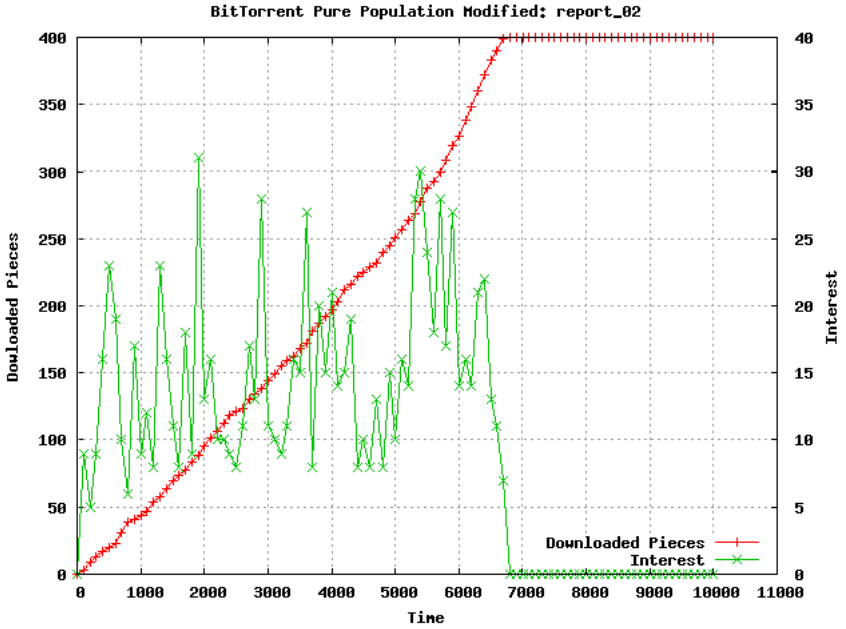
(a)



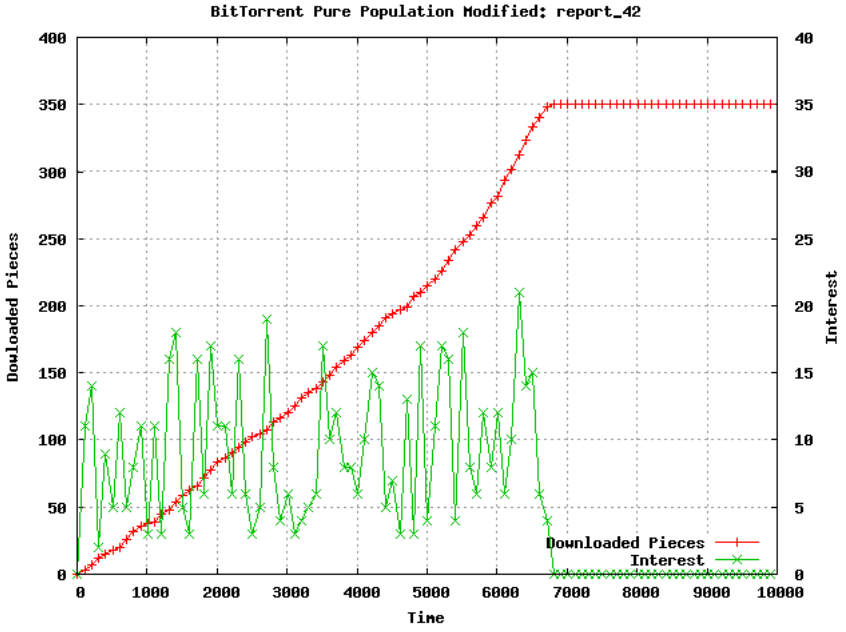
(b)

Figure 5.6: Results of availability and uploaded bytes of a modified client (a) and a freerider (b) (50 withheld)

Figures 5.6a and 5.6b show the uploaded bytes and the availability for a modified client and a free-rider. Figure 5.6a shows that the availability is always not as high as in an original client but it shortly reaches one as the peer is seeing enough chunks. A sharp increase in availability was recorded when the neighbours reached seed status. Figure 5.6b on the other hand shows that the availability never reaches 1. This means that the peer won't be able to obtain the missing chunks and is not be able to finish its download. Both Figures show that there is no upload activity after the normal peers have finished their download and stay in the system as seeds. The free-riders can't profit from the seeds because the chunks they need are not revealed to them. From the point of view of a free-rider, the system is filled with peers that have downloaded all but the chunks that are withheld from it.



(a)



(b)

Figure 5.7: Results of downloaded chunks and interest of a modified client (a) and a freerider (b) (50 withheld)

Figures 5.7a and 5.7b show the downloaded chunks and the interest of two selected peers over the time. Figure 5.7a depicts the data of a peer using the proposed mechanism and Figure 5.7b that of a free-rider. One can see that the number of peers in which the free-rider shows interest is lower on average than that of the modified peer. The interest of the modified peer stops as soon as it downloads the last chunk and becomes a seed. The free-rider does not finish its download and has no interest in anyone because the chunks it needs to complete the download are not revealed to it.

It becomes even clearer if more chunks are withheld. Figure 5.8 shows first the download time of 40 modified clients with 10 free-riders with 250 chunks withheld. One can see that the torrent is downloaded in less time because the peers are able to reveal enough chunks while downloading to keep the speed up and not so much bandwidth is wasted to free-riders. Figures 5.9a and 5.9b show the interest and downloaded chunks of a downloader and a free-riding peer for this configuration.

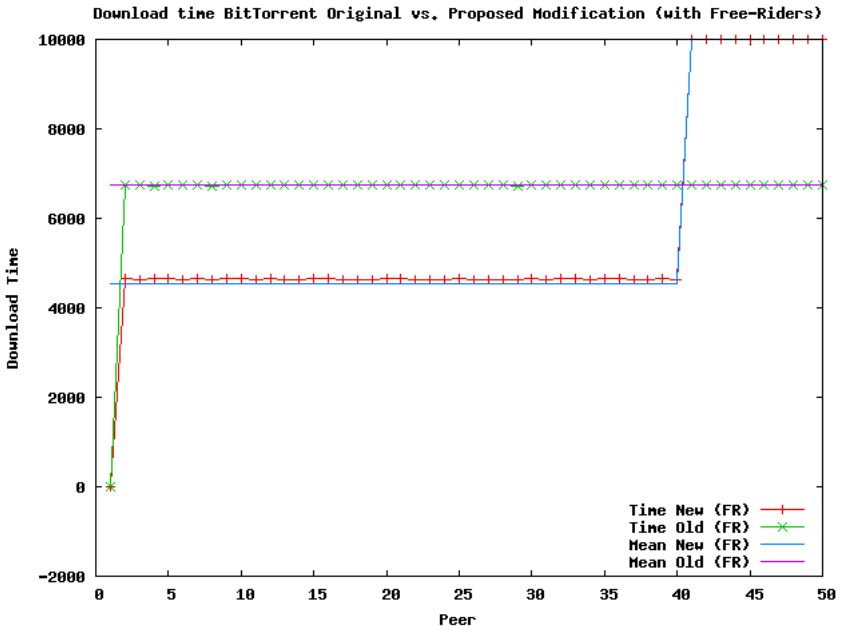
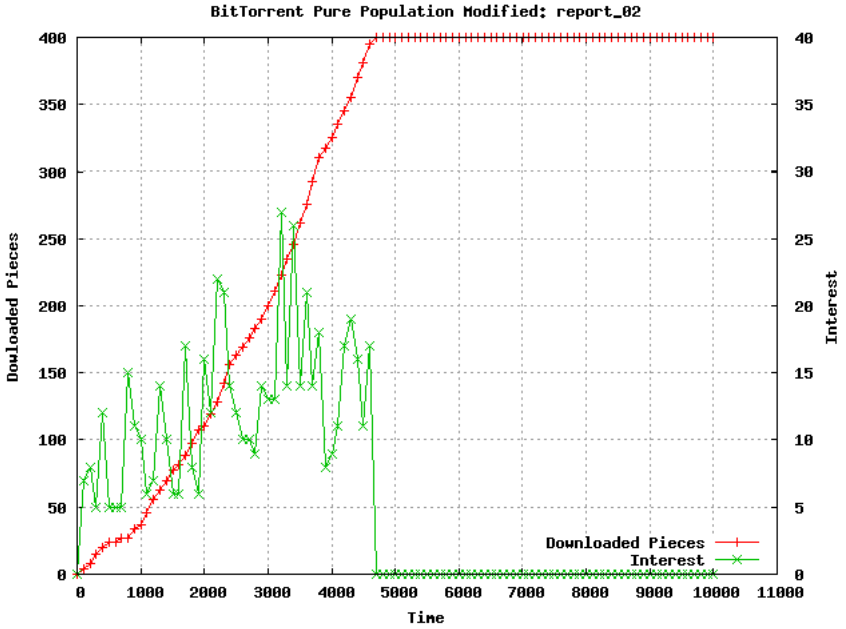
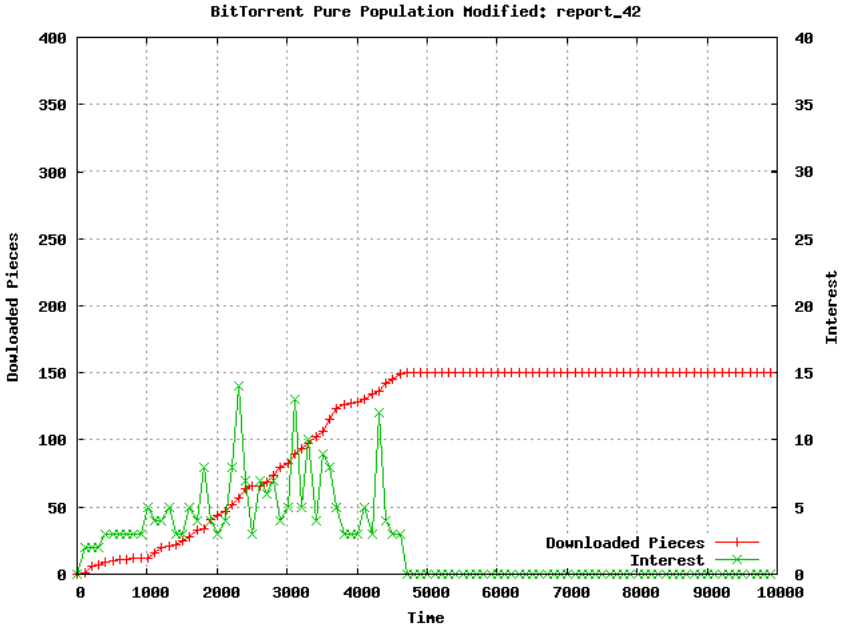


Figure 5.8: Download time: Pure population modified clients with free-riders (250 withheld)



(a)



(b)

Figure 5.9: Results of downloaded chunks and interest of a modified client (a) and a freerider (b) (250 withheld)

Looking at the availability from the point of view of the free-rider, one can see that it is increasing up to $\frac{N-|W|}{N}$, N being the number of chunks of the torrent (400) and $|W|$ the amount of withheld chunks (250). The resulting availability from the point of view of the free-rider is $\frac{N-|W|}{N} = \frac{400-250}{400} = 0.375$. This is confirmed by the measurements plotted in Figure 5.10.

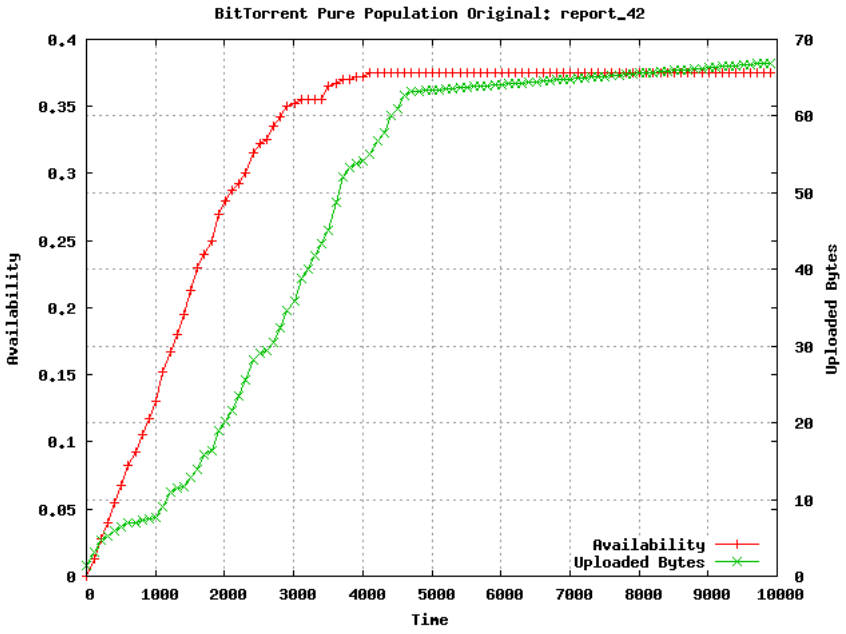


Figure 5.10: Availability from the point of view of a free-rider in a swarm of modified peers(250 withheld)

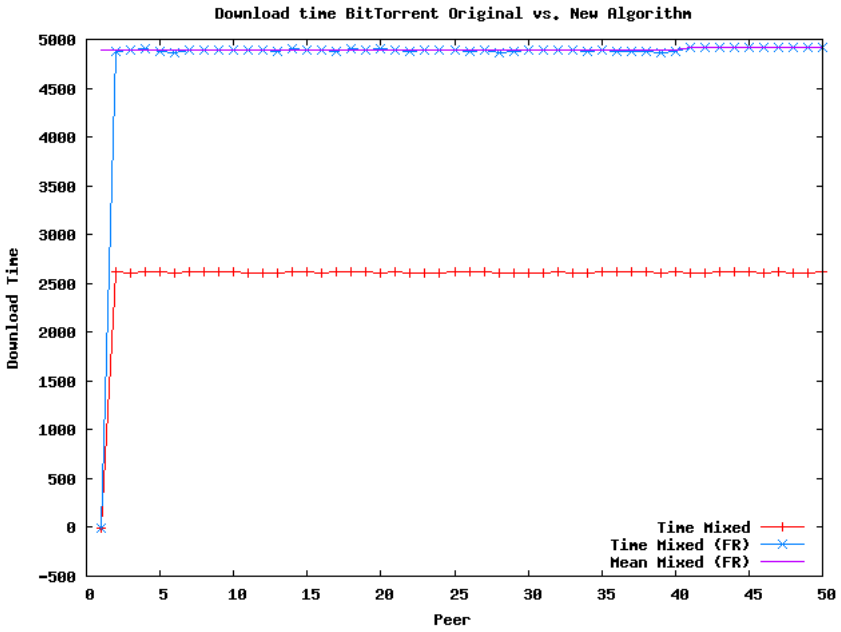
5.2.2 Mixed Population

For the proposed mechanism to be successful, it is vital that it can be used at the same time as the old mechanism is still in place. It would be unrealistic to believe that the variety of clients could be replaced overnight. Other BitTorrent modifications that aim at preventing free-riding fail to be able to run in parallel because the modification require major changes to the way peers interact.

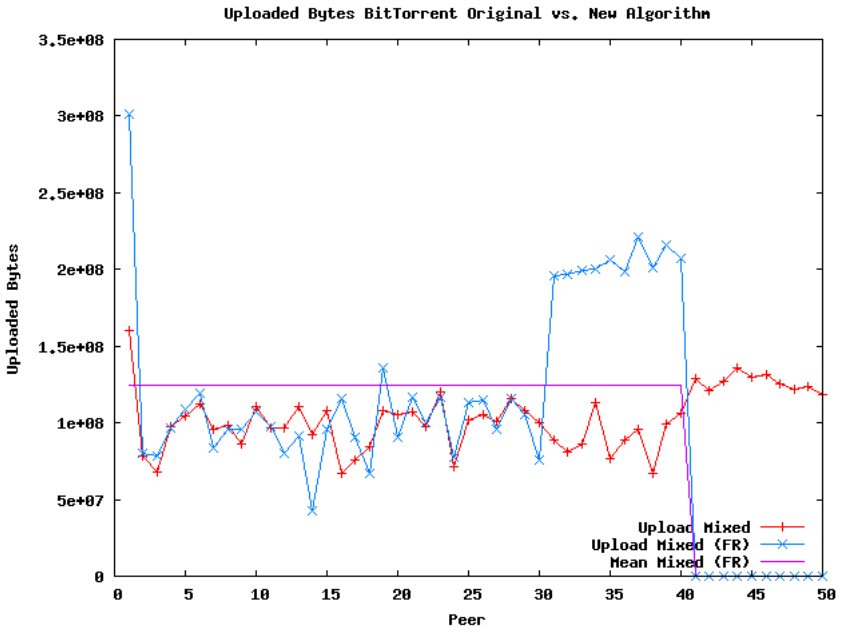
An example would be the central server of [LYWM07] or the bootstrapping process of [LLSB08] that requires three peers to interact in a way that was not designed into the original protocol. To prove that the modified peers can interact with the original peers, several simulations have been run which will show the effectiveness of the proposed modification and its advantage over the original clients.

The simulation parameters will be the same as in the previous section. Within this section, a number of original clients will be put in a swarm together with modified clients and free-riders. The peer ids of the free-riders are still 41 to 50, the original clients preceed them. The first peer ids (including the initial seed) are modified clients. 250 chunks are withheld.

From Figure 5.11a, it can be seen that the proposed mechanism does not manage to prevent free-riding effectively. The download time is increased only marginally. Looking at Figure 5.11b, the reason for this becomes clear. The original clients upload significantly more than the modified clients. This is because they have to provide the free-riders with all the withheld chunks. This means that modified peers are not giving away as much upload to free-riders. Original clients will have to fill the gap. This will, in the long run, provide an incentive to users to use the modified clients where their altruism benefits the system and can't be exploited by free-riders to such an extent.



(a)



(b)

Figure 5.11: Mixed population (30 modified, 10 original, 10 free-riders): Download Time (a) and Uploaded Bytes (b) (250 withheld)

Chapter 6

Conclusion and Future Work

This chapter will present the conclusions drawn from the analysis and results and provides an overview of the further work.

6.1 Conclusion

The simulation results clearly show that a free-rider cannot finish its download in a swarm in which all peers run a modified client. Two metrics are heavily influenced by the proposed modification: Availability and interest. A user who is joining a swarm will have a look at the availability information and the number of seeds the client provides and decide whether this torrent is healthy enough to stay or not. With the modified BitTorrent clients in the swarm, the torrent will at first look as if there would be no possiblility to obtain a full copy of it. This will change gradually as the user's client exchanges pieces with its neighbouring clients. For a free-rider on the other hand, the view will not change. It will always seem as if there is no full copy within the system.

Seeing this, a user can either stay in the system and wait until the download is stalled, or it can disconnect and leave in search for a more promising target to free-ride. The research of [HP05] shows that there is an evolutionary element within BitTorrent swarms in which the "fittest" swarms survive because the ones that do not look promising enough will be left. This evolutionary element can not only help to provide an incentive to free-riders to leave swarms that use the approach described in this dissertation. It can also result in a wider distribution of the modified clients because users are not willing to provide upload to free-riders.

When compared to the desired attributes described in chapter 3.3, one can say that the proposed mechanism is **Nice**, because in the beginning, it will send data to its neighbour, **Retaliatory**, because when the neighbour does not send data in return, it will eventually stop sending data. One weakness is that the retaliation does not follow immediately. The mechanism is **Forgiving** because for each chunk a punished player sends, another chunk will be revealed to it and normal exchange is possible again. It is **Clear** because a peer can calculate its own withheld chunk set and knows that there are chunks withheld from it and that sending of data will increase its chances of obtaining a complete copy.

Game theory has been used to analyse the probable strategies of the players (users). Because rational players, accoring to game theory, will always choose the strategy that will yield the highest payoff, an incentive has been introduced and analysed which that make it impossible for peers to apply a free-riding strategy at all times. Although a peer can whitewash itself by reconnecting, the process would have to be repeated to be successful. But game theory

shows also a weakness of the modification. If a peer is in possession of all the chunks that are in its initial withheld chunk set, then there is no incentive to contribute anymore. A rational player would switch back to downloading without uploading. The proposed mechanism fails to ensure complete fairness but provides a measure to enforce a minimum amount of contribution.

6.2 Future Work

The proposed BitTorrent modification has proven to be effective in a simulation. A next step could be to modify a popular BitTorrent client and do real world measurements.

For now, the size of the withheld chunk set has been predefined and compiled into the client. It has been shown that the set can contain even more than half of the chunks of the torrent. For the future, the withheld chunk set could be calculated dynamically according to parameters like the size of the torrent. Because the number of peers in the swarm also has an effect on the number of chunks that can be withheld, this number could also be part of the dynamic calculation.

References

[AH00] Eytan Adar and Bernardo A. Huberman. Free riding on gnutella. *First Monday*, 5, 2000.

[AMCB04] N. Andrade, M. Mowbray, W. Cirne, and F. Brasileiro. When can an autonomous reputation scheme discourage free-riding in a peer-to-peer system? In *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on*, pages 440–448, 2004.

[Axe84] Robert Axelrod. *The Evolution of Cooperation*. Basic Books, 1984.

[BAS03] Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. A game theoretic framework for incentives in p2p systems, 2003.

[BH05] Ashwin R. Bharambe and Cormac Herley. Analyzing and improving bittorrent performance, 2005.

[EHBK07] K. Eger, T. Hofeld, A. Binzenhfer, and G. Kunzmann. Efficient simulation of large-scale p2p networks: Packet-level vs. flow-level simulations, 2007.

[Gut97] Joel M. Guttman. The explanatory power of game theory in international politics: Syrian-israeli crisis interactions, 1951-87. *Economics and Politics*, 9(1):71–85, 03 1997.

[Har68] G. Hardin. The tragedy of the commons. *Science*, vol. 162 no. 3859:1243–1248, 1968.

[HP05] David Hales and Simon Patarin. How to cheat bittorrent and why nobody does, 2005.

[ipo07] ipoque. internet study 2007: data about p2p, voip, skype, file hosters like rapidshare and streaming services like youtube. `http://www.ipoque.com/resources/internet-studies/internet-study-2007`, 2007. This is an electronic document. Date of publication: May 12, 2007. Date retrieved: July 26, 2009.

[IUKB+04] M. Izal, G. Uroy-Keller, E.W. Biersack, P. A. Felber, A. Al Hamra, and L. Garces-Erice. Dissecting bittorrent: Five months in torrent's lifetime, 2004.

[JA05] Seung Jun and Mustaque Ahamad. Incentives in bittorrent induce free riding, 2005.

[KSGm03] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-molina. The eigentrust algorithm for reputation management in p2p networks. In *in Proceedings of the 12th International World Wide Web Conference*, 2003.

[LFSC03] Kevin Lai, Michal Feldman, Ion Stoica, and John Chuang. Incentives for cooperation in peer-to-peer networks, 2003.

[LLKZ07] Arnaud Legout, Nikitas Liogkas, Eddie Kohler, and Lixia Zhang. Clustering and sharing incentives in bittorrent systems. In *SIGMETRICS '07: Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 301–312, New York, NY, USA, 2007. ACM.

[LLSB08] Dave Levin, Katrina LaCurts, Neil Spring, and Bobby Bhattacharjee. Bittorrent is an auction: Analyzing and improving bittorrent's incentives, 2008.

[LMSW06] Thomas Locher, Patrick Moor, Stefan Schmid, and Roger Wattenhofer. Free riding in bittorrent is cheap. In *In HotNets*, 2006.

[LYWM07] Minglu Li, Jiadi Yu, Jie Wu, and Senior Member. Free-riding on bittorrent-like peer-to-peer file sharing systems: Modeling analysis and improvement. *IEEE Transactions on Parallel and Distributed Systems*, 2007.

[LZL08] W. Sabrina Lin, H. Vicky Zhao, and K.J. Ray Liu. A game theoretic framework for incentive-based peer-to-peer live-streaming social networks, 2008.

[Mar96] Robert C. Martin. The dependency inversion principle. *The C++ Report*, 1996.

[MLLY04] Richard T. B. Ma, Sam C. M. Lee, John C. S. Lui, and David K. Y. Yau. Incentive and service differentiation in p2p networks: A game theoretic approach, 2004.

[Moo06] Patrick Moor. Free riding in bittorrent and countermeasures. Master's thesis, Distributed Computing Group, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology Zurich, 2006.

[MPES08] J.J.D. Mol, J.A. Pouwelse, D.H.J. Epema, and H.J. Sips. Free-riding, fairness, and firewalls in p2p file-sharing, 2008.

[MPM+08] J. J. D. Mol, J. A. Pouwelse, M. Meulpolder, D. H. J. Epema, and H. J. Sips. Give-to-get: free-riding resilient video-on-demand in p2p systems. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 6818 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, 2008.

[NS2] NS2. Network simulator 2. `http://www.isi.edu/nsnam/ns/`. This is an electronic document. Date retrieved: August 12, 2009.

[QS04] Dongyu Qiu and R. Srikant. Modeling and performance analysis of bittorrent-like peer-to-peer networks, 2004.

[Sch04] Walter Schlee. *Einführung in die Spieltheorie*. Vieweg + Teubner, 2004.

[SHRY07] Michael Sirivianos, Jong Han, Park Rex, and Chen Xiaowei Yang. Free-riding in bittorrent networks with the large view exploit. In *In IPTPS 07*, 2007.

[Teo07] Wei Ling Michele Teo. A bittorrent implementation and simulation, 2006/2007.

[Wik10] Vuze Wiki. Scrape. `http://wiki.vuze.com/w/Scrape`, 2010. This is an electronic document. Date of publication: November 25, 2005. Last edit: March 03, 2010. Date retrieved: March 05, 2010.