Sharing Incentives for Lazy Free-Riders in BitTorrent and the BarterCast Reputation System

Arvind Ganga

Sharing Incentives for Lazy Free-Riders in BitTorrent and the BarterCast Reputation System

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Arvind Ganga
born in Leiden, the Netherlands

Algorithmics Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
`www.ewi.tudelft.nl`

Sharing Incentives for Lazy Free-Riders in BitTorrent and the BarterCast Reputation System

Author: Arvind Ganga
Student id: 9249526
Email: a.k.r.ganga@gmail.com

Abstract

A well-known problem in peer-to-peer networks is *free-riding*, where users do not share resources in return for what they consume. Free-riders can be distinguished in two categories: *die-hard free-riders* that are willing to subvert the network's protocol in order to free-ride, and *lazy free-riders* that are reluctant to share but do follow the protocol. An important body of research focuses on die-hard free-riders in the popular BitTorrent file-sharing network, but in practice die-hard free-riding in BitTorrent is not often observed. Lazy free-riding, on the other hand, is often observed, and in this thesis we investigate whether BitTorrent provides lazy free-riders with an incentive to share. Based on a game-theoretical model, we prove that this is the case for some lazy free-riders, but not for all. We then proceed to investigate the same for BarterCast, a new distributed reputation mechanism that is added to the BitTorrent-based Tribler network to provide additional sharing incentives. Based on an extended version of the same model, we prove that BarterCast also provides incentives only to some lazy free-riders, but not for all. We verify these results with simulations, and find that in practice, even fewer incentives are given than our model predicts. However, we show that lazy free-riding can provide a gain but also a loss, and that the net result is difficult to predict, which can be seen as an additional incentive against free-riding.

Thesis Committee:

Chair:	prof. dr. C. Witteveen, Faculty EEMCS, TU Delft
Committee Member:	dr. ir. J.A. Pouwelse, Faculty EEMCS, TU Delft
University Supervisor:	dr. M.M. de Weerdt, Faculty EEMCS, TU Delft
University Supervisor:	M. Meulpolder MSc., Faculty EEMCS, TU Delft

Preface

After many months of hard work, I am more than happy to present this master thesis. Even though it is quite different from what I originally intended to do, I am pleased with the result and I think it is an interesting piece of research.

Of course, this thesis would not have been possible without the help of others. First, I want to thank my supervisors Mathijs de Weerdt, Michel Meulpolder, and Léon Planken. Mathijs I thank for his always positive and motivating guidance. His in-depth knowledge of any topic that came up was both helpful and impressive. Léon was my supervisor for the first part of writing this thesis. His detailed comments and clear understanding of the theory were very valuable. Michel supervised me for the second part of writing, and his in-depth knowledge of both BitTorrent and BarterCast and his practical approach to research were of great value for this thesis.

I would like to thank Cees Witteveen and Johan Pouwelse in advance for reading and assessing my work. I also want to thank my employer, TOPdesk, for allowing me to work part-time while doing this project. Finally, I thank my family and friends for their support.

Arvind Ganga
Delft, the Netherlands
May 31, 2010

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

One of the major applications of the Internet is the exchange of files between people. It is safe to say that anything that can be binary encoded – from research documents to movies and music – has by now been transferred over the Internet. Early transfer methods include e-mail, FTP, or downloading the file from a web page. In 1999 Napster introduced a new method: peer-to-peer (P2P) file-sharing. The basic idea is that a user interested in sharing or downloading files runs a software program that connects to other users, so that all users together form a network where every user can download files from any other user. Napster became popular quickly, with over 20 million concurrent users, but collapsed due to legal issues. Many other P2P file-sharing networks followed, and by now file-sharing is responsible for a major portion of all Internet traffic: up to 57% in some geographic regions [2].

Even though file-sharing networks are very popular, they face some technological issues which are areas of active research. Examples are how the network is structured, and searching for content in the network. One major area of research, however, is more psychologically motivated: it turns out that many users are willing to download files, but not to share files with the other users. This phenomenon is referred to as *free-riding*. In a P2P file-sharing network, users download directly from each other, and if users do not share files, the network has no files to download. Most file-sharing networks address free-riding through technological means, but recently more social approaches are investigated.

The BitTorrent file-sharing network is the most popular file-sharing network today. This is partly due to the way that it addresses free-riding: by using a technological mechanism that forces users to share a file while it is being downloaded. Research has shown that this mechanism can be circumvented [46, 35, 26]. In other words, it is possible for a user to download without sharing, or to download faster, thereby staying online sharing for a shorter period of time. This requires the user to install a modified client for the BitTorrent network. In practice, it turns out that users hardly do this [28]. This is not to say that users are indifferent about sharing or their download performance. Most clients come with many settings, and message boards on the Internet are full of questions on which settings yield optimal performance.

Following Meulpolder et al. [28], we refer to free-riding users that go to such lengths as installing modified clients in order to free-ride as *die-hard free-riders*, while we refer to users that are only willing to tweak their settings as *lazy free-riders*. Most current research in BitTorrent focuses on die-hard free-riders, and investigates technical solutions that circumvent free-riding prevention mechanisms. This is an important area of research, as it shows how these mechanisms can be improved upon. However, as we describe above, the resulting clients are not widely adopted by BitTorrent's user base. In this thesis, we therefore focus on lazy free-riders, and the effectiveness of the free-riding strategies these users have at their disposal. We investigate this for BitTorrent as this is the most popular file-sharing network today, and for BarterCast [28], an additional free-riding prevention mechanism that was recently added to BitTorrent by the Tribler [37] team at Delft University of Technology.

We formulate the central research question for this thesis as:

Do BitTorrent and BarterCast provide incentives to lazy free-riding users to share?

To answer this research question, we develop a model of BitTorrent which allows us to study what we call the *outcome* of a BitTorrent network. This outcome describes which users download from which other users, and directly determines any user's download performance. By investigating how the outcome changes for different free-riding strategies, we then determine the effect on the user's download performance. Following that, we extend the model with a model of BarterCast, and study the additional free-riding strategies that BarterCast offers. We verify the results from these studies with experiments.

1.2 Outline

We provide background information on P2P file-sharing networks, and BitTorrent in particular, in Chapter 2. There, we also introduce the techniques we use in our model of the BitTorrent network. The model itself we develop in Chapter 3. In Chapter 4, we investigate BitTorrent's outcome using our model, and investigate theoretically whether lazy free-riding can improve a user's performance. In that chapter, we extend our model with BarterCast, and investigate the effect of lazy free-riding in BarterCast. We verify the results from Chapter 4 experimentally in Chapter 5. Finally, Chapter 6 draws conclusions and identifies directions for future research.

Chapter 2

Preliminaries

Chapter 1 introduces peer-to-peer file-sharing networks, and their main problem: free-riding, where users profit from the network but do not contribute in return. We also state that in this thesis, our focus is on how individual users can improve on their download completion time by strategically specifying their settings in the BitTorrent file-sharing network. We investigate these questions in the following chapters. In this chapter, we provide preliminaries and an overview of the literature that is relevant to this central theme.

We start with a short overview of P2P file-sharing networks in Section 2.1. Here, we identify some key characteristics of file-sharing networks. Section 2.2 discusses one specific topic prevalent in the BitTorrent file-sharing network: how files are distributed between the peers. This topic is of central importance in this thesis as it allows us to answer our research questions stated in the previous chapter. We investigate this using a model based on game theory, a field we introduce in Section 2.3.

2.1 P2P File-Sharing Networks

In this section, we present some key characteristics of P2P file-sharing networks. We elaborate on free-riding, one of the major problems in P2P networks, and provide an overview of proposed solutions to prevent free-riding.

2.1.1 Introduction

File-Sharing Networks Overview

A peer-to-peer (P2P) network is a network in which the participants, the peers or agents[1], are directly connected to each other. Peers have an equal role, acting both as clients and servers. This is different from the more traditional client-server model, where communication is usually to and from a central server.

Although peers in P2P networks have an equal role, the networks may contain central components, such as a superpeer maintaining an index of connected peers. Such networks are referred to as *centralized* P2P networks. The central components in these networks are often considered problematic because they need to be maintained and paid for, and are a single point of failure for the network: when the central component is unreachable, the whole network is down. A P2P network that does not have any centralized components is called a *distributed* P2P network.[2]

[1]In this thesis, the terms peers and agents are used interchangeably.

[2]Distributed P2P networks are also referred to as *decentralized* or *pure* P2P networks.

P2P networks have many applications; examples are distributed computing, task allocation, and resource scheduling. In this thesis, our focus is on distributed P2P *file-sharing* networks, which allow peers to exchange files.

The first P2P file-sharing network was Napster, which grew to 25 million users in the first 12 months after its introduction in 1999. Napster clearly illustrates the problems inherent in a centralized network. It maintained a central catalogue of available files in the network, which was shutdown following a lawsuit filed by the Recording Industry Association of America, which led to the demise of the network as a whole [23].

Many networks that have emerged following Napter's demise adopted a decentralized or hybrid approach to reduce both legal and technical risks from the loss of a central server, and to reduce the monetary investment required to operate such a server. Examples are Gnutella, Kazaa, and BitTorrent. Soon after its release, a study by Adar and Huberman [4] showed that 70% of Gnutella's users were *free-riding* (not uploading any content) and nearly 50% of all responses were returned by the top 1% of all sharing hosts. From these numbers, it is clear that the basic principle of all users acting both as servers and clients was not adhered to.

The problem of free-riding is not unique to Gnutella, but arises in all file-sharing networks. As one user's download is another user's upload, the network offers no content if all users free-ride. A substantial part of research in P2P file-sharing networks therefore concentrates on free-riding, and mechanisms to prevent it.

Incentives to Prevent Free-Riding

Feldman et al. [18] show, based on a simple economic model, that a P2P network can tolerate a certain fraction of free-riders, but that it collapses if this fraction exceeds some threshold which depends on the level of churning (how often users join or leave the network) and the generosity in the user population. Mechanisms to prevent free-riding are required to keep the fraction of free-riders below that threshold, but a mechanism that eliminates free-riding completely may do more harm than good as it may impose burdens on all users that render the system less attractive to non-free-riders. Every free-riding prevention mechanism must therefore select an optimum between its effectiveness in preventing free-riding, and the cost it incurs on its users.

The main problem with preventing free-riding in a P2P network is that the peers are autonomous: the client to the network is a program that runs on the user's computer, and nothing in the network can prevent the user from e.g. specifying unwanted settings, or installing a malicious client [31]. This is similar to the real world, where people are autonomous and make their own decisions. Where in the real world, the law prevents people from behaving maliciously, this is no option in P2P networks because peers can easily switch their identity which makes them untraceable [18]. As an alternative, most research focuses on offering the peers incentives to motivate them to contribute to the system.

Many networks adopted a tit-for-tat mechanism to offer incentives: peers must upload files in order to be allowed to download files. This is often referred to as *direct reciprocity*. When *a* downloads from *b*, he will allow *b* to download files from him in return. But if *b* refuses to let *a* download, *a* will not upload to *b*. Although this idea sounds promising and fair in theory, in practice there are some problems with it. The major problems here are the large scale of the networks, combined with high churning, and the relative anonymity of agents in the network. This makes most P2P transactions one-shot interactions between strangers that will never meet again [6]. An additional problem is what is called *asymmetry of interest*: when user *a* downloads a file from user *b*, even in the unlikely case the two do meet again for a file exchange, there is no guarantee user *a* offers the file user *b* is looking for. The probability of repeated interaction is even smaller than that of a second encounter.

To overcome the problems of the direct reciprocity approach, *indirect reciprocity* was proposed as an alternative. To illustrate this concept with an example, suppose user *a* downloads a file from user *b*. User *a* now knows that user *b* participates in the network. When user *b* requests a file from user *c*, *c* can ask the network including user *a* for the reputation of *b*, and when it finds that *b* has a good reputation, it will offer the requested file to *b*. Thus, for indirect reciprocity to work, some *reputation system* is needed that keeps track of the reputation of all participating peers. However, a reputation system is not trivial to implement. In the Kazaa network, for example, peers build up a reputation score by uploading, and highly reputed peers receive preferential treatment in their downloads [6], but the client itself is responsible for broadcasting its own reputation. Not surprisingly, this was soon exploited by the Kazaa Light client which by default broadcasts a very high reputation value. More secure distributed mechanisms provide theoretically valid frameworks, but are often not feasible in practice and as of yet, none of these mechanisms have successfully been deployed [17, 28]. We investigate reputation systems below.

Reputation Systems

Because of their usefulness in attaining indirect reciprocity, there has been significant research on reputation systems over the years. An overview is given by Tang [47], who distinguishes between global and local reputation systems. In a global system, every agent *i* has the same reputation of some agent *j*, while in a local system, *i*'s view of *j*'s reputation is subjective. Both have different computational efficiency and incentive compatibility characteristics, and neither strictly dominates the other. The problem with reputation systems is that they are easily manipulated, and very difficult to implement distributedly.

Centralized reputation systems are successfully implemented in the form of *private trackers* [28]. Here, a user needs an account and shares its upload and download statistics with the tracker. It can only download if its upload/download-ratio exceeds some threshold. These trackers suffer from the problems that all centralized components in a distributed network have; additionally, users may be reluctant to share their statistics with some third party, and there is the possibility of inflation of reputation [22].

Both Bachrach et al. [7] and Piatek et al. [36] propose distributed local reputation systems, specifically designed for application in P2P systems. With both approaches, every agent shares its experience in dealing with other agents with a small subset of the other agents in the system. Even though the ideas are interesting, neither of the systems has been deployed in practice.

Meulpolder et al. [28] propose BarterCast, a distributed local reputation system that is based on a similar idea, but that has been deployed in practice in the BitTorrent-based Tribler file-sharing network. We study BarterCast in detail in this thesis, when we investigate whether a user can improve on its performance by enabling or disabling BarterCast in order to answer our research question in Section 1.1.

Preventing Free-Riding with the Use of Currency

An alternative approach to reputation systems for attaining indirect reciprocity is the use of *currency*. A user gets some currency for uploading a file, that he can spend in order to download another file from a third user. Unfortunately, currency approaches face a number of practical issues as well. The currency needs to be “signed” to distinguish valid from counterfeit currency, which brings the need for some public key infrastructure [41]. Furthermore, the question is where the currency comes from: is it real money, that users can bring to the network? How is the currency linked to the valuation of a file, and to a network connection being used? Finally, it might actually destroy the incentive it strives to bring to the network: a peer that shares popular content can get so rich that it has no incentive to share its content anymore [41]. The seminal work in the use of currency is by Golle et al. [21]. Because of its shortcomings, we do not consider the use of currency in this thesis.

2.1.2 The BitTorrent File-Sharing Network

Our main focus in this thesis is on the BitTorrent network as it is the most popular file-sharing network today [37]. BitTorrent uses direct reciprocity as an incentive mechanism, but adds a twist to increase the probability that two peers will meet again. It does this with a mechanism called *bartering*, which we describe and analyze in great detail in this thesis to answer our research questions. For now, we suffice with a short intuitive explanation.

The file to be downloaded is partitioned into small chunks. A downloading peer p is matched with a small set of peers, that are also downloading or uploading pieces of the same file. From this set, p periodically selects a few peers that upload to it at the highest rate, and in return it uploads its own pieces to those selected peers. With this design, peers in BitTorrent engage in multiple interactions with a small number of peers for the duration of a file download period. For larger files, the number of repeated interactions is large enough to allow cooperation to take hold through direct reciprocity [6].

Free-Riding in BitTorrent

Because bartering forces repeated interactions, there is no need to keep long-term state information in the form of either reputation or currency, which simplifies the design and improves BitTorrent's robustness against attacks. Empirical studies found much lower levels of free-riding in BitTorrent communities [6], but theoretical analysis (e.g., Shneidman et al. [42], Sirivianos et al. [46], Piatek et al. [35], and Levin et al. [26]) has demonstrated that it can still be manipulated by selfish peers in their favor, improving download times or reducing uploaded volume.

This manipulation is not possible with any common BitTorrent client. Instead, a user that wants to perform these types of manipulation needs to download and install a modified client that sophistically subverts the bartering protocol. Empirical studies find that the use of such clients is not widespread [28]. Whether this means that users are not aware of these clients, or that they are – for one reason or another – not interested or not able to use them, is an open question, to our knowledge. However, this does not mean that users are not interested in minimizing their download time or reducing the amount of data they upload, and are willing to take strategic actions in order to achieve this. To give an example, the community support forum [1] for the Vuze client[3] is full of questions from users asking which settings to specify for maximum performance. We refer to such users as lazy free-riders (as opposed to die-hard free-riders that are willing to install manipulating clients). One of the two central questions of this thesis is whether BitTorrent provides these lazy free-riders with an incentive to share (see Section 1.1).

Adding Social Components to BitTorrent

With Pouwelse et al. [37], a new direction in offering incentives is taken. They introduce Tribler, a file-sharing system that is based on BitTorrent. Tribler strives to bring incentives to the system by adding social components to its network. Where users are anonymous in BitTorrent, in Tribler they have an identity. This allows them to import friends from other social networks they participate in, or make friends based on their tastes. They form a social network, with the idea that "kinship fosters cooperation". This is due to the higher probability of repeated interactions between users in a social circle, and the possibility that a user may gain social status in real life by actively cooperating in Tribler. We describe Tribler in detail in this thesis.

[3]Vuze is a highly configurable client for the BitTorrent network, available at `http://www.vuze.com/`.

2.2 Strategic Behavior in BitTorrent

Research on BitTorrent is not limited to free-riding and manipulation. Other topics include overlay topology formation, peer discovery, and content search. As these topics are not relevant to our research questions, we do not cover them in this thesis. There is, however, one additional topic that only recently receives attention [10]: the data distribution in BitTorrent, which governs how the file pieces are transmitted and distributed among peers. As peers are connected to the Internet with different upload- and download capacities, it will intuitively be clear that strategically selecting the right peer to download from directly improves performance.

This topic is covered by Chan et al. [10], Kumar and Ross [24], Ma et al. [27], and Qio and Srikant [39]. All try to determine an optimal piece distribution schedule, which is sometimes (e.g., with Ma et al.) linked to offering incentives to contribute to the network. All make problematic assumptions that limit the applicability in real P2P networks, such as relying on central components, the presence of a reputation system, or non-scalable computation times to determine the optimal schedule. In a file-sharing network with millions of concurrent users exchanging many files simultaneously, determining an optimal schedule on the pieces-level is a daunting task.

More recent research takes a different path. Instead of determining an optimal schedule from scratch, the data distribution in real BitTorrent swarms is investigated, and ideas for improvement are drawn from the results. Examples of this line of work are Bharambe et al. [8], Fan et al. [14], Legout et al. [25], and Meulpolder et al. [29]. All group the peers in the network in classes (e.g., slow, medium, and fast peers), and show that most data is exchanged within these classes, rather than between classes.

Our work in this thesis also follows this more recent path. We develop a model that allows for different peer classes based on some metric (such as upload capacity), and using that model we determine for any peer which peer classes it is likely to be bartering with. This allows us to predict the download completion time for a peer, and how strategically selecting its settings affects a peer's completion time. Additionally, we are able to explain, theoretically, observed phenomena from the articles mentioned above.

2.3 Game Theory and Mechanism Design

In the previous section, we briefly mentioned that in this thesis, we develop a model of the BitTorrent file-sharing network in order to answer the research questions stated in Chapter 1. We model a file-sharing network as a multi-agent system, where the agents are the actors in the system. What exactly constitutes an agent varies: sometimes this is limited to the client that communicates with the network, sometimes it involves the user as well. We model the agents to be autonomous (i.e., they make their own decisions), rational (i.e., these decisions can be motivated) and self-interested (i.e., they want to improve their own situation, and if necessary at the expense of other agents).

Our model heavily relies on game theory, which originally stems from economics where it is used to model agent behavior in markets. There, agents are usually people or companies that interact with each other. The market is modelled as a game, where each actor, being self-interested, tries to maximize its own profit. As multi-agent systems are very similar to markets, Nisan and Ronen [33] applied game theory in multi-agent settings. Specifically, they were interested in mechanism design, a subfield of game theory which does not directly model agent behavior, but instead asks how to design a game such that autonomous, rational and self-interested agents behave according to the system designer's specification. In general, this is done by offering incentives, as explained in Section 2.1.1. A thorough overview of mechanism design is given by Nisan [32].

Applied to BitTorrent, mechanism design would specify some mechanism that the agents interact with, and that motivates the agents not to free-ride. The problem with traditional mechanism design is that the resulting mechanism is centralized, and therefore not a good fit for a distributed network as BitTorrent. Feigenbaum et al. [15] investigate the possibilities of designing distributed mechanisms, but find that this is not trivial and no off-the-shelf solution can be given. One of the problems is that the mechanism needs to be carried out by the agents interacting with that very mechanism, allowing the agents to carry out the mechanism untruthfully in an attempt to improve on their current situation. Another problem is that, in general, agents need to communicate while executing the mechanism. First of all, this allows them to send false messages, and second, the communication overhead may be so large that it becomes intractable [16]. Distributing mechanisms is further investigated in three articles by Parkes and Shneidman [34, 43, 44].

In our model, we consider BitTorrent as a distributed mechanism. This is possible because in BitTorrent, the only mechanism is the peer selection mechanism we briefly introduced in Section 2.2 and investigate in depth in this thesis. This mechanism is carried out with minimal communication between the nodes – although it should be noted that Levin et al. [26] show how this communication can be exploited by a strategic client.

Even though mechanism design is by now an important technique in designing multi-agent systems, in this thesis we only borrow some ideas from it. Our main focus is on modelling agent behavior in BitTorrent networks, and for that, plain game theory suffices. Using game theory to that end is common practice by now; in fact, most of the literature presented in this chapter so far borrow from game theory.

Chapter 3

Model

Our main result in this chapter is a model of a generic peer-to-peer file-sharing network and its users. In the remainder of this thesis, we apply this model to BitTorrent and extend it with the BarterCast reputation system to answer our research question from Chapter 1.

We start with a description of the BitTorrent file-sharing protocol in Section 3.1, which is the basis for the BitTorrent file-sharing network. In Section 3.2, we use game theory to develop a model of a generic peer-to-peer network, and we show how to apply this model to BitTorrent. Section 3.3 describes how we model users in BitTorrent and BarterCast, and elaborates on the distinction between lazy and die-hard free-riders.

3.1 The BitTorrent File-Sharing Network

Our focus in this thesis is on the BitTorrent file-sharing network, and the Tribler client in particular. In this section, we describe both the BitTorrent protocol (Section 3.1.1), and the Tribler client (in Section 3.1.2).

3.1.1 The BitTorrent File-Sharing Protocol

The BitTorrent file-sharing network is one of the few P2P file-sharing networks that over the past five years has attracted and served a very large user community [38]. BitTorrent in itself is only a file-sharing protocol, which is implemented by many different clients. The protocol defines the entities in the network, and the messages exchanged between clients. In this section, we describe version 11031 [12], the current version of this protocol.

The BitTorrent network consists of the following entities:

- The *torrent*: a meta-info file, which describes the file that is to be exchanged,
- A *tracker server*, which keeps track of the peers exchanging the file described in the torrent,
- The *user* that originally shared the file,
- *leechers*: the users downloading the file ,
- *seeders*: the users that have completed the download and stay online, sharing the file to leechers.

In the following, we describe how these entities are used in the BitTorrent network.

A user that wants to share a file f creates a meta-info file for f, often referred to as a *torrent file* or *torrent*, after its extension `.torrent`. Most BitTorrent clients provide the option to create a torrent. A torrent can be created both for a single file, as well as for multiple files.

When creating the torrent, the file is logically split into fixed size *pieces* of the same length, typically 256 KB. The last piece may be truncated. Each piece is given an index. Then, for each piece, the SHA-1 hash[1] is calculated. The torrent lists the hashes ordered by piece index.

In addition to this hash list, the torrent provides the address of the *tracker* server. This tracker keeps track of all peers currently participating in the download, and collects statistics. It is not involved in the actual distribution of the file content. The tracker can be a *public tracker*, which is a tracker server available for public use, or a *private tracker*, for which a user account is needed, managed by a central authority. Many public trackers are available, one of the better known is `https://thepiratebay.org/`.

The torrent file is then made available to the public, often by placing it on a web site dedicated to hosting torrents. Most public trackers also function as torrent providers.

A peer p that wants to download f needs to obtain the torrent file. With every new download, p randomly creates an id. Then p connects to the tracker t, and sends t its id and address so t can track p. In response, t provides p with a list of peers currently exchanging f. This list of peers is typically a subset of all peers the tracker knows, i.e. a peer is not aware of all peers currently known to the tracker. We refer to the list of peers as the *remote peers*, and to the peer receiving the list as the *local peer*. Because peers join and leave the network continuously, the remote peer list needs to be kept up to date. A peer can update the list by sending a *re-request* to the tracker.

After it receives the remote peer list, a local peer connects to peers on this list. A connection starts with a handshake in which peers exchange their ids. After that, the peers exchange a bit vector, in which each bit represents whether the corresponding piece (ordered by index) is present or not.

A peer a is *interested* in peer b when b has pieces that a does not have; otherwise it is *uninterested*. A peer a is *choked* by peer b when b decides not to send any data to a. If b is willing to send data to a, a is *unchoked* by b. This happens when a has pieces that b does not have. All connections start out choked and uninterested. A local peer notifies a remote peer when the remote peer gets choked or unchoked, or when the local peer becomes interested or uninterested in it.

A local peer that is unchoked by a remote peer a can request a for a specific piece. In return, a will send the piece to the local peer. When the local peer completes the download of a piece, it notifies its remote peers that it now has this specific piece.

The local peer is free to decide which pieces it requests, and in theory each implementation of the protocol could employ a different piece selection strategy. Examples of a piece selection strategy are downloading pieces in order, or randomly selecting which piece to download. In practice, many clients, including the official BitTorrent client[2] and Tribler, adopt the *Rarest Piece First* (RPF) piece selection strategy [25], and this is the piece selection strategy we consider in this thesis. With this strategy, each peer maintains a list of the pieces with the least number of copies among its remote peers, and pieces from this list are requested first. This minimizes the probability that pieces become unavailable when a single peer goes offline.

A local peer is also free to decide which peers it chokes or unchokes. Choking has two benefits. First, TCP congestion control behaves poorly when sending over many connections at once, so a local peer achieves better upload performance when uploading to a limited number of peers simultaneously. Second, the protocol designers hope that, from the choking algorithm, a tit-for-tat-ish behavior will emerge where agents upload file pieces proportional to what they download. The idea

[1]SHA-1 is the first version of the Secure Hash Algorithm, a cryptographic message digest algorithm. See `http://www.w3.org/PICS/DSig/SHA1_1_0.html` for more details.

[2]The official BitTorrent client is developed by BitTorrent, Inc., and freely available from `http://www.bittorrent.com/`.

behind the latter is that a local peer unchokes remote peers from which it can download at high rate, and chokes the slower uploading peers, thus motivating the remote peers to upload at high rate.

The specification provides five criteria a good choking algorithm should meet:

1. The number of simultaneous uploads should be capped for good TCP performance,
2. Quickly choking and unchoking (*fibrillation*) should be avoided,
3. It should reciprocate to peers who let it download,
4. It should try out unused connections once in a while to find out if they might be better than the currently used ones. This is known as *optimistic unchoking*,
5. the algorithm should work well both in a network consisting entirely of clients implementing the algorithm, and in a network consisting mostly of clients implementing the algorithm.

As with the piece selection strategy, every protocol implementation can create an implementation of a choking algorithm. In this thesis, we confine ourselves to the currently deployed choking algorithm in the official BitTorrent client and Tribler. This algorithm avoids fibrillation by only changing who is choked every ten seconds. This ten second period is a *round*, and we say that a new round starts whenever the choking algorithm is executed.

Two choking algorithms are in use. One is employed when the user is a leecher, and the other when the user is a seeder. When leeching, reciprocation and number of uploads capping is implemented by unchoking a few peers from which the local peer has the best download rates, and that are interested. Peers with better upload rates are unchoked and when they become interested the worst uploaders get choked. This is known as a *regular unchoke*. Optimistic unchoking is implemented by unchoking a randomly selected peer every three rounds, regardless of it's upload rate, in the hope of finding better peers.

When seeding, the agent bases its decision on who to unchoke on its upload rate rather than its download rate, and prefers peers to which it can upload fastest.

Although BitTorrent is famous for its tit-for-tat-ish *bartering* (the exchange of file pieces between downloading peers), this behavior is not specified in the protocol. The protocol only specifies which messages can be send between clients. The bartering emerges from the limited number of upload slots an agent has, and the choking algorithm. As a result, an agent is given an incentive to upload while it is downloading. When the other agents in the network are bartering, it is in any agent's best interest to participate in bartering, as an agent that is free-riding only receives pieces by being optimistically unchoked.

3.1.2 Tribler

The BitTorrent file-sharing protocol we described in the previous section introduces bartering of file pieces as a technical incentive to share files: high transfer rates can only be achieved by users that contribute pieces of the downloaded file. Where a large portion of the research in file-sharing networks (and in P2P networks in general) focuses on technical incentives, Pouwelse et al. [37] introduce Tribler, a BitTorrent client based on a social P2P file-sharing paradigm that "exploits social phenomena by maintaining social networks and using these in content discovery, content recommendation, and downloading." The authors argue that the problem of free-riding can be alleviated when users are considered social partners that tend to cooperate with the social group they belong to.

At the basis, Tribler is a client for the BitTorrent network and thus implements the BitTorrent specification. The social components are added on top of the BitTorrent layer. Using Tribler, users can find users with similar interests and add these as contacts. This way, a social network is formed. In the future, it will become possible to expand this network by importing contacts from other social networks (e.g. MSN or GMail). This requires each peer to have a permanent identifier (PermID), because otherwise it cannot be identified by its contacts when it rejoins the network after a disconnect. This permanent PermID is used as the randomly generated identifier suggested by the BitTorrent protocol.

The social network provides Tribler users with new functionality unavailable from traditional BitTorrent clients. This includes content discovery (files can be downloaded from contacts without the need for torrents), suggesting users with similar interests, give recommendations for files based on a user's taste, and cooperative downloading (peers from a social network assist each other in downloading, resulting in higher transfer rates). In this thesis, our focus is on a new feature introduced in Tribler in 2009: the BarterCast reputation mechanism [28]. This provides both a technical and a social incentive to users to be online sharing files even when not downloading, thus improving the availability of files in the network. We describe BarterCast in detail in Section 4.3.

3.2 Model of a P2P File-Sharing Network

Based on our discussion of BitTorrent and Tribler in the previous section, in this section we define a formal model of a P2P file-sharing network based on game theory. This model models peer-to-peer file-sharing networks in general, and BitTorrent in particular. We use the model derived in this section in the remainder of this thesis to gain insight into a user's download performance, and what options are available to users to improve this performance. We extend this model in Section 4.3 to determine how the BarterCast reputation mechanism affects the download performance and options for improvement.

In Section 3.2.1, we state the objectives and requirements of a P2P file-sharing network, and define the problem description for such networks. Section 3.2.2 defines a static model, followed by an example in Section 3.2.3. Based on that static model, Section 3.2.4 introduces a dynamic model that takes into account that agents may join or leave the network, change files they request or share, etc. This is followed by a discussion of how agents may benefit from using the network (Section 3.2.6). Finally, we list what aspects of file-sharing networks we omitted from our model.

3.2.1 Problem Description

Users join a file-sharing network to download files, and it is safe to assume that they want to download these files as quickly as possible. The user's benefit from using the network increases with higher transfer rates. We refer to this benefit as a *utility*, which we formally define in Section 3.2.2. For now, we just assume that utility is linked to its download rate, and that a user wants to maximize its utility. The actual utility a user receives is different for each user. Utility is negatively influenced by the costs of using the network, which will generally consist of files a user needs to upload. A technical motivation for this is that uploading files affects a user's download capacity, while a more emotional motivation is that, where possible, people prefer to receive things without doing something in return. However, as one agent's download is another agent's upload, uploading files is required for the network to function. A file-sharing network that offers requested content for download at high speeds is attractive to users, as long as the benefits of using the network outweigh the costs.

Since the end of the 1990s, many P2P file-sharing networks have been introduced. Pouwelse et al. [38] find that of these networks, BitTorrent is one of the few that, over the past years, has consistently managed to attract millions of users. In the article, they argue for four requirements a P2P file-sharing system must have to be attractive to users, and show that BitTorrent has all these properties to some extent. The requirements are:

1. High availability: the system must be available to users most of the time. Additionally, files that are available in the network should constantly be available,

2. File integrity: the files must be as advertised, and not fake, manipulated, or unplayable,

3. The network must be able to deal with flash crowds. A flash crowd is the phenomenon that occurs when new popular content first enters the network and is immediately requested by a large number of users. This generates a burst of network activity and traffic which may negatively affect the system's performance,

4. The system must offer users a relatively high download speed.

In this thesis, we look at file availability indirectly when we investigate the BarterCast reputation system, and investigate the download speed that is offered to the users. In BitTorrent, this is addressed by the assignment policy: BitTorrent's decision on which agent downloads which file from which agents, and at which rate. We described BitTorrent in detail in Section 3.1.1, and describe there how clients make a selection of which clients to upload to or to download from. We refer to the result of this selection as an assignment, and elaborate on it in the remainder of this thesis. In this section, we confine ourselves to the idea behind BitTorrent's assignment policy, as given by Cohen [11]: "The strategy for allocating upload which seems most likely to make peers happy with their download rates is to make each peer's download rate be proportional to their upload rate."

The idea that an agent's download should be proportional to its upload has been suggested and implemented before, e.g. in Kazaa, and is fair in that it enables users to benefit from the network, but requires the users to participate proportionally with the network in return.

Many users connect to BitTorrent simultaneously, and each of these users has its own utility function. In general, it will not be possible to maximize every user's utility. An increase of one user's utility may come at the expense of that of another user. However, following the last requirement, the network must be attractive to all users. We refer to this attractiveness as *social welfare*, and it is our goal, as system designers, to maximize this social welfare in order to make the system attractive to users. At this point, we do not restrict ourselves to one specific definition of utility or social welfare.

If we assume, as we do in the beginning of this section, that utility is linked to a user's transfer rate, one example of a social welfare function would be to maximize the sum of the agents' utilities, i.e. maximizing the sum of all transfer speeds. This, however, is not effective when transfer rates greatly vary and social welfare is maximized when all users have very low transfer speeds except for one agent that has an exceptionally high transfer speed. In such a network, a more suitable social welfare function would be a function that minimizes the deviation of the agents' transfer speeds from the average transfer speed.

When we want the user's download rate to be proportional to its upload rate, we affect his user utility as we force him to upload. However, the proportionality constraint ensures that, depending on a user's utility function, a positive utility can be obtained from using the network. Additionally, many users will understand that it is fair to contribute to a network that offers benefits in return, and will rather contribute to the network then leave it if they are given the choice.

The assignment policy determines for every user the utility it receives. As a result, users may attempt to free-ride by manipulating the assignment policy. This would result in an increased utility for these users at the expense of the other agents in the system. That is an unwanted situation, so we require the assignment policy to be non-manipulable by the agents. In the remainder of this section, we derive a a model for a file-sharing network. We use this model to investigate BitTorrent's assignment policy.

3.2.2 Static Model

In this section, we derive a model for a file-sharing network. This model is not specific for one particular file-sharing network, but rather applies to file-sharing networks in general. Section 3.2.5 demonstrates how this model can be applied to the BitTorrent file-sharing network. We extend the model derived in this Section in Section 3.2.4, and the two models are used in the remainder of this thesis as tools to analyze file-sharing networks.

Preliminaries

We distinguish the following entities in our model:

1. The *agents* that wish to exchange files using the file-sharing network,

2. The *Internet*, which is used as an underlay network. All communication between the agents is physically transferred over the Internet,

3. The *network connections* over which agents are connected to the Internet,

4. The *file-sharing network* that is formed by the agents using this file-sharing network's protocol to communicate, using the Internet as an underlay network,

5. The *files* that are available to the agents to exchange in the file-sharing network,

6. The *mechanism* that determines for every agent which files it can download, and from which agents.

The model we derive in this section is static. By that, we mean two things: first, that all entities are fixed. No elements are added or removed from any set, network connections do not change, etc. Second, this means that when actions need to be performed, they are performed before the game starts and will not be changed during the game. An example of an action is an agent sharing a file, or specifying which portion of its Internet connection is available for the file-sharing network. In Section 3.2.4, we introduce dynamic aspects into our model and remove the assumption we make here that everything is fixed.

A file sharing network is about exchanging files. Agents bring to the network a set of files they wish to share, and maintain a list of files they wish to download. We assume that there is some mechanism M that, based on which files are available, which files are requested, and the agents' network connections, decides for any agent at what speed it downloads a file piece from which agent, and when this download starts. Intuitively, one can think of such a mechanism as some broker that makes this decision. How exactly the mechanism is implemented depends on the file-sharing network. In BitTorrent, as we described in Section 3.1.1, this decision is implemented by the BitTorrent clients. See Section 3.2.5 for an application of our model to BitTorrent.

Network

Denote by N a set of n agents. Every agent $i \in N$ is connected to the Internet with some network connection, and is capable of participating in the file-sharing network, i.e. it can send the file-sharing network's protocol messages over the network connection to the Internet. Through the file-sharing network, every agent i can reach every other agent it wants.

The connection to the Internet provides agents with both an upload connection and a download connection to the file-sharing network. Both types of connections have a data transfer rate, or capacity, which is expressed in bytes per second. This capacity is limited by the capacity of the physical connection to the Internet. The capacities for the upload and download connections of an agent to the file-sharing network are given by c_u and c_d, respectively:

$$c_u : N \to \mathbb{R}^+ \tag{3.1}$$

$$c_d : N \to \mathbb{R}^+ \tag{3.2}$$

An agent has a number of *upload slots*, which determines the maximum number of peers the agent will upload to simultaneously. See Section 3.1.1 for more details. The upload slots are given by the function k:

$$k : N \to \mathbb{N}^+ \tag{3.3}$$

Files

Agents in the network request files and share files for download by other agents. Denote by $F_{S,i}$ the set of files shared by agent i, and by $F_{R,i}$ the set of files requested by i.[3] The set of shared files in the network is then denoted by $F_S = \bigcup_{i \in N} F_{S,i}$. Similarly, $F_R = \bigcup_{i \in N} F_{R,i}$ is the set of requested files. Finally, denote by F the collection of all the files considered in the network: $F = F_S \bigcup F_R$. Note that some requested files may not be shared or even available in the network.

Files can be transferred in small pieces, as is the case in BitTorrent (see Section 3.1.1). A file $f \in F$ is a set of file pieces $p \in P$. How exactly files are divided into pieces depends on the network in question.

Download Assignment

In a file-sharing network, many agents are connected. Every agent may share multiple files, or request multiple files. Multiple agents may request the same file f, and multiple agents may offer f. The broker M we introduced in Section 3.2.2 considers all agents, the files they request and share, and their network connections, and based on this information decides which file pieces are exchanged by which agents, the rate of the transfer, and the time at which this transfer starts. We refer to such a decision as a *download assignment*, which is a collection of *scheduled piece exchanges*. We add one restriction to a download assignment: an agent is assigned only one download for every file piece, i.e., it is not possible for an agent to download one piece from multiple peers, or from one peer at multiple times and/or at multiple rates.

We consider time as an infinite series of discrete time steps: $T = \{0, 1, 2, \ldots\}$, and then formally define a download assignment as follows:

Definition 3.1. *A* download assignment *is a set of 5-tuples* (n_d, p, n_u, r, t)*, where:*

- $n_d \in N$ *is the downloading agent,*
- $p \in P$ *the downloaded file piece,*
- $n_u \in N$ *the uploading agent;* $n_u \neq n_d$*,*
- $r \in \mathbb{R}^+$ *is the rate in bytes/second at which the transfer takes place,*[4]

[3] In the remainder of this chapter, we use the subscript i to denote a function or variable for an agent i. So, $F_{S,i}$ is the set of files shared by agent i, and $c_{u,i} = c_u(i)$, etc.

[4] In this thesis, we define $\mathbb{R}^+$ as $\{x \in \mathbb{R} \mid x > 0\}$, and $\mathbb{R}^+_0$ as $\{x \in \mathbb{R} \mid x \geq 0\}$.

- *$t \in T$ is the time at which the download starts.*

In any download assignment, the combination of (n_d, p) is unique.

Denote by Π the set of all possible combinations of piece exchanges: $\Pi = 2^{N \times P \times N \times \mathbb{R}^+ \times T}$, the power set of all piece exchanges. Every element of Π is a possible download assignment. Based on its information on the network and connected agents, M chooses one assignment out of Π that best meets its objective (see Section 3.2.1). This assignment specifies all piece exchanges in the network, and we refer to that set as the *outcome*.

Agent Types

For every agent i, there will be possible outcomes $\pi \in \Pi$ that are more beneficial than others. In one outcome, an agent can download all requested files, for example, while in another this is not possible, and yet another outcome may dictate the agent to upload more files than other outcomes. Every agent i wants M to select an outcome that maximizes i's utility.

The mechanism M does not know which files an agent i has stored on its computer, and which of these files are actually shared by i. Only i itself is informed of this information. We say that agents have *private information*. Another example of private information is the upload and download capacities an agent has available. An agent performs actions based on this private information: request and share files, and make upload and download capacity available to the network. The outcome decided on by M depends on these actions: if for example an agent decides not to share a file f, it may not be available in the network and cannot be assigned to an agent requesting f. This means that an agent can influence the outcome by choosing its actions strategically.

An agent's private information is modelled by its *type*: for each agent i, there is a set of types Θ_i. One of these, $\theta_i \in \Theta_i$, is i's type and models i's private information. A user's type models many things, including:

1. The type of content the user is interested in. For example: action movies, disco music, etc,
2. The cost of uploading,
3. The effects of altruism: will sharing a file increase a user's happiness?
4. A user's willingness to free-ride, and to what extend a user will free-ride.

Actions

Based on its type, an agent performs *actions* in order to reach its goal: maximizing its utility. The actions available to an agent depend on the file-sharing network. Different agents may have different actions at their disposal, which is modeled by a agent's action space:

$$x : N \to 2^X \tag{3.4}$$

Here, X is the set of all actions available in the network's protocol. Every agent i has an action space $X_i \subseteq X$.

We enumerate the actions available to users in the BitTorrent network in Section 3.2.5. In general, the actions include sharing and requesting files, and specifying the upload and download capacity somewhere between zero and the physical capacity of the link.

By giving every agent a set of actions, we imply that different agents may have different actions at their disposal. We elaborate on this in Section 3.3, where we distinguish different classes of agents, and define for every agent class what actions are available to the agents in that class.

Outcome

M's goal is to consider all agents' actions and from those, determine an outcome that satisfies some objective. An example of such an objective is given in Section 3.2.1: to maximize the average transfer rate, keeping for each agent download in proportion to upload. The mechanism thus can be thought of as some protocol that specifies possible actions and implements an outcome based on these actions. The outcome is determined using an outcome function:

$$g : X_1 \times \ldots \times X_n \rightarrow \Pi \tag{3.5}$$

It is important to realize that in our static model, the outcome is static as well. We assume all agents choose their actions (i.e., share files, request files, set upload and download capacity, etc.), and that those actions do not change. The outcome is then fixed based upon that information. This means that a downloading agent will seed its downloaded files according to the seeding strategy it selected. As the set N is static, in this model agents do not leave the network after their download completes, but they may opt to not seed the downloaded file, which has the same effect.

Utility

Section 3.2.1 introduced an agent's utility as some measure of the benefit it receives from using the network. We are now ready to give a formal definition of utility. From the above, it will be obvious that this utility depends on an agent's type, which captures its preferences, and the outcome, which specifies which files are exchanged at which rates. For every agent $i \in N$, we define its utility function as:

$$u_i : \Theta_i \times \Pi \rightarrow \mathbb{R} \tag{3.6}$$

Some realistic utility functions in Tribler are given in Section 3.2.6. As a small illustration of a utility function, think of a function that returns the sum of all download rates for an agent: the faster the outcome allows the agent to download, the higher its utility from that outcome will be.

We assume that an agent receives its utility when download completes. A seeding agent receives utility when it stops seeding.

Strategies

An agent's utility indirectly depends on all agents' actions, because the joint set of actions determines the outcome. Any agent i only knows its own private type θ_i, and chooses its own action $x_i \in X_i$. Although the other agents' types are not known to i, their actions (which are based on these unknown types) do affect i's utility. As i tries to maximize its utility, it needs to select an action from its action space that will achieve just that. The decision on which action to choose is made by an agent's *strategy function*, which chooses an action based on the agent's type:

$$s_i : \Theta_i \rightarrow X_i \tag{3.7}$$

The strategy function s_i in turn is an element of the agent's strategy function space S_i, which is specified in the agent's type.

Static Game

An agent's type θ_i captures i's private information. In a file-sharing network, although θ_i is private information, other agents may have some belief about it. Bandwidth, for example, is not uniformly distributed over Internet users, because Internet providers offer a limited set of subscriptions, some of which are more popular than others. An agent can therefore make an informed guess about the capacities of the network connections of the other agents. Another example is that some files are very popular and, depending on the file's age, probably requested by many agents or available from many agents. This may influence an agent's decision on when to share or request a file, if at all. This information is publicly available, and we assume that all agents share the same information.[5] Then, the agent's types are drawn from a probability distribution over the types. We refer to this distribution as a *common prior* over the types, and denote this by P:

$$P : \Theta \rightarrow [0,1], \tag{3.8}$$

with $\Theta = \Theta_1 \times \ldots \times \Theta_n$.

With the above, we define file-sharing as a *Bayesian game*, which Leyton-Browne and Shoham [9] define as follows:

Definition 3.2. *A* Bayesian game *is a tuple (N, X, S, Θ, P, u, g), where:*

- *N is a set of n agents,*
- *$X = X_1 \times \ldots \times X_n$ is the set of actions available to the agents,*
- *$S = S_1 \times \ldots \times S_N$ is the set of strategies available to the agents,*
- *$\Theta = \Theta_i \times \ldots \Theta_n$ is the set of type spaces available to the agents*
- *P is the common prior defined above,*
- *$u = (u_1, \ldots, u_n)$ is the vector of the agents' utility functions,*
- *g is the outcome function from Equation 3.5.*

For this game, we assume the following:

One-shot game: We assume file-sharing is a *one-shot game*. By that, we mean that every execution of the game stands on itself and no information or state is carried from one execution of the game to the other. Effectively, this corresponds to a game that is executed only once.

Game proceedings: At the beginning of the game, all agents simultaneously use their strategy function to choose their actions. Actions are fixed after being chosen. When all actions are chosen, M decides on an outcome, according to which the file transfer is executed. The game ends when all file transfers are completed as specified in the outcome. Because actions are fixed, and per the assumption of a static network in which no agents leave the network as we explained in Section 3.2.2, the game will always end.

Solution Concepts

Every agent i in the network has a strategy function, which chooses the action that i expects will maximize its utility. By doing so, i affects the other agents' utilities. Another user anticipates on that and adapts its actions, which in turn affects the other agents' utilities, etc. This way, the agents' actions are intricately interwoven.

We refer to the tuple $\{s_i, \ldots, s_n\}$, which contains one strategy for every agent, as a *strategy profile*. Every agent selects a strategy that provides it with maximum utility. If every agent selects

[5]This is a very limiting assumption, but we make it to be able to apply the main ideas from Bayesian games later in this section. Most (but not all) work in game theory makes this assumption [9].

such a strategy, and no agent can select a different strategy without lowering its utility, then we refer to that strategy profile as an *equilibrium*. It is our goal, as system designers, that every agent selects a strategy that has it specify full upload capacity and long seeding times. Therefore, we need to design an equilibrium in which every agent selects such a strategy.

Different types of equilibria exist, and in this thesis we confine ourselves to the three that are most used: the Nash equilibrium, the Bayesian-Nash equilibrium, and the Dominant Strategies-equilibrium [32]. The equilibria differ by the assumptions they require an agent to make of the other agents in the network.

The most influential solution concept in game theory is the Nash equilibrium [9]. Before we define this equilibrium, we first define what a *best response* is for an agent. We define $s_{-i} = \{s_1, \dots, s_{i-1}, s_{i+1}, \dots, s_n\}$ as the strategy profile s without i's strategy. We can then write $s = (s_i, s_{-i})$. If all agents other than i play s_{-i}, i needs to determine the strategy that provides it with maximum utility: his best response.

Definition 3.3. *(Best response) Agent i's* best response *to the strategy profile s_{-i} is a strategy $s_i^* \in S_i$ such that $u_i(s_i^*, s_{-i}) \geq u_i(s_i, s_{-i})$ for all strategies $s_i \in S_i$.*[6]

Note that the best response is not necessarily unique, as multiple strategies may result in the same utility.

In general, an agent i does not know which profile s_{-i} the other agents will play. However, if every strategy s_i in the strategy profile s is a best response to the other strategies s_{-i}, then no agent has an incentive to choose another action as that will always decrease its utility. Therefore, this is a stable strategy profile, which motivates the term equilibrium. Specifically, this is the Nash equilibrium, which Leyton-Brown and Shoham [9] formally define as follows:

Definition 3.4. *(Nash equilibrium) A strategy profile s is a* Nash equilibrium *if, for all agents i, s_i is a best response to s_{-i}.*

We distinguish between *strict* and *weak* Nash equilibria. In the former, i's utility obtained by playing s_i is unique, while in the latter it is not.

In the Bayesian game we defined above (Definition 3.2), the types are distributed over the agents according to the distribution P. As a result, an agent does not know what its utility will be when it plays s_i; instead, it expects some utility based on P. We denote this *expected utility* for agent i by $EU_i(s)$. Because the meaning of expected utility is intuitively clear, while the formal definition is convoluted, we do not formally define this here but instead refer the reader to Leyton-Brown and Shoham [9] for a formal definition. Next, we define the best response in a Bayesian game:

Definition 3.5. *(Best response in a Bayesian game) The set of agent i's* best responses *to a strategy profile s_{-i} is given by $BR_i(s_{-i}) = \arg\max_{s_i \in S_i} EU_i(s_i, s_{-i})$.*

This allows us to define the Nash-equilibrium in a Bayesian game, which is referred to as the Bayes-Nash equilibrium:

Definition 3.6. *(Bayes-Nash equilibrium) A strategy profile s is a* Bayes-Nash equilibrium *if, for all agents i, $s_i \in BR_i(s_{-i})$.*

Sometimes, an agent i has a strategy s_i that yields a greater utility than any of its other strategies, for any strategy profile of the remaining agents. Such a strategy is a *dominant strategy*. If this is the case, then i always plays s_i, regardless of P. A special form of Nash equilibrium is the Dominant

[6]Note that we are a little loose in notation here, as formally an agent's utility is a function of its type and the outcome. However, this is justified here as the strategy chooses the agent's actions based on its type, which in turn determine the outcome.

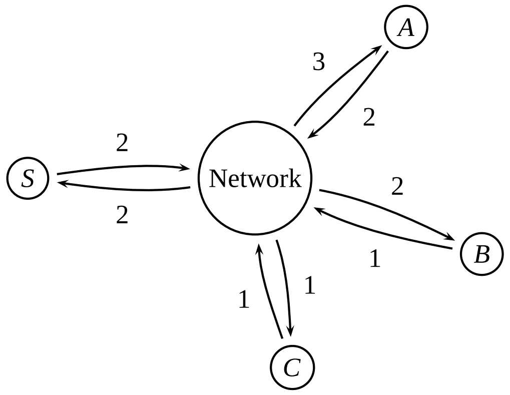


Figure 3.1: *Example of a file-sharing network with one seeder (S), and three leechers (A, B, and C). Arrows indicate connections to and from the network. Capacities are given next to the connections.*

Strategies equilibrium, in which every strategy is a dominant strategy. Based on Shoham and Leyton-Brown [45], we define this as follows:

Definition 3.7. *(Dominant Strategies equilibrium) A strategy profile s is a* Dominant Strategies equilibrium, *if, for all agents i, s_i is a dominant strategy.*

In Section 3.2.1, we motivated that we do not want the outcome to be manipulable. It will now be clear how an agent may try to manipulate the outcome to its own advantage: by sharing only few files it has available, or making only a fraction of its upload capacity available to the network, for example. This way, it avoids uploading while it may be able to download files nonetheless. Because the files that an agent can share, or its maximum upload capacity, are private information, it is impossible for the mechanism to verify an agent's actions and check whether it manipulates. However, we do know that the agents are rational and they try to maximize their utility. If it is possible to design the mechanism such that it implements g (Equation 3.5) in some equilibrium, then an agent maximizes its utility by playing the equilibrium strategy (assuming the other agents are rational and do the same). We thus need to make sure that this equilibrium strategy is the desired strategy from our perspective.

3.2.3 Static Model Example

In this section we provide a small example to illustrate the static model we just defined. In Figure 3.1 a file-sharing network is given, with one swarm consisting of four agents: a seeder S, and leechers A, B, and C. All agents have an upload connection to the network (outward arrow) and a download connection from the network (inward arrow). For every connection the capacity is listed. All capacities are in kB per second. Every agent makes its full connection capacity available to the network. For simplicity, we define every file piece to be 1 kB in size. Then the connection capacity corresponds to the number of pieces that can be sent over the connection in one second. Finally, every agent has two upload slots.

The file considered in this swarm is f, which consists of three pieces: $f = \{p_1, p_2, p_3\}$. Initially, the seeder has all three pieces available (i.e., the complete file, while the leechers have none. The leechers decide to seed the downloaded file after their download completes.

We assume that the leeching agents have the same type. Their utility functions are also the same, and the faster download completes, the higher utility is:

Time	p_1	p_2	p_3
0	$S \rightarrow A$	$S \rightarrow B$	
1	$A \rightarrow B$	$S \rightarrow C$ $B \rightarrow A$	$S \rightarrow A$
2			$S \rightarrow C$ $A \rightarrow B$
3	$S \rightarrow C$		

Table 3.1: *Assignment from g for the network in Figure 3.1. Each row shows at what time the corresponding piece is sent from the uploading to the downloading agent.*

Agent	Utility
S	0.6
A	0.5
B	0.33
C	0.25
Social welfare:	1.68

Table 3.2: *Utilities and social welfare for the outcome in Table 3.1.*

$$u_i(\theta_i, \pi) = \frac{1}{\Delta t}$$

where Δt is the number of seconds it takes i to complete its download in outcome π.

With the above utility function, the seeder would never receive any utility because it does not download anything. Vassilakis and Vassalos [48] distinguish two ways a seeding peer may receive utility, and we adopt one here: utility is proportional to the uploaded volume. The seeder's utility function is then defined as:

$$u_S(\theta_S, \pi) = \frac{\text{number of uploaded pieces}}{10}$$

A possible outcome decided on by g is given in Table 3.1, which could be an outcome in a real BitTorrent swarm. Because agents A and B have the largest download capacity available, S prefers to upload to those agents. It sends different pieces to each of them to ensure a good diversity of the pieces in the system. At $t = 1$, S optimistically unchokes C and sends it p_2, and it sends p_3 to its fastest downloader, A. Because A and B have different pieces available locally, they can start bartering and they exchange p_1 and p_2. A has now completed its download. It stays online as a second seeder, sending p_3 to B, which allows B to finish its download. S sends the final two pieces to C.

The utilities for the agents and social welfare arising from this download assignment are given in Table 3.2.

We find that, with the given utility functions and outcome, all agents obtain a positive utility. However, other outcomes are possible as well. Suppose the mechanism's objective is to minimize the deviation of the average download time, or, in other words, to have all agents finish their downloads roughly at the same time. An outcome satisfying that objective is given in Table 3.3 shows. Here, all agents finish their download after $t = 2$.

Obviously, with the alternative assignment all leeching agents have the same utility, as is shown in Table 3.4. The seeder's utility does not change with this new assignment, but social welfare is

Time	p_1	p_2	p_3
0	$S \rightarrow C$	$S \rightarrow B$	
1	$S \rightarrow B$ $C \rightarrow A$	$B \rightarrow A$	$S \rightarrow C$
2		$A \rightarrow C$	$S \rightarrow A$ $S \rightarrow B$

Table 3.3: *Alternative outcome allows all agents to finish simultaneously.*

Agent	Utility
S	0.6
A	0.33
B	0.33
C	0.33
Social welfare:	1.6

Table 3.4: *Utilities and social welfare for the alternative outcome from Table 3.3.*

slightly lower than with the previous assignment.

In this example, all agents made full connection capacity available to the network. Suppose that B would have refused to upload, so it would set its upload capacity to zero. This would have had an impact on both outcomes. In both outcomes, B sends p_2 to A at $t = 1$, which is impossible without upload capacity. As a result, A cannot finish its download after $t = 1$. Assuming the seeder would send the missing piece to A, utility for B would be the same in both outcomes, but A's utility would drop significantly: from 0.5 to 0.33 in the first outcome, and from 0.33 to 0.25 in the second. In the first assignment, B's refusal to upload would cause social welfare to drop from 1.68 to 1.61. In the latter, social welfare rises slightly from 1.6 to 1.61, although the mechanism's objectives are better met with the original outcome. This shows the effect one agent's manipulation can have on social welfare and the other agents' utilities.

3.2.4 Towards a Fully Dynamic Model

The model we defined in Section 3.2.2 describes a static game, in which the agents choose their actions at the beginning of the game and these actions are fixed for the remainder of the game. This is a major simplification of reality in P2P file-sharing networks, which are inherently dynamic: agents join and leave the network continuously, new files are added to the network, shared files are removed, etc. In this section, we add dynamic aspects to the static model. Although a fully dynamic model gives a good representation of reality, it may also needlessly complicate the model. We therefore choose to review the aspects that can be dynamic. This allows us to use the simpler static model with only the dynamic aspects that are of interest.

In the following sections, we first define the dynamic game and how it relates to the static game. Then, we investigate what aspects of the static model are dynamic in reality and how they can be modelled in a dynamic fashion. We do this in order of importance: we start with the files that are requested and shared, as agents will typically change these after each successful download. We then proceed to agents joining and leaving the network. After that, we consider agents that change their network capacity. Finally, we investigate the dynamic aspects of the utility function. Any aspects we omitted from this model are listed in Section 3.2.7.

The Dynamic Game

In Section 3.2.2, we defined the static model as a Bayesian game. This is not changed in this section where we make the model dynamic. As in the static model, the game consists of a set of agents, each of which have action and type spaces, a utility function, and a common prior over the agent types.

The other assumptions we made in Section 3.2.2 no longer hold. We replace those with the following assumptions.

Dynamic game: For the static model, we defined a static game in which agents specify their actions once at the beginning, after which the actions are fixed and the mechanism determines an outcome. In our dynamic model, we remove this limiting assumption and instead allow the agents to perform actions continuously.

Game proceedings: As we describe above, in the dynamic model, agents can perform new actions at any time. For example, agents go offline at some point and rejoin the network later, or when a new movie comes out, agents will request that. As a result, the game does not end. Agents can exchange files whenever they wish, and do not need to wait for all transfers to complete whenever a new file is shared or requested. Because the outcome depends on the actions the agents perform, a new outcome needs to be determined whenever an actions is performed.

Types and Strategies: The dynamic model runs for an extended period of time. Over that time, possibly many years, an agent's type and strategies can change. We choose not to include this in our model to not make it overly complex. Instead, we assume that an agent's type, and its strategy function, are fixed.

History and agent memory: We assume that the agents have a limited memory available. How exactly memory is limited depends on the particular file sharing network under consideration.

Dynamic Actions In the dynamic model, agents have all actions at their disposal as in the static model. Additionally, they can join or leave the network. The actions available to the agents in BitTorrent are enumerated in Section 3.2.5.

Download Assignment and Outcome

We keep our definitions of a download assignment and outcome (Definition 3.1). A download assignment may be scheduled in the future. This may happen, for example, because the uploading agent is not yet online at some time t, or its upload connection is currently filled to capacity. At any time t, an outcome can be calculated based on the information available at time t. This outcome includes download assignments that start immediately, as well as assignments scheduled in the future.

$$g : X_1 \times \ldots \times X_n \times T \rightarrow \Pi \tag{3.9}$$

In the dynamic model, agents can perform actions continuously. As every action influences the outcome, this means that an outcome becomes invalid whenever an action is performed. When that happens, a new outcome is determined, which is valid until the next action is performed.

Utility

As in the static model, in the dynamic model we assume that a leecher receives its utility when the download completes, because an incomplete download is useless and has no value to the user. This

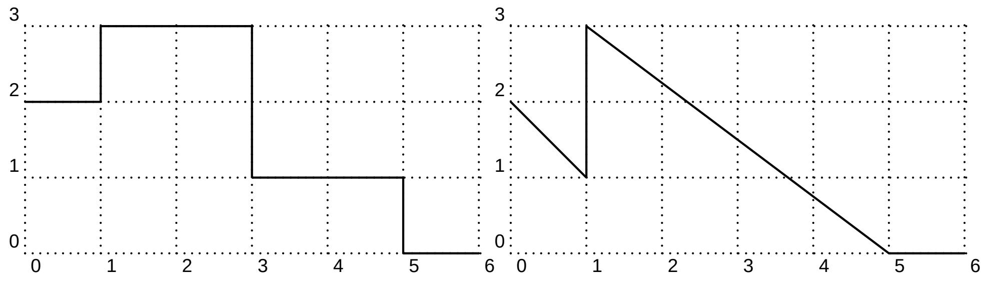


Figure 3.2: *Utility changes over time. Downloads complete at $t = 0$ and $t = 1$.*

may be a simplification of reality: one can imagine that a user receives a small utility when it notices a download for a movie has completed, but receives full utility when the movie is actually being watched. Because that is outside the scope of this model, we ignore that possibility.

It is different for a seeder, as a seeder uploads pieces and not necessarily a complete file. We therefore assume that a seeder receives its altruistic utility after every uploaded piece. As a result, a leecher receives no utility while downloading but all utility when download completes, while a seeder receives utility for every uploaded file piece but no utility when the seeded file is uploaded completely.

The question is then how long utility lasts. We assume that, for both seeders and leechers, it does not last forever, but somehow decreases over time. Two examples of this are depicted in Figure 3.2, which shows utility that is constant for some time interval, after which it disappears, and utility that gradually decreases. How exactly utility decreases is specified in an agent's utility function and depends on the outcome. We redefine the utility function to incorporate time:

$$u_i : \Theta_i \times \Pi \times T \to \mathbb{R} \tag{3.10}$$

3.2.5 Model Applied To BitTorrent

The static and dynamic models we presented in Sections 3.2.2 and 3.2.4, respectively, are generic models that can be applied to file-sharing networks in general. In this thesis, our focus is on the BitTorrent file-sharing network. In this section, we apply our models to BitTorrent. We use the BitTorrent terminology as introduced in Section 3.1.

Remote Peers and Swarms

In BitTorrent, a file is identified by the `.torrent`-file, which also enumerates the pieces and lists the tracker server. Agents that exchange a file f, either by leeching or seeding it, announce themselves to the tracker server. From the tracker, a leecher receives a selection of other agents currently exchanging the file. As a result, any agent is aware of only a subset of the other peers exchanging the file. To model this, in the static model we supply every agent $i \in N$ with the subset $N_{r,i} \subseteq N \backslash \{i\}$; the agents in this set are i's remote peers. This does not suffice in the dynamic model, because the remote peer list is updated periodically from the tracker, or peers may go offline. In the dynamic model, an agent's remote peers at any moment in time are given by the function r:

$$r : N \times T \to 2^N \tag{3.11}$$

Because the remote peer list is received from the tracker for a certain file, it only contains peers exchanging that file. The group of all agents exchanging one file f is referred to as a swarm. In our model, we identify a swarm by grouping the outcome by file – each group represents a swarm.

In a swarm, agents barter for file pieces, which is expressed in our model as one download assignment for a specific piece, followed (in time) by upload assignments for that same piece.

Although, in the static model, it is possible to express multiple swarms in the network, we choose to consider a single swarm only. This is justified because, in BitTorrent, no state information is exchanged between swarms. An agent enters a new swarm without any knowledge of the other agents in the swarm. Even if an agent encounters an agent it bartered with before, it will exchange pieces only when the agent reciprocates pieces in the new swarm.

In the dynamic model, this no longer holds, because we allow agents to request and share new files, or stop sharing a file. This implies that they join or leave a swarm. However, as in the static model, we do assume that agents do not take bartering information from one swarm to another. When an agent encounters an agent it bartered with before in another swarm, that agent is treated as if it were a stranger, and a regular optimistic unchoke initiates a bartering session. This is exactly as specified in the protocol.

Implementation Of The Mechanism

In our model we assume a mechanism M that decides on the outcome. Intuitively, we thought of M there as some broker. In BitTorrent, such a broker does not exist. Instead, the mechanism is implemented by the agents themselves: a distributed implementation.

The mechanism executes the outcome rule g (Equation 3.5) to determine which file pieces are exchanged by which agents. The outcome rule takes as input the actions performed by the agents, i.e., the files they share and download (in other words, which swarms they are in), their upload and download capacities, number of upload slots, and seeding strategy. This is the same in BitTorrent, as we describe in Section 3.1.1. There, an agent i optimistically unchokes an agent j and sends it a file piece. The transfer speed of this file piece is given by i's upload capacity per upload slot, and j's download capacity. If i is among j's fastest uploaders, j will reciprocate by sending a piece back to i. Then, if j is among i's fastest uploaders, i will reciprocate and a bartering relationship is established until either i or j encounters a better bartering partner. In a bartering session, every agent decides for itself which file piece it requests from which bartering partner; in general, the Rarest Piece First policy is used to make this decision.

We see here how an agent's actions influence the outcome: making more upload capacity available, for example, allows an agent to barter with faster uploading agents.

One final word on the memory of the agents. BitTorrent bartering proceeds in rounds, and an agent sends pieces to the agents that sent it most pieces in the previous round. An agent does not take any information from earlier rounds into account. We therefore limit an agent's memory to be limited to one round.

Actions in BitTorrent

As we described in Section 3.2.5, the BitTorrent protocol requires an agent to specify an upload and download capacity, and number of upload slots. These are actions that the user will perform, typically by adjusting settings in the interface of its BitTorrent client.

Different clients allow a user to specify different settings. Most clients include settings to specify upload and download rate, and some clients include setting the number of upload slots as well. Some clients, such as Vuze, allow the user to control most of its behavior, while others, such as Tribler, offer a limited number of settings to the user.

In this thesis, we consider the settings that can be set from most clients, and we limit ourselves to those settings that influence the outcome. Below is a list of the settings we consider:

1. **Upload rate:** Specifies the maximum upload rate (from zero to unlimited) used by BitTorrent, expressed in KB/s. Distinguishes between upload rate when downloading and when not

downloading. The upload rate is set to unlimited by default,

2. **Download rate:** Specifies the maximum download rate, from zero to unlimited, expressed in KB/s. The download rate is set to unlimited by default,

3. **Upload slots:** Specifies the number of upload slots used,

4. **Seeding options:** The user can choose one out of four options to select its seeding strategy:

    a) Seed until the upload/download-ratio exceeds 1. This is the default option in BitTorrent, and has as effect that a local peer will upload as much as it downloads,

    b) Unlimited seeding,

    c) Seed for a specified amount of time,

    d) No seeding.

    Note that for these options to be effective, the user needs to leave the client running after the download completes.

In the dynamic model, agents can join or leave the network as well.

At any moment in time, for every setting available to the user, something is specified: either explicitly by the user, or implicitly in the form of some default setting. We refer to specifying a combination of settings above as performing an *action*.

Specifying one single setting, such as the number of upload slots, we refer to as performing a *subaction*. For every setting above, we introduce a subaction space: $X_{i,u}, X_{i,d}, X_{i,k}$, and $X_{i,e}$, respectively. Then, for every agent i, we introduce a set of actions X_i, which is the Cartesian product of $X_{i,u}, X_{i,d}, X_{i,k}$, and $X_{i,e}$. Finally, by x_i we denote the action played by agent i; $x_i \in X_i$.

As we mentioned in Section 3.2.2, every agent has a set of actions at its disposal, and this set depends on the agent's type. We elaborate on this in Section 3.3, where we group the agents in different classes based on their action space.

When the dynamic model is applied, not all settings may be equally relevant. For example, if our goal is to investigate the effect of sharing and requesting files on agent utility, we are not interested in the number of upload slots at any moment in time. For this reason, we specifically allow only some of the subactions to be dynamic, while the remaining subactions are static. Such a model is only partially dynamic.

Of the six settings, the subactions of sharing and requesting files will be performed most often, as new content comes available all the time, or users delete files.

Specifying upload and download rates will be performed less frequently. This specifically holds for specifying the number of upload slots, which usually will be set once by more advanced users, while the majority of the users will not set it at all but use the default value. However, as we show in Section 4.1, setting the number of upload slots strategically can have a significant impact on utility, which calculating advanced users may use to their advantage. Such users will also change their upload and download speed more frequently than other users.

Finally, in practice, users will not often change their seeding strategy. We show the reason for this in Section 4.1: the other subactions influence utility in a different way than the seeding strategy subaction does. Therefore, the utility derived from this subaction can be considered more constant than the utility derived from the other subactions, which is more dependent on the file being downloaded and the agents downloading that file. Of course, it is our goal to motivate agents to choose a sharing setting here.

3.2.6 Utility Functions

In Sections 3.2.2 and 3.2.4, we introduced an agent's utility function as a function that, given an agents type, calculates some measure of the benefit an agent receives from the mechanism's outcome. In this section, we consider utility functions in BitTorrent in more detail and provide some examples.

Measurements have shown that in practice, users are reluctant to share files. Apparently, uploading incurs them a cost. We can model this with a simple utility function.

Consider a function that values the benefit of the download of a file at some rate twice as much as the cost of uploading at the same rate. In other words, this function returns the sum of all download rates, minus half the sum of all upload rates. We need some notation to formally describe this function. Denote by a an assignment; $a \in \Pi$. Denote by $a_{d,i}$ the subset of a in which i is the downloading agent. Similarly, denote by $a_{u,i}$ the subset of a in which i is the uploading agent. We introduce the function $z : \pi \rightarrow \mathbb{R}^+$ that outputs the transfer rate of a download assignment, i.e. $r(n_1, f, n_2, p) = p$. The utility function is then:

$$v_i(\theta_i, a) = \sum_{x \in a_{d,i}} z(x) - \frac{1}{2} \sum_{x \in a_{u,i}} z(x) \tag{3.12}$$

Obviously, this example is simplified, in that it does not discriminate between files that have or have not been requested, all transfer rates are valued equally for all files, etc. However, one recognizes these factors can all be expressed in a utility function.

The utility function above makes two unrealistic assumptions. First, it assumes that the higher the transfer rate, the higher the utility. Second, it assumes that users are carefully observing their upload rate. We first address the first assumption. It is reasonable to assume that the user will become saturated at some point. This is easier to see in the dynamic model: after downloading movies constantly for 30 days the user may want to actually watch some movies, and downloading yet another movie will only marginally increase its utility, if at all.

However, the same may hold for download rates: the increase of utility may diminish with the transfer rate. We could express this by making the benefits grow logarithmically instead of linearly with the transfer rate. The same will then apply to the costs: when uploading at a substantial rate, extra uploads may impose less of a cost. The above utility function would then become:

$$v_i(\theta_i, a) = \log\left(1 + \sum_{x \in a_{d,i}} r(x)\right) - \frac{1}{2} \log\left(1 + \sum_{x \in a_{u,i}} r(x)\right) \tag{3.13}$$

When we do consider download (and upload) volumes, we assume that the increase of utility decreases with the downloaded volume, and vice versa for the uploaded volume:

$$v_i(\theta_i, a) = \log\left(\int_{t=0}^{\infty} \sum_{x \in a_{d,i}} r(x)\right) - \frac{1}{2} \log\left(\int_{t=0}^{\infty} \sum_{x \in a_{u,i}} r(x)\right) \tag{3.14}$$

The second assumption of Equation 3.12 is that users would carefully observe their upload rates. In practice, this will not be the case. Users will notice when they are uploading, and that may reduce their utility compared to not uploading with some constant:

$$v_i(\theta_i, a) = \begin{cases} \sum_{x \in a_{d,i}} r(x) & \text{if } \sum_{x \in a_{u,i}} r(x) = 0 \\ \sum_{x \in a_{d,i}} r(x) - c & \text{otherwise, with } c \in \mathbb{R}^+ \end{cases} \tag{3.15}$$

Similarly, the costs can relate to the fraction of upload capacity used. If only a small portion is used, this does not affect the utility. If a large portion is used, utility is negative as no matter how much is downloaded, the costs of using that much capacity is too high. For upload rates in between, utility drops with some constant:

$$v_i(\theta_i, a) = \begin{cases} \sum_{x \in a_{d,i}} r(x) & \text{if } \sum_{x \in a_{u,i}} r(x) < \frac{c_u}{4} \\ \sum_{x \in a_{d,i}} r(x) - c & \text{if } \frac{c_u}{4} < \sum_{x \in a_{u,i}} r(x) < \frac{3c_u}{4} \in \mathbb{R}^+ \\ c & \text{otherwise, with } c \in \mathbb{R}^- \end{cases} \tag{3.16}$$

Another example of a utility function is motivated by possible contract terms an ISP may impose. With some contracts, users are allowed to upload only a fixed amount of data. When this is exceeded, some fine is imposed. Users will want to avoid this fine, and will value an assignment that exceeds this limit very low.

$$v_i(\theta_i, a) = \begin{cases} \sum_{x \in a_{d,i}} r(x) & \text{if } \int_T \sum_{x \in a_{u,i}} r(x) < c, \text{ with } c \in \mathbb{R}^+ \\ k & \text{otherwise, with } k \in \mathbb{R}^- \end{cases} \tag{3.17}$$

We explain in Section 3.1 that in BitTorrent, users need to upload to be able to download. To most users, uploading is then no longer a cost. Their utility is directly linked to their download rate:

$$v_i(\theta_i, a) = \sum_{x \in a_{d,i}} r(x) \tag{3.18}$$

Finally, we consider a utility function for which the costs depend on the file that is uploaded. This is motivated by the fact that users illegally share copyrighted material, and are afraid of getting caught. Users may want to share such files unless they are downloaded to often, for example, or these files may be downloaded and shared automatically. We introduce the function y that returns the uploaded files in an agent's upload assignments $a_{u,i}$, and the set $I \subseteq F$ that holds the illegally shared files agent i. The utility function then becomes:

$$v_i(\theta_i, a) = \begin{cases} v_i(\theta_i, a) = \sum_{x \in a_{d,i}} r(x) - 2\sum_{x \in a_{u,i}} r(x) & \text{if } y(a_{u,i}) \bigcap I \neq \emptyset \\ v_i(\theta_i, a) = \sum_{x \in a_{d,i}} r(x) - \frac{1}{2}\sum_{x \in a_{u,i}} r(x) & \text{otherwise} \end{cases} \tag{3.19}$$

From the above, we find that there are many different utility functions, with very different objectives. This motivates that maximizing social welfare in a file-sharing network is a difficult task.

3.2.7 Omitted Aspects

Every model is a simplification of reality and does not incorporate all aspects of what is modelled. Below, we list the properties we do not consider in our model:

1. Irrational behavior: we assume all agents are rational. Agents do not need to be the same, and may have different motivations for joining the network. However, we require these motivations to be rational, and we require the agents to try to maximize some utility function. We require this rationality because we model the agents to choose their actions based on the assumption that the other agents are rational, as we motivate in Section 3.2.2,

2. Network topology: in our model, all agents are connected to all other agents in the network. P2P networks are typically implemented as overlay networks on top of the Internet. We assume that the Internet provides our network with the possibilities that all agents are connected as modelled,

3. Transfer speeds: the transfer speed for a file may be affected by the underlying network, and not only depends on the uploading and downloading agents' connection capacities. We ignore this and assume that the file transfer rate is only limited by the users' capacities,

4. File identification: we assume all files can be identified and distinguished. In reality, in file-sharing networks it is difficult to identify files because of naming conventions applied by each user. Additionally, files with identical names can be different because of different encodings,

5. Identification of agents: we assume it is possible for both the broker *M* as for the agents to identify and locate other agents,

6. Costs of assignments: in our model we assume that uploading incurs a cost because it puts load on the users Internet connection. It is very likely that users have other, less rational, motivations for uploading being a cost,

7. In Section 3.1.1, we mentioned that a peer always sends the requested file piece. However, according to the protocol, pieces are implicitly correlated with request messages, and it is possible for an unexpected to arrive. In practice, this happens only in the end-game, when the local peer has only a few missing pieces and does many requests to many agents. In our model, we assume that a peer always sends the requested file piece.

3.3 Agents in BitTorrent

The previous section introduced a model for P2P file-sharing networks from a game-theoretic perspective. We modelled agents connected to the network as rational, self-interested entities that strive to maximize their utilities, i.e. they try to download their desired files as fast as possible. If possible, such rational agents may opt to manipulate the outcome in their favor, possibly at the expense of the other connected agents. In this section, we investigate the agents in a file-sharing network in more detail, and motivate our focus in this thesis on one particular type of agents: the lazy free-riders.

Meulpolder et al. [28] distinguish three classes of agents in BitTorrent: altruistic agents, lazy free-riders and die-hard free-riders. The altruistic agents share files because they want to, and if all agents were altruistic, no incentive mechanism would be necessary. The die-hard free-riders will go to great lengths to free-ride. They have both the skills and the motivation to create or obtain cheating clients to prevent any uploading at all. Although this is possible, in practice such behavior is not often seen in BitTorrent [28].

However, we do find that agents go off-line immediately after their download finishes, or that they make only part of their upload connection available to the network. Such agents we refer to as lazy free-riders: agents that free-ride whenever they can, but that will not actively search for possibilities. Their options are limited to the options provided by the interface their network client provides.

There is one additional class of agents: the byzantine agents. This class of agents may display any behavior to obtain their desired files, including exploiting the file-sharing network protocol, breaking into other users' computers, or even breaking into their houses and stealing their computers. The difference between byzantine agents and die-hard free-riders is, that the latter use the file-sharing network to obtain their desired files. They faithfully execute the network protocol, but may not be faithful in their actions using the protocol, for example by obtaining a cheating client, as we mentioned above.

Distinguishing only four classes of agents is a simplification: in reality, there are more classes of agents. Users in each class will be somewhere 'between' altruism and die-hard free-riding in the lengths they will go to to free-ride. The actions available to the different classes of agents are visualized in Figure 3.3. The byzantine agents have all actions at their disposal. The die-hard free-riders' actions are limited to all actions that can be implemented using the file-sharing network. This includes installing or developing other clients for the network. The lazy free-riders' actions are confined to the actions that can be performed using the official client, and are thus a subset of the

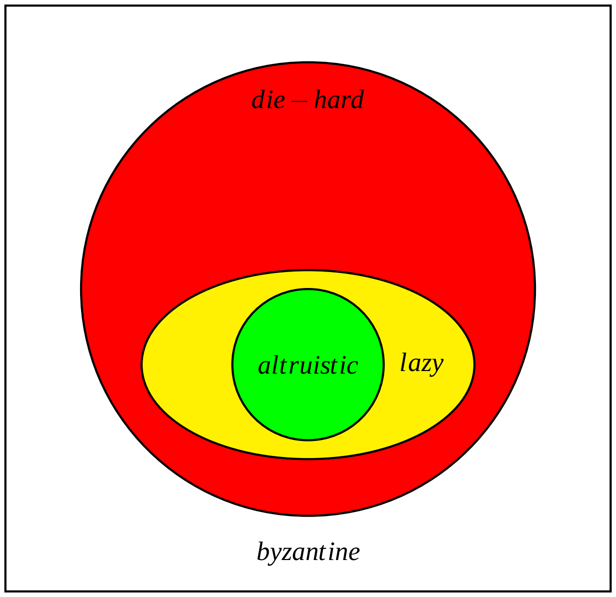


Figure 3.3: *Available strategies for different agent classes.*

die-hard free-riders' actions. Obviously, the set of the altruistic agents' actions is the smallest, and a subset of all other sets of actions.

It should be noted here that the lazy free-riders and altruistic agents share the same strategy space, as both use the official client. The altruistic agents, however, refrain from using some of the actions available to them. One example is that downloaded files are automatically shared with the network. A lazy free-rider will decide not to share such a file, but an altruist is intrinsically motivated to cooperate with the network and will not consider some of the actions available to him.

Our distinction of different classes allows us to model different classes of agents differently, as depicted in Figure 3.4. For lazy free-riders, the agent is the user. The user's interface to the network is the client, but, by definition, the lazy free-rider does not replace or alter this client. Therefore, the user's action space is limited to the actions the client has to offer. We depict this in the upper figure in Figure 3.4. This figure applies to altruistic agents as well, although altruistic agents will in addition refrain from performing some of the actions available to them.

With die-hard free-riders, the agent comprises the user and the client, its interface to the network (the lower figure in Figure 3.4). Because the client is under the user's control, the agent can perform any action possible within the network.

In Section 3.2.6, we provide utility functions for a number of realistic types of users. All of these users are lazy free-riders. This shows that there are in fact many more agent classes than the four we distinguish here.

Previous research in BitTorrent has focused on die-hard free-riding, and identified methods that allow an agent to download without uploading. Examples are Sirivianos et al. [46], Piatek et al. [35], and Levin et al. [26]. However, as stated before, die-hard free-riding is hardly seen in practice. In this thesis, we focus on lazy free-riders. We are interested in how they can improve their utility using only actions made available through the interface of official clients, and how this affects the utility of the other agents in the network. Based on the model derived in this chapter, we derive theoretical results for this in Chapter 4. These results are experimentally verified in Chapter 5.

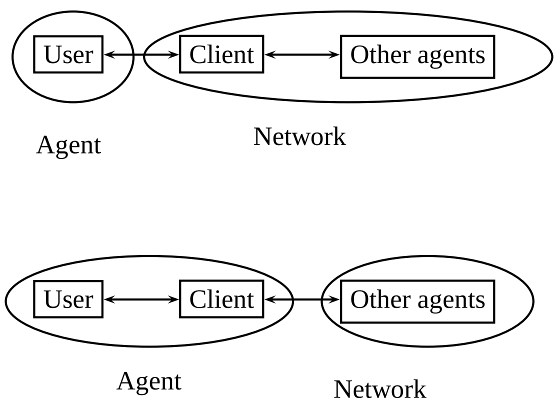


Figure 3.4: *An altruistic user (top) interacts with its client as if it were the network. This user does not change any settings, but instead accepts all the defaults. A lazy free-rider (bottom), on the other hand, has full control over its client and uses its client to interact with the network. This type of user may change any setting that is available through the interface.*

Chapter 4

Application of the Model to BitTorrent

In Chapter 3, we developed a model of a peer-to-peer file-sharing network. The agents of such a network derive their utility from the outcome: a single decision which specifies for every agent, which file pieces are downloaded from which agents, and the order and transfer rate with which these pieces are downloaded.

In this thesis, we focus on the BitTorrent file-sharing network. There, this outcome is determined distributedly, by the clients to the network. In this chapter, we investigate this outcome. In Section 4.1, we show that the outcome groups agents on their upload capacity, and that in general, agents barter only with agents in their group. We show that this, combined with BitTorrent's optimistic unchoke policy, increases the download completion time and uploaded data volume of the faster agents in the network, while it decreases that of the slower agents. In Section 4.2, we investigate possibilities for manipulation BitTorrent's outcome provides, and how much agents can win by manipulating the outcome. In Section 4.3, we extend the model with the BarterCast reputation system, in order to identify whether agents have an incentive to use this system.

4.1 Network Composition and Outcomes

In this section, we show that in BitTorrent's outcome, agents are grouped on their upload capacity, and barter only with agents from their own group. We start with an overview of related literature that observed this grouping in practice (Section 4.1.1), and then show that this follows naturally from our model (Section 4.1.2). We use the results from our model to explain other observed phenomena in BitTorrent's outcome in Section 4.1.3, and draw conclusions in Section 4.1.4.

4.1.1 Clustering Phenomena

Clustering

In Chapter 3, we modeled the users of a file-sharing system as utility-maximizing agents. Agents derive their utility from their download rate. Therefore, all agents want to download from the fastest uploading agents in the swarm – including these fastest agents. Because agents have a limited number of upload slots, the fast agents reciprocate to their fastest uploaders, and not to the slower agents. Because these slower agents do not receive pieces from the fastest agents, they lose their interest in these agents and try to obtain pieces from slower uploading agents. Intuitively, this demonstrates that agents end up bartering only with agents with similar upload capacity.

Consider a network consisting of 10 agents, a_1 through a_{10}, where a_1 has the highest upload capacity and a_{10} the lowest. Every agent has 3 upload slots (with the optimistic unchoke slot excluded). In this example, all agents prefer interacting with a_1 over interacting with any other agent.

This means that a_1 gets to decide on which agents it interacts with. Best candidates for a_1 are a_2, a_3 and a_4. Agent a_2 now has one upload slot dedicated to a_1. Best candidates for the remaining upload slots are a_3 and a_4, etc. This means a cluster is formed by a_1 through a_4. Similarly, a_5 through a_8 form a cluster, and a_9 clusters together with a_{10}.

In this example, we find that agent a_1 provides the best service (fastest uploads) to all peers, and it is in its best interest to interact with a_2, a_3 and a_4. However, these agents receive this high quality service in exchange for their lower quality service. When agents have Equation 3.12 as utility function (utility is download rate minus half the upload rate), a_4's utility outweighs a_1's utility. In fact, if a_1 would upload just slightly faster than a_5, it would still connect to a_2 through a_4, and receive the same service from them, while delivering suboptimal service.

Clustering is problematic for two reasons. First, it introduces possibilities for manipulation, as described above. Second, since there is no data exchanged between clusters, clusters lock content. If a_9 and a_{10} together do not have all pieces of the file, they may never be able to finish the download. This means that all pieces of a file must be present in every cluster to ensure that all agents can finish their download. In practice, this means that every cluster must be connected to a seeder. Because seeders prefer uploading to the faster leechers, it is unlikely that that will happen.

Stratification

Clustering is not a purely theoretical phenomenon. Bharambe et al. [8] and Legout et al. [25] describe clustering from measurements in real BitTorrent systems. Both show that clusters are formed by the faster, medium and slower peers in the network. Legout et al. find that although data is exchanged from the faster to the slower peers, this is due to optimistic unchoking. This means that the slower peers in the network depend on optimistic unchokes for a significant part of their pieces.

Interestingly, both find that the faster peers in a cluster upload significantly more pieces than they download, while the slower peers in a cluster upload less than they download. However, this does allow these faster peers to finish their download more quickly. Depending on the agent's utility function, this observed inequality in uploaded volume could provide the agent with an incentive to choose its actions strategically.

What Bharambe et al. and Legout et al. observed is not the clustering as we described above, where the network falls apart into disjoint clusters of $p+1$ agents. Instead, they find something similar, to which we refer as *stratification*, following Gai et al. [20]. For clarity, we now define the following terms:

Definition 4.1. *(Clustering)* Clustering *is a process that groups agents together based on their slot capacities.*

Definition 4.2. *(Segmentation) Segmentation is a clustering process which results in the network being split into disjoint groups (segments) of agents. Pieces are only exchanged between segments as a result of optimistic unchokes, not by bartering.*

As a result, we refer to the clusters we described in Section 4.1.1 as segments from now on.

Definition 4.3. *(Stratification) Stratification is a clustering process which results in a network in which for every agent $i \in N$, there is a range of slot capacities centered around i's slot capacity such that i barters exclusively with agents that have a slot capacity that falls within this range. Pieces are only exchanged with the other agents as a result of optimistic unchokes, not by bartering.*

Stratification may lead to segmentation. When segments are large, it is possible for stratification to occur within segments.

Both Bharambe et al. [8] and Legout et al. [25] provide two explanations for stratification to occur instead of segmentation. First is that for the segmentation into many small clusters of size

$p+1$, as we described in Section 4.1.1, all agents must have the same number of upload slots. In BitTorrent, this number is not fixed but depends on the upload capacity. Both describe, based on measurements, that this prevents segmentation, but that it does not prevent stratification.

The second explanation for stratification to occur is that agents do not know all other agents in the network, as we assumed above. In Section 3.1.1, we described that agents get a random selection of other peers in the swarm from the tracker. As a result, an agent is not able to probe all agents in the swarm. As every agent receives a different remote peer list from the tracker, every agent will connect to other agents and one would expect that segments will not be formed. This is true to some extent – Bharambe et al. do not observe small segments, but have found large segments. Stratification is observed despite the random selection of remote peers.

Until now, we assumed that agents rank other agents based solely on their upload capacity per upload slot. However, in BitTorrent, this is only a part of how agents rank other agents: we describe in Section 3.1.1 that a lazy free-riding agent barters only with agents that have complementary pieces, i.e., have pieces that i does not have, and that i prefers agents with more complementary pieces. When we incorporate this in our analysis, it is no longer possible to rank agents globally based on their upload capacities alone, and second, it means that an agent's ranking of the other agents may change whenever it exchanges a file piece.

However, Legout et al. [25] and Bharambe et al. [8] both show from measurements that stratification does occur despite this. A possible explanation is given in Legout et al., where the authors argue that BitTorrent's Rarest Piece First piece selection policy distributes file pieces over a swarm such that agents remain interested in other agents as long as the transfer speed is large enough, or until the download is almost finished and it becomes harder to find missing pieces.

4.1.2 Clustering Explained From The Model

In the previous section, we explained intuitively how clustering and stratification arise, and gave an overview of the literature that observed this phenomenon in practice. We now prove that clustering and stratification follow naturally from the models we presented in Section 3.2.

Preliminaries

In our static model, an agent i has an upload capacity $c_{u,i}$ and k_i upload slots. This number is with the optimistic unchoke slot included. In the following, we will often consider the upload slots determined for bartering only. For any agent i, we denote by $k_{i,b}$ the number of upload slots i assigns to bartering.

We use the static model for our analysis in this section, and extend it to the dynamic model later in this section. Specifically, this means that agents do not change k_i or $c_{u,i}$. We consider a single swarm only; N is the set of all n agents in that swarm.

The users we consider are lazy free-riders (see Section 3.3). By definition, these users use official or mainstream clients. In the BitTorrent file-sharing network, such clients include the official BitTorrent client and Tribler. We assume that all clients used by lazy free-riding users have the same utility function: to maximize the download rate, given constraints specified by the user: the number of upload slots k_i, and upload and download capacity. We refer to this utility function as the *client utility function*. Note that this client utility does not necessarily match the user's utility function – we investigate this further in Section 4.2.

Ranking

The client utility implies that agents prefer downloading from faster uploading agents over downloading from the others. To facilitate our reasoning in this section, we first introduce the concept of a *global ranking* that orders all agents in N based on the upload capacity they offer per upload slot. We refer to this metric from this moment onwards as *slot capacity*: agent i's slot capacity equals $\frac{c_{u,i}}{k_i}$.

Definition 4.4. *A* global ranking *is an order over the agents $i \in N$ based on their slot capacity $\frac{c_{u,i}}{k_i}$, where ties have equal rank. $r : N \to \mathbb{N}^+$ gives an agent's rank in the global ranking, and $r_i \leq r_j$ iff $\frac{c_{u,i}}{k_i} \leq \frac{c_{u,j}}{k_j}$.*

A *higher*, or *better*, rank corresponds to higher slot capacity. A visualization of a ranking is given in Figure 4.1. The lowest rank is 1; the highest rank possible is n if there are no ties. If there are ties, we say that these agents *share a rank*.

Proposition 4.1. *The global ranking is acyclic.*

Proof. This follows directly from Definition 4.4. □

Note that in Definition 4.4, agents are ranked on their upload capacity alone. In Section 3.1.1, we described that when an agent decides on which agent to download from, it not only considers the other agent's upload capacity, but also its complementary pieces. Incorporating complementary pieces results in a ranking that is no longer global, but local to each agent. In the remainder of this section, we find that the most important property of the global ranking is that it is acyclic (Proposition 4.1). Gai et al. [19] prove not only that a ranking based solely on complementary pieces is acyclic, but also that any linear combination of acyclic global and complementary rankings is acyclic. In this thesis, we ignore the additional ranking on complementary pieces, and focus on upload capacity alone. Based on the work by Gai et al., this should not affect our results in this section.

If an agent downloads from another agent, either through bartering or optimistic unchoking, it discovers the other agent's slot capacity. When all agents have interacted with each other, every agent knows the slot capacity of every other agent in the network. Because in the static model agents do not perform actions, their slot capacities do not change during the game and all agents have equal knowledge.

Proposition 4.2. *With the BitTorrent protocol, an agent knows the slot capacities of all other agents in the network if and only if it has downloaded from all other agents.*

Proof. If an agent downloaded from all other agents in the network, it knows all slot capacities from experience. This proves the if-part. Agents exchange protocol messages and file pieces only. Protocol messages do not include exchanging information about slot capacities, so that information must come from experience, i.e., downloading from every other agent in the network. This proves the only-if-part. □

Proposition 4.3. *If every agent employs the client utility function, knows the slot capacities of all other agents in the network, and agents do not perform actions during the game, then all agents employ a preference over the other agents that is equal to the global ranking.*

Proof. Every agent i prefers an agent m over agent n if

$$\frac{c_{u,m}}{k_m} > \frac{c_{u,n}}{k_n}$$

because m gives i a higher utility than n, and i prefers m and n equally if their slot capacities are the same. This is the same metric as used in the global ranking. Because agents do not perform actions, an agent's ranking of another agent is constant during the game. □

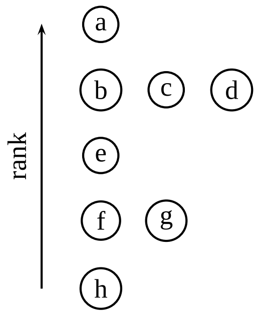


Figure 4.1: *Ranking of eight agents. Agent a is ranked best and has highest rank. Agents b, c and d are ranked second best and have equal slot capacities. Agent h has lowest slot capacity and is ranked worst.*

Bartering Relationships

Upload slots put a restriction on the number of agents to which an agent uploads simultaneously. As a result, agents need to be strategic in who they upload to. They prefer uploading to higher-ranked reciprocating agents as those will provide good utility in return. For downloading, there is no such restriction, and an agent will not refuse to download even from its slowest uploaders. It will, however, refuse to upload to the slowest uploaders, because it uploads to its $k_{b,i}$ fastest uploaders only.

If an agent j is among i's fastest uploaders, i reciprocates to j. In return, j reciprocates to i if i is among j's fastest uploaders. We say that agents i and j form a *bartering relationship*, exchanging pieces for a longer period of time, until either i or j encounters a better partner to barter with. It follows that no agent i is in more than $k_{b,i}$ relationships. Note that an optimistic unchoke is not a bartering relationship but may result in one if both agents involved reciprocate to each other.

For any agent, being in a relationship with any other agent is better than being in no relationship, because then at least some utility is obtained. However, higher-ranked agents provide more utility than lower-ranked agents. Consider three agents a, b and c, each with one upload slot, with $r(a) > r(b) > r(c)$. Suppose that agents a and c are in a bartering relationship with each other, while b is in no relationship. Agent b prefers to be in a relationship with a or c over its current situation and unchokes both in the hope of forming a relationship. Agent c prefers its relationship with a because $r(a) > r(b)$, but a prefers b over c and breaks its relationship with c for a relationship with b. We observe the following:

1. The broken relationship between a and c is *initiated* by agent b that *proposes* to a. Therefore, b is the *initiating agent*;

2. Any agent can take the initiative to propose to another agent. In BitTorrent, proposing is done through optimistic unchokes;

3. The utility of a improved. We say a played a *better response* to b's initiative by breaking its current relationship for a relationship that gives it higher utility;

4. In this network of three agents, a cannot obtain higher utility. The better response was therefore a *best response*;

5. The utility of a and b increased, while c's utility dropped to zero. A rational agent will never break a relationship for a relationship that provides it lower utility. If an agent's utility over a given upload slot decreases, it must be because its partner broke the relationship playing a better response and leaving the agent with zero utility.

If a best response is mutual, i.e., both the initiating and responding agent cannot be in a relationship that give it higher utility, then this relationship will not be broken.

Definition 4.5. (Stable Relationship) *A stable relationship is a bartering relationship that is broken by neither of the agents to obtain higher utility.*

It is important to realize here that although initially every agent prefers bartering with the best-ranked agents, these best-ranked agents reciprocate to best-ranked agents only. Therefore, the worse-ranked agents do not obtain any utility from the best-ranked agents, and as a result, the higher ranked agents are not attractive bartering partners for the worse-ranked agents so they will not form a stable relationship.

Lemma 4.4. *An agent i playing a series of better responses ends in a stable relationship.*

Proof. By Proposition 4.1, the ranking is acyclic, and since N is a finite set, there is one highest rank from which agents will reciprocate to i. □

The stable relationship forms an important concept in the remainder of this section because it maximizes an agent's utility over the corresponding upload slot. As the agents in a file-sharing network are utility-maximizing agents, all agents strive to be in stable relationships. From Lemma 4.2 and Proposition 4.3, we know that when all agents in the network have interacted with all other agents in the network, all agents rank the other agents according to the global ranking. If we can use the global ranking to find stable relationships for all agents in the network, then we can have every agent maximize its utility.

Network Configurations

Agents are in many relationships simultaneously. We refer to the collection of all (not necessarily stable) bartering relationships in a network as a *network configuration*. Some configurations are more beneficial to an agent than others, because in such a configuration it can barter with agents with higher slot capacity, or even be in a stable relationship which maximizes its utility over the corresponding upload slot. Every agent i prefers to be in $k_{b,i}$ stable relationships, because only then its total utility is maximized.

Definition 4.6. *A* stable configuration *is a file-sharing network in which all bartering relationships are stable, and no additional stable relationships can be made.*

An additional stable relationship is a relationship that is formed over previously unused upload slots, so no relationship needs to be broken to enable it.

In BitTorrent, agents decide for themselves who to barter with, based on their client utility function. The following lemma shows that this way, a stable configuration is formed:

Lemma 4.5. *If every agent knows the slot capacities of all other agents in the network and actions are fixed, then a stable configuration is formed.*

Proof. Assume that every agent knows the slot capacities of all other agents, and that actions are fixed. Then, every agent knows the global ranking. Any existing stable relationships will not be broken. An agent in an unstable relationship replaces that either by a stable relationship through a series of better responses (Lemma 4.4), or the relationship is broken by its partner playing a better response. If an agent i is not in $k_{b,i}$ relationships, it will accept any agent to form an unstable relationship. If no agent is willing to accept that relationship in response, no stable relationship is possible. This leads to a configuration in which all relationships are stable and no more relationships can be added, which by Definition 4.6 is a stable configuration. □

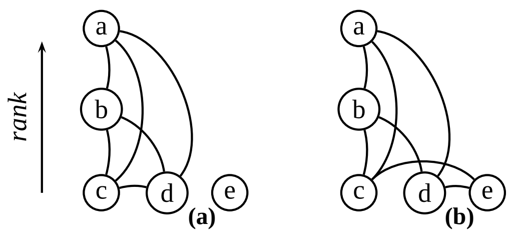


Figure 4.2: *Two configurations of the same network. Every agent has 3 upload slots. Both configurations are stable because all relationships are stable, and no additional relationships can be made. In configuration 4.2a, agent e receives no utility. In configuration 4.2b, the relationship between c and d is broken for a relationship between c and e, which enables an additional relationship between d and e. Now e is in two relationships and receives utility, without lowering the other agents' utilities.*

A network can have multiple stable configurations, and in each of these configurations, an agent can receive different utility. Consider for example the two configurations of the same network in Figure 4.2. Both configurations are stable. In Figure 4.2a, all agents are in three bartering relationships, except for agent *e*, which is in none. Because none of the other agents have upload slots available, *e* can be in no relationships. However, we can break the stable relationship between *c* and *d* for a stable relationship with *c* and *e*. This enables an extra relationship between *d* and *e*, as shown in Figure 4.2b. Note that, again, all relationships are stable. Agents *a* through *d* receive the same utility, but now *e* is in two relationships while it was in none. This configuration is clearly preferred over the first configuration.

In the configuration in Figure 4.2a, it is possible for *e* to improve its utility without lowering the utilities of any of the other agents. In Figure 4.2b, this is not possible. If we want to increase *e*'s utility there, an existing relation has to be broken, lowering another agent's utility. Adapting Leyton-Brown and Shoham [9], we refer to configurations as the one in Figure 4.2a as a Pareto-dominated configuration, i.e., a configuration in which some agent can be made better off without making any other agent worse off. Formally:

Definition 4.7. *(Pareto domination) Configuration c* Pareto dominates *configuration c' if for all $i \in N$, $u_i(c) \geq u_i(c')$, and there exists some $j \in N$ for which $u_j(c) > u_j(c')$.*[1]

The configuration in Figure 4.2b is not Pareto dominated by another configuration. Such a configuration is Pareto-efficient:

Definition 4.8. *(Pareto efficiency) Network configuration c is* Pareto efficient *if there does not exist another configuration c' that Pareto dominates c.*

A network can have multiple Pareto efficient configurations. A configuration does not need to be stable to be Pareto efficient, as Figure 4.3 illustrates. Note that whenever an agent improves its utility with a better response, the utility of its current bartering partner decreases. On the other hand, not every stable configuration is Pareto efficient, as we demonstrated earlier in this section with Figure 4.2.

There can be multiple stable configurations of a network. The configuration in Figure 4.2b is stable and Pareto efficient. If we switch agent *d* with agent *e* , we find another. Without knowing the agents' utility functions and a concise definition of social welfare, it is impossible to identify a single

[1]We overload notation here: $u_i(c)$ is the utility agent i receives in configuration c.

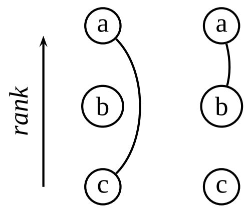


Figure 4.3: *Two configurations of a network in which all agents employ one upload slot. The left configuration is unstable because a can play a better response by bartering with b, as it does in the right configuration which is stable. Both configurations are Pareto efficient.*

best stable configuration. Instead, the notion of Pareto efficient stable configurations gives us a set of non-comparable optima. We want the agents to reach some Pareto efficient stable configuration, but we are indifferent about which one is reached. Whereas Lemma 4.5 proves that a stable configuration is reached, the following theorem proves that we cannot guarantee that a Pareto efficient configuration is reached.

Theorem 4.6. *If a network configuration is stable but not Pareto efficient, a Pareto efficient configuration is never reached.*

Proof. Assume a network configuration is stable but not Pareto efficient. In a stable configuration, every agent maximizes its utility over all upload slots it uses for bartering, so no agent will break any relationship it is in. No additional relationships can be established in a stable configuration. Therefore, once a stable configuration is reached, it does not change, and thus will not be Pareto efficient unless it already is. □

The configuration that emerges using the distributed implementation is not necessarily optimal. This analysis holds for the static model. Later in this section, we investigate these results for the dynamic model. There, we find that even though this result holds for the dynamic model as well, it is not as dramatic as it may seem at first sight.

At this point, we have shown that with a distributed implementation such as BitTorrent, a stable configuration is reached. In the next section, we investigate stratification and segmentation in stable configurations.

Stratification and Segmentation

We started this section with a description of two forms of clustering: segmentation (see Definition 4.2) and stratification (see Definition 4.3). With segmentation, the network is segmented into disjoint clusters, while with stratification, agents are in bartering relationships only with peers with similar upload capacity, but the network is not necessarily segmented. In this section, we prove that stratification and segmentation follow naturally from our model. We start by observing that a stable configuration is segmented when certain conditions apply.

Theorem 4.7. *In a stable configuration of a file-sharing network consisting of at least $2k+1$ agents, where every agent uses k upload slots for bartering, an agent i is in bartering relationships with agents ranked between $r(i)+k$ and $r(i)-k$ exclusively.*

Proof. We use proof by induction. Let P be the proposition in a stable configuration as mentioned, an agent i barters with agents ranked between $r(i)+k$ and $r(i)-k$ exclusively.

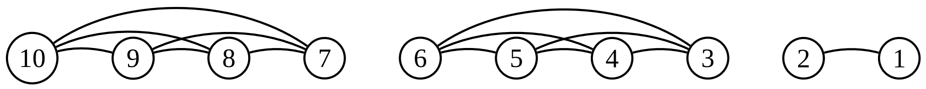


Figure 4.4: *A network of 10 agents, ordered by descending rank. All agents employ 3 upload slots, which causes the network to segment.*

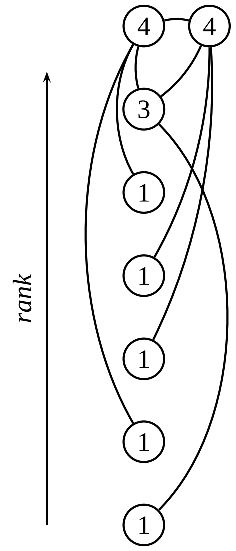


Figure 4.5: *A network where the number of upload slots varies over the agents. Numbers denote the number of upload slots an agent has. The maximum number of upload slots used is 4, but some agents barter with agents 5 ranks below them. Even though agents barter with similar-ranked agents as much as possible, we cannot put a bound on the rankings of the agents any given agent is bartering with.*

Base case: P is true for agents with the highest rank n: if no ranks are shared amongst the first $k+1$ ranks, agent i is in stable relationships with the k agents ranked just below i. Otherwise, stable relationships are formed with agents of a shared rank, and i barters with the same or fewer ranks below it.

Induction step: Assume P is true for ranks n through a, $a > 1$. Let $b = a - 1$. Two disjoint cases apply, for which we use case analysis.

Case 1: No agent ranked b is in a stable relationship with any agent ranked $\geq a$. Then, because the configuration is stable and every agent has k upload slots, neither is any agent ranked lower than b. Similar to the base case, agents ranked b are in relationships with agents ranked between b and $b-k$ exclusively.

Case 2: At least one agent ranked b is in a stable relationship with any agent ranked $\geq a$. This means that less relationships with equally- or lower-ranked agents are possible than with Case 1, so these agents ranked b cannot barter with more ranks than agents ranked b in Case 1. Therefore, agents ranked b are in relationships with agents ranked between b and $b-k$ exclusively.

This implies that if P holds for some rank a, it also holds for rank $a-1$. By the principle of induction, P must be true. $\square$

We explained in Section 4.1.1 that segmentation is an extreme form of stratification. Figure 4.4 shows how segmentation emerges in a network where no ranks are shared and all agents employ the same number of upload slots.

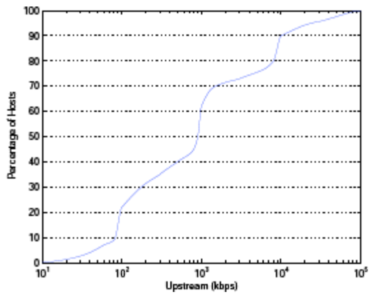


Figure 4.6: *Cumulative upload capacity in the Gnutella file-sharing network. Taken from Gai et al. [20], based on an earlier study by Saroiu et al. [40].*

The proof for Theorem 4.7 depends on all agents having *k* upload slots. In BitTorrent, this is not the case in general, because agents can specify this number themselves. Because of this, we cannot prove stratification in networks where ranks are shared but agents can choose their number of upload slots. In such networks, whether stratification emerges depends on the ranking of the agents, and the number of upload slots every agent uses. An extreme example is given in Figure 4.5, where one, three or four upload slots are used, but the order of the agents is such that some agents barter with agents five ranks lower. However, this does not mean that stratification does not occur as in the above, we have demonstrated that, where possible, agents barter with agents with similar upload capacity.

Figure 4.6 shows the results from a measurement study by Saroiu et al. [40] in the Gnutella file-sharing network in 2002. It is clearly visible that there are a few distinct upload capacities that are often used, while intermediate capacities are rarely used. The number of upload slots used in BitTorrent is related to the physical upload capacity by default (see Section 4.2), which implies that in BitTorrent networks, there is a relatively small number of ranks that are shared by many agents. Because agents barter with agents from their own rank before resorting to bartering with lower-ranked agents, in such networks, stratification is bound to occur. The network used by Legout et al. [25] is such a network, where each of the 30 agents has one out of three possible ranks. This motivates the following claim:

Claim 4.8. *Stratification emerges in general in BitTorrent networks.*

Just as with stratification, proving segmentation in general is more difficult. Figure 4.7 shows an example of a network without shared ranks. Here, one agent can be the cause of a segmented network because of its choice of upload slots and its place in the ranking.

In a network where agents share a rank, segmentation can occur even in different stable configurations of the same network. This means that it depends on which configuration is reached by the distributed implementation whether the network is segmented or not. See Figure 4.8 for an example where all agents employ 3 upload slots. Segmentation into larger segments, as in Figure 4.7, is what has been observed in practice by Bharambe et al. [8] and Legout et al. [25], as we described in Section 4.1.1.

Stratification Without Global Ranking

Proposition 4.3 states that when all agents have interacted with each other, their rankings of the other agents are equal to the global ranking. In BitTorrent, agents do not maintain a ranking. Instead, they optimistically unchoke other agents, and barter with the agents that upload to them fastest in the previous round. Bartering relations between two agents *i* and *j* are sustained as long as both

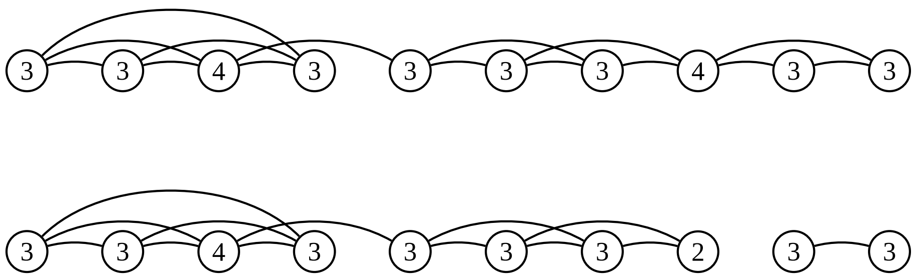


Figure 4.7: *Network of agents that all have a different rank. Numbers denote the number of upload slots an agent deploys. Segmentation does not occur in the upper figure, where the third agent from the right employs 4 upload slots. In the lower figure, this agent employs 2 upload slots, which causes the network to segment. If this agent was ranked one place lower, the network would not have segmented.*

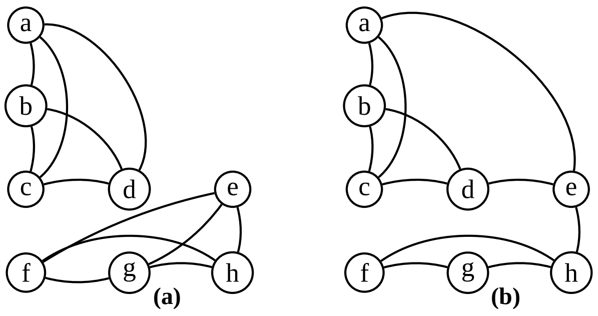


Figure 4.8: *Two stable configurations of the same network in which agents employ 3 upload slots. Segmentation occurs in Figure 4.8a, but not in Figure 4.8b.*

agents reciprocate, but as soon as either encounters a third agent k that reciprocates faster, its slowest bartering relationship is broken for relationship with k.

When the file-sharing game starts, every agent i unchokes k_i randomly selected peers. The resulting configuration is a *random configuration*. In this random configuration, some agents i will be in k_i relationships because they offer good slot capacity, and some agents i will be in less than k_i relationships because some or all of the agents they unchoked encountered better bartering partners. As time progresses, agents unchoke other agents, which enables them to engage in more bartering relationships, or, if they are already in k_i relationships, to improve their bartering relationships by playing better or best responses. Stable relationships are kept, and unstable relationships are replaced by new, possibly stable, relationships. We say that an unstable configuration progresses towards the stable configuration, as we already showed with Lemma 4.5. The following lemma proves that this holds when agents do not maintain a global ranking:

Lemma 4.9. *An unstable configuration progresses towards a stable configuration in the limit, even if agents do not maintain a global ranking.*

Proof. The proof for this lemma is the same as the proof for Lemma 4.5, except that agents do not maintain a global ranking but instead unchoke other agents to discover their slot capacities. This allows them to break relationships with agents with lower slot capacity for relationships with agents with higher slot capacity. In the limit, every agent has repeatedly unchoked every other agent in the network so a stable configuration is reached. □

Claim 4.10. *Stratification occurs in the limit even if agents do not maintain a global ranking.*

Proof. This follows directly from Lemma 4.9 and Claim 4.8. □

Claim 4.10 shows that although our assumption in the previous sections that agents maintain a global ranking does not reflect reality, stratification occurs in general in real BitTorrent networks, given enough time. However, there are two requirements for the corollary to hold: first, the agents must know all other agents in the network, and second, there must be enough time for the agents to interact with all those agents in the network before the download is completed. We address these requirements below.

We start with the first requirement. Agents receive from the tracker a random selection of all peers in the network (see Section 3.1.1). Periodically, this list is updated when the tracker sends a new random selection. Because all selections are random, it is safe to assume that these selections are representative of the whole agent population. Whether an agent i can find relationships with agents that are ranked equal to agents they would be in stable relationships with had they known the global ranking largely depends on the size of the network, the distribution of slot capacities over the agents, and the number of upload slots agent i employs. If i cannot find such relationships, some relationships may be with better-ranked agents and others with lower-ranked agents. This makes it difficult to predict how utility differs for any agent when agents do not know all agents in the network.

The second requirement is that agents need to interact with all other agents in the network. When the swarm consists of many agents compared to the file exchanged, this will not happen. As with the first requirement, because agents make random connections to random selections of agents, the net effect of this on an agent's utility is difficult to predict.

Stratification in the Dynamic Model

At the end of the previous section, we already mentioned that in BitTorrent, an agent's remote peer list is periodically updated. This is a first step of considering stratification in a dynamic network. Until now, we assumed that actions are fixed, i.e., that agents specify an upload capacity and number of upload slots, and do not change these settings or go online or offline. That assumption is unrealistic in practice.

Lemma 4.5 shows that an unstable configuration develops into a stable configuration when actions are fixed and agents know the slot capacities of the other agents. Agents discover these slot capacities iteratively. In a period between two performed actions, this allows the current configuration of the network to become more stable. Whenever an action is performed, the network can become less stable, but after that agents can establish new relationships over unused slots or improve on unstable relationships. The network keeps on progressing towards a stable configuration, but because actions are performed continuously, that configuration will never be reached.

Theorem 4.11. *In a dynamic network, the network configuration progresses towards a stable configuration in the periods where agents do not perform actions.*

Proof. Assume that the network is dynamic but agents do not perform actions. Any agent i unchokes other agents, which enables i to improve on unstable relationships with a better or best response, or establish a relationship where there was none. The former is a progression because an unstable relationship is replaced by a stable or otherwise closer to stable relationship; the latter is a progression because there is a relationship where there was none, which is closer to the stable configuration in which no additional relationships can be made. □

Comparison With Stable Matching

In the above, we have shown that stratification emerges from the client utility function. Gai et al. [20] investigate stratification from a similar point of view: they model BitTorrent as a *Stable Matching* problem, where agents give a preference over the other agents and the question is whether a configuration exists where agents are matched to other agents such that neither wants to change the matching. This is similar to our concept of a stable relationship. Agents are ranked on their slot capacity, and the ranking contains no ties. Based on that, they prove that a unique stable configuration exists, and prove that stratification occurs in this configuration when all agents use the same number of upload slots. As with our results, they are not able to prove stratification in the general case, but instead make it plausible.

There are two main differences between the work by Gai et al. and ours. First, we allow the ranking to contain ties. As a result, a network can have multiple stable configurations. Additionally, we show that not all stable configurations are Pareto-efficient. If a non-Pareto-efficient stable configuration is reached, we show that a Pareto-efficient one is never reached. This means that the outcome in BitTorrent can be suboptimal.

Second, the work by Gai et al. relies heavily on the ranking, even though agents in BitTorrent do not employ such a ranking but decide on who to barter with based only on perceived transfer speed during the last round. In Section 4.1.2, we prove that the results are equivalent to those when agents do maintain a ranking.

4.1.3 Other Observations Explained From The Model

In Section 4.1.1 we discussed Legout et al. [25] which describes stratification from observations of individual BitTorrent clients on a closed network. In their experiment, a file is exchanged in a network of 40 leechers and 1 seeder. The leechers are divided into three groups: agents 1 to 13 have an upload limit of 20 kB/s, agents 14 to 27 have an upload limit of 50 kB/s, and agents 28 to 40 have have an upload limit of 200 kB/s. The seeder is agent 41. The agents exchange a 113 MB file that consists of 453 pieces of 256 kB each. Every agent sets its number of upload slots to 4.[2] With this setup, the experiment was repeated an unspecified number of times, and the results presented in the article are averaged over all runs. In this section, we explain these results from our model.

Cluster formation

Figure 4.9, taken from the article, shows the amount of data exchanged between the agents through regular unchokes (i.e., data sent through bartering and not through optimistic unchoking). Clustering is easily recognizable in this figure from the three dark squares. Note that the total number of unchokes is shown, which results in lighter cells for the faster peers because they finish downloading the file faster and in this simulation, peers leave the system as soon as download completes.[3] In the following, we explain Figure 4.9 from our model.

[2]The article is not clear about whether this includes the optimistic unchoke slot. Without loss of generality we assume that the four slots are for regular bartering, and that agents have one additional slot solely dedicated to optimistic unchoking.

[3]Agent 27 is clearly an outlier. According to the authors, this agent suffered from a bad network connection. As a result, it did not finish downloading with the rest of the agents of its group, and was forced to obtain its remaining pieces from the slowest agents.

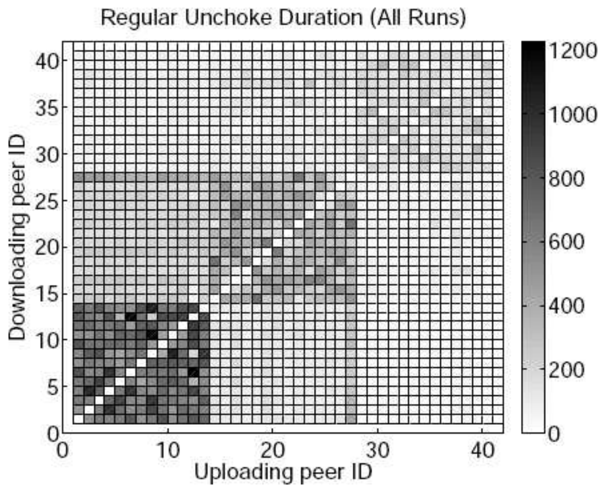


Figure 4.9: *Time duration in seconds that agents unchoked each other via a regular unchoke, averaged over all runs (from Legout et al. [25]).*

The network consists of three classes of agents, and therefore agents have one of three ranks. Every rank is shared among thirteen agents. From our model, we would expect that the majority of regular unchokes is between similar-ranked agents. This is indeed the case, as is clearly visible in the figure from the three dark squares.

Additionally, two faint squares are visible above the two lower squares. This means that the slower agents of ranks 1 and 2 unchoke agents of ranks 2 and 3, respectively. The absence of similar squares to the right of the lower squares means that these unchokes are not reciprocated by the faster agents. This is expected from our model: the faster peers are attractive to the slower peers, but the reverse is not true. The slower peers try to form a relationship with the faster peers, but because the faster peers do not reciprocate, the slower peers lose interest and stick to agents of their own rank.

Bartering relationships

From our model, we expect that the agents form bartering relationships. This would result in a figure that is symmetric around the 45 degree line because that would mean that an agent *a* reciprocates to *b* what it receives from *b*. Indeed, we find that the figure is largely symmetric.

Stable Relationships

Not only does our model predict the formation of bartering relationships, but it also predicts that the agents establish stable relationships given enough time, as described in Section 4.1.2. If every peer uses four upload slots, every peer would be in four stable relationships in every run of the experiment. Because Figure 4.9 averages the results over all runs, the existence of stable relationships can no longer be deduced.

However, we do know that with unchoking, agents are randomly selected. Because the experiment is performed multiple times, if stable relationships were not formed, we would expect the unchokes to be more evenly spread within each class. The fact that this does not happen implies that stable relationships are indeed formed.

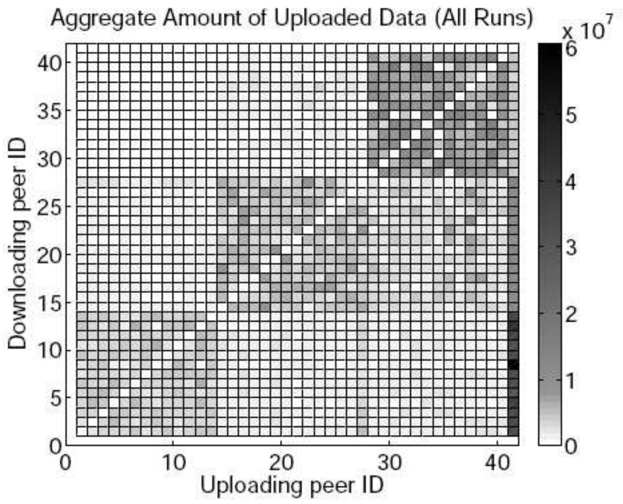


Figure 4.10: *Total number of bytes uploaded by agents to each other, averaged over all runs (from Legout et al. [25]).*

Upload/download ratio

Bharambe et al. [8] find in measurements on simulated BitTorrent swarms that the faster peers in a swarm upload significantly more data than the slower peers do. This simulated result is backed by tracker logs for real torrents. Legout et al. [25] observe the same in their simulation. Bharambe et al. argue that this results in unfairness in terms of the volume of data served, while according to Legout et al., the difference is not unfair as long as the faster peers are able to finish their downloads sooner than the others.

Figure 4.10 shows the data volumes in the experiment by Legout et al. Because the results are again averaged over all runs of the experiment, it is difficult to quantify the exact data volume for any peer from the legend. However, it is clear that the faster peers upload a substantially larger volume of data than the slower peers. We now explain this using our model.

Assume that the network in the experiment has formed a stable configuration. Agents have four bartering slots and one optimistic unchoke slot. Then, every agent is in four bartering relationships with agents of its own rank, and downloads the exact same volume that it uploads by bartering. This excludes the data sent over the optimistic unchoke slot to optimistically unchoked peers. As these are randomly selected and the three classes in the experiments are of equal size, each class receives on average 1/3 of any agent's data sent through optimistic unchokes. With the upload ratio of fast : medium : slow agents in the experiment set to 200 : 50 : 20 kB/s, every agent receives on average $(200+5+2)/3 = 90$ kB/s through optimistic unchokes. Every second a fast peer uploads 200 kB optimistically unchoking other peers, and receives only 90 kB from being unchoked. If a fast peer uploads 5 pieces, it downloads 4 + 90/200 pieces, which results in an upload/download ratio of 1.12. In other words, a fast peer uploads 112% of the data it downloads.

Similarly, we find that a medium peer has an upload/download ratio of 0.86, and a slow peer of 0.59. Note that the slowest peers upload slightly more than half of what a fastest peer uploads. This complies with the observations of Bharambe et al. and Legout et al. and provides an agent with an incentive to specify a low slot capacity if it values low upload volume over short download times.

In the above, we assumed that the network configuration is stable. If it is not, this results in an even higher upload/download ratio for the faster agents. This is because in an unstable configura-

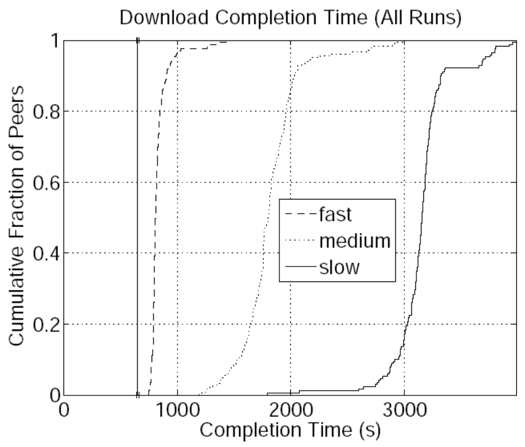


Figure 4.11: *Cumulative distribution of the download completion time for the three different classes of leechers (from Legout et al. [25]).*

tion, a faster agent is connected to some slower agents, which means that it cannot receive a 1:1 upload/download ratio over its regular upload slots. Similarly, an unstable configuration results in an even lower upload/download ratio for the slower agents.

From the above, we conclude that it follows naturally from our model that BitTorrent is unfair with regard to data volume upload. This unfairness is a problem because it increases the download time for the faster agents, as we show below in Section 4.1.3. The unfairness is caused by the optimistic unchoking policy which randomly selects agents to unchoke. Bharambe et al. [8] suggest that this issue can be addressed if the tracker sends remote peer lists filled with peers of similar ranking to the requesting peer, instead of a random selection of all peers. However, they argue that this may cause segmentation. Note that this would be beneficial to the faster agents only, because the slower agents will then no longer receive a significant part of their pieces from the fast agents. The implications of this on the network need investigating.

Based on the above, we propose a novel die-hard manipulation strategy that is effective for the faster agents in the network, i.e., all agents with an upload/download ratio larger than 1. If, at some point, such an agent is in bartering relationships with similar-ranked agents over all its regular upload slots, and encounters another similar-ranked agent through optimistic unchoking,[4] it can decide to stop optimistic unchoking, and instead use the optimistic unchoke slot for regular bartering with the newly discovered agent. This will lower its upload/download ratio towards 1, thereby decreasing its download time.

Download Completion Time

In Section 4.1.2, we assume that agents strive to minimize their download times. In their experiment described earlier in this section, Legout et al. [25] measure the download completion time for the different agent classes. Their findings are presented in Figure 4.11.

In the figure, the vertical line represents the optimal completion time, which is the time the seed finished uploading a complete copy of the file. According to the authors, that time is around 650 seconds on average. However, we remark here that with a file size of 113 MB and an upload capacity of 200 kB/s for the fastest peers, it should theoretically be possible to download the file in 565 seconds, provided that the network is in stable configuration and the file is well distributed over

[4]A heuristic similar to the well-known Secretary Problem could be used to select suitable agents.

the other fast peers. The figure of 565 seconds includes optimistic unchokes, so would require that the fast agents are unchoked by fast agents only, which makes this a theoretical lower bound only.

It is clearly visible in the figure that the fast peers complete their download well before the medium and slow peers. Legout et al. conclude that BitTorrent's choking algorithm fosters reciprocation by rewarding contributing peers, and provides agents with efficient sharing incentives. In the following, however, we put some remarks to this conclusion.

If we look at Figure 4.11 in more detail, we find that the majority of the fast agents complete their download in about 800 seconds. Compared to the optimal completion time of 650 seconds, this is a 23% increase, and compared to the theoretical lower bound of 565 seconds, it is a 42% increase.

We now consider the slow agents in Figure 4.11. About 90% of these agents complete their download at about 3300 seconds. If these agents would download the file completely through bartering in a stable configuration, this would require them 113 MB / 20 kB/s = 5650 seconds. Even though the slow agents finish their download last, their effective download time is 58% of the theoretical download time.

We now take a different perspective on the same issue, using our model and the results from Section 4.1.3 in which we showed that faster peers have a larger upload/download ratio. Consider a file consisting of 1,000 pieces exchanged by the same agents from the experiment by Legout et al. The fast peers, with an upload/download ratio of 1.12, need to upload 1,120 pieces to download all 1,000, which accounts for a 12% longer download time compared to a bartering-only download. The slow peers, with their upload/download ratio of 0.59, need to upload only 590 pieces, yielding an effective download time of 59% of the time needed for a bartering-only download.[5] Note how these figures comply with the theoretical figures derived above. As we mentioned in Section 4.1.3, the differences become even larger if the network is not in a stable configuration.

In conclusion, we find that from current BitTorrent, some form of egalitarianism emerges which supports the slow agents at the cost of the fast agents in the network. This provides calculating users with an incentive to tweak their settings such that their download time decreases.

4.1.4 Conclusions

We started this section with a review of the literature that observes stratification and segmentation in real BitTorrent swarms. There, it is proposed that the client utility function (see Section 4.1.2) is the cause of this. In Section 4.1.2, we prove that indeed both stratification and segmentation follow naturally from our model.

We prove that over time, a static BitTorrent network reaches a stable configuration in which no agent can barter with higher ranked agents, and no additional bartering relationships can be made. Where related work shows such a stable configuration to be unique, based on our ranking which allows ranks to be shared by multiple agents, we prove that this is not the case. Multiple stable configurations may exist, and some are Pareto dominated by others. We also prove that if a stable configuration is not Pareto efficient, a Pareto efficient configuration is never reached. This result seems quite dramatic, as it suggests that the outcome in BitTorrent is often suboptimal. However, it relies on the assumption of a static network. In a dynamic network, the network configuration does progress towards a stable configuration (which may or may not be Pareto efficient), but because agents perform actions continuously, this configuration is never reached.

Similar to related literature, we are not able to formally prove that stratification emerges in general in BitTorrent networks, but we do make a reasonable case with Claim 4.8. Our reasoning in this section is facilitated by the concept of a global ranking. Unlike related literature, we show that our claims hold even without this concept.

[5]We assume here that agents leave the network after download completes, and are replaced by agents of the same class.

In addition to proving stratification, our model allows us to prove that BitTorrent is unfair with regard to data volume upload, as has been observed in related literature. This unfairness supports the slow agents in the network at the cost of the faster agents. Based on this result, we propose a novel die-hard manipulation strategy that is effective for the faster agents in the network.

4.2 Equilibria

In Section 4.1, we showed that in a file-sharing network in which every agent i reciprocates to its k_i fastest uploaders, a stable network configuration emerges. In this configuration, agents are grouped on their slot capacities and barter only with agents with similar slot capacity. In this section, we investigate whether this resulting configuration provides the agents with an incentive to assign their full physical upload capacity to the file-sharing network.

4.2.1 Different Classes of Lazy Free-Riders

In Section 3.3 we introduced lazy free-riding agents as agents that use a client that conforms to the network's protocol. Users use the options provided by that client's interface strategically to maximize their utility. In this section, we consider three different classes of lazy free-riding users: average users, advanced users, and optimizing users.

Average Users

An average user is a user that does not have the skill, knowledge, or motivation to adjust settings in the client interface. This includes specifying the number of upload slots used. With most clients, it is possible to adjust the upload capacity in the main window of the interface. Therefore, we assume that an average user can change this capacity, even though some will not be aware of how it influences its download rate. By default, clients dedicate full upload capacity to the network. We now investigate whether this default setting maximizes an average user's utility.

Denote by d_i agent i's download rate. We define an average user's utility function to be:

$$u_{\text{average,i}} = d_i \tag{4.1}$$

In words: the higher i's download rate, the higher its utility. Pieces can be downloaded by bartering, or by being optimistically unchoked.

Theorem 4.12. *In a stable configuration of a network in which every agent has Equation 4.1 as its utility function, and where the only action available to the agents is specifying the upload capacity, the strategy profile where every user specifies the full upload capacity is a dominant strategies equilibrium.*

Proof. Assume that the network configuration is stable (see Definition 4.6). Then, by Claim 4.8, an agent barters with agents with similar upload capacity exclusively. Therefore, any agent i's utility is maximized by specifying full upload capacity, regardless of the slot capacities of the other agents. Another action x_i' can either allow i to barter with the same agents, yielding equal utility, or forces i to barter with agents with lower slot capacity, yielding lower utility. According to Definition 3.7, this is a dominant strategies equilibrium. □

A result similar to Theorem 4.12 is given by Qiu and Srikant [39], Gai et al. [20], and Fan et al. [14], amongst others. There BitTorrent's outcome is found to be a Nash equilibrium. There are two differences between those results and ours. First, where the others consider the client utility function only (see Section 4.1.2), we recognize that the user's utility function does not need to match that client utility function. In the following sections, we consider other types of users and show that this is an important distinction. To our knowledge, we are the first to make this distinction. Second, they claim the equilibrium to be a Nash equilibrium (see Definition 3.4), whereas we show it to be a dominant strategies equilibrium (Definition 3.7).

Manipulation Possibilities for Average Users

The proof for Theorem 4.12 already shows that in the dominant strategies equilibrium, there is some room for an agent to manipulate: if it decreases its upload capacity with an amount small enough to stay connected to the same bartering partners, its utility will not decrease. This equilibrium is a weak equilibrium, as defined in Section 3.2.2. With Equation 4.1 as utility function, the agent's utility will not increase either, so the user has no incentive to do so. For such an incentive, the utility function needs to incorporate an upload component. Denote by r_i agent i's upload rate. Then, such a utility function is:

$$u_{\text{average},i} = \alpha d_i - \beta r_i, \alpha, \beta \in \mathbb{R}^+ \tag{4.2}$$

where α and β are some arbitrary positive constants. We make the following observation for a network where all users have this utility function:

Observation 4.13. *In a stable configuration of a network in which every agent has Equation 4.2 as its utility function, and where the only action available to the agents is specifying the upload capacity, the dominant strategies equilibrium from Theorem 4.12 is not reached.*

We motivate this with an example. Assume an agent i with Equation 4.2 as its utility function, and common prior P over the types of the other agents. This distribution includes the distribution of upload bandwidths over the other agents. Assume for simplicity that that distribution is uniform over $[0, M]$, where M is some maximum upload capacity. P also contains the probability for an agent to be interested in the content i is interested in, or, in other words, P allows i to make an estimate on the swarm size for the file it wants to download. Based on this information, i can calculate its expected utility $u_i = E_P[u_i(\theta_i, x_i, s_{-i}(\theta_{-i}))]$ because, on average, it will be bartering with the k_i agents whose slot capacities are closest to i's slot capacity. If i lowers its upload capacity (i plays x_i' instead of x_i), its slot capacity drops proportionally, thereby decreasing the term βr_i. As long as i's slot capacity is larger than that of the first agent ranked below it, i barters with the same agents as before, and the term αd_i is unchanged. Therefore, its total utility increases. If we denote by $\delta = \frac{c_{u,i}}{k_i} - (\frac{c_{u,i}}{k_i})'$, then with the uniform distribution over the upload capacities i can lower its slot capacity with $\delta = \frac{M}{n}$ to obtain its maximized utility with minimal uploaded data.

We are not aware of any literature in which such behavior is described. Intuitively, this behavior seems unlikely, and we give four possible explanations for this:

1. Equation 4.2 is an unrealistic utility function, or in practice, β is so small compared to α that the actual increase in utility from manipulation is marginal;

2. If uploaded volume were part of the utility function, then expected utility would increase with this form of manipulation. However, actual utility may differ from expected utility, which renders this form of manipulation unattractive to risk-averse users. The user is certain that utility increases only when the manipulation is carried out repeatedly. For a user that downloads sporadically, the uncertainty of whether utility will actually increase may provide an incentive to not manipulate. This motivates looking into a repeated equilibrium as found in the Repeated Prisoners Dilemma [6]. Then, specifying all upload capacity can motivate others to do the same as they will benefit from that in future downloads. This, however, seems unlikely in our situation;

Upload capacity (kB/s)	# Upload slots	Slot capacity (kB/s)
16	3	5.3
32	3	10.7
64	5	12.8
96	6	16
128	8	16
256	10	25.6

Table 4.1: *Upload capacity and default number of upload slots in the Vuze BitTorrent client, taken from [3]. Resulting slot capacity is also listed.*

3. There is another reason for expected utility to differ from actual utility, which is explained from the *bounded rationality* of the users. By that, we mean that users have limits to the information they can have, cognitive limitations of their minds, and a finite amount of time to make decisions, especially when, as in our setting, the common prior P is enormous. The expected utility can be based on wrong information, wrong beliefs, or wrong calculations. This especially holds in the case of the average user, that may very well not be aware of the connection between upload capacity and download rate;

4. Manipulation as we described above requires effort: information on bandwidth distributions must be gathered, estimates must be made, and settings must be adjusted. Such costs should be included into the utility function, because too much effort outweighs the benefits of manipulation. This makes manipulation less attractive;

Advanced Users

An advanced user is a user that is willing to change any of the settings we listed in Section 3.2.2. In particular, this means that an advanced user can tweak the maximum upload capacity and number of upload slots. Just as an average user, an advanced user strives to minimize its download time, or to maximize download rate, and therefore it has Equation 4.1 as its utility function.

Because an advanced user can change its number of upload slots, it is more flexible in determining which, and how many, agents it barters with than an average user. We illustrate this with an example. Consider a network in which every agent has a slot capacity of 25 kB/s. Agent i enters this network with a total upload capacity of 100 kB/s. If i employs three upload slots, it will receive 75 kB/s through bartering, whereas it receives 100 kB/s with four upload slots.

In Section 4.1.2, we presented Figure 4.6, which shows the distribution of upload capacities measured in the Gnutella file-sharing network in 2002. It is clearly visible that there are a few distinct upload capacities that are often used, while intermediate capacities are rarely used. Combined with our results in Section 4.1, large layers of users with the same slot capacity are expected, if we assume that all users use the same number of upload slots. However, the number of upload slots that a client (not the user) specifies by default increases with the total upload capacity. Table 4.1 lists these settings for the Vuze BitTorrent client.[6] Because the resulting slot capacities are different, segments are still expected.

In Section 4.1.1, following Legout et al. [25], we discussed networks in which peers have one of three different upload capacities. In order to demonstrate strategies available to advanced users, we now consider a similar network, in which agents have an upload capacity of 64, 128 or 256 kB/s. We assume that the agents follow the guidelines of the Vuze client, resulting in the agents employing 5, 8 or 10 upload slots, respectively, with a respective slot capacity of 12.8, 16 and 25.6 kB/s. As a

[6]Vuze (`http://www.vuze.com`) is among the most popular BitTorrent clients today.

result, the network will be grouped into three layers. In large networks, it is safe to assume that there are sufficient agents in every layer for a new agent to barter with.

Consider an agent i that joins the swarm. Assume $c_{u,i} = 256$ kB/s, and $k_i = 10$, i.e., the default value. Then, i barters with 10 other agents with the same upload capacity, and $u_i = d_i = 10 \cdot 25.6 = 256$. If i specifies $k'_i < 10$, its slot capacity rises, but it still connects to the agents with slot capacity 25.6 kB/s because no agents are ranked higher, and its utility does not increase. If $k'_i > 10$, i's slot capacity decreases. Take $k'_i = 15$ as an example, yielding a slot capacity of 17 kB/s, just above the slot capacity in the middle layer. i then connects to the agents in the middle layer, and $u'_i = 15 \cdot 16 = 240 < u_i$. In general, if $\frac{c_{u,i}}{k'_i}$ equals the slot capacity in one of the layers, then $u'_i = u_i$; otherwise, $u'_i < u_i$. If $c'_{u,i} < c_{u,i}$, then i connects to lower-ranked agents and its utility decreases. This example motivates Theorem 4.14 below. In order to prove that theorem, first we define what we mean by a large network.

Definition 4.9. *A* large network *is a network consisting of so many agents that an agent can expect to be bartering with agents from one rank only.*

Theorem 4.14. *In a stable configuration of a large network in which every agent has Equation 4.1 as its utility function, and agents can specify their upload capacity and number of upload slots, specifying full upload capacity and default number of upload slots is a Bayesian-Nash equilibrium.*

Proof. Assume a stable network in which the conditions mentioned hold. Then, by Definition 4.9 and Claim 4.8, an agent i expects a finite number of layers, each of which is shared by multiple agents. Denote by u^*_i the utility i obtains if it specifies full upload capacity and default number of upload slots k^*_i.

If an agent i specifies upload capacity $c'_{u,i}$ other than full capacity $c_{u,i}$, it will barter either with the same, or with lower-ranked agents, because $c'_{u,i} < c_{u,i}$ because of physical constraints. This yields a utility equal to or lower than u^*_i.

If i specifies upload capacity $c'_{u,i} \leq c_{u,i}$ and $k'_i \neq k^*_i$ upload slots, then two disjoint cases are possible:

Case 1: If there is a layer in which the upload capacity equals $\frac{c'_{u,i}}{k'_i}$, then i's expected utility is exactly $c'_{u,i} \leq u^*_i$;

Case 2: If there is no such layer, then i's slot capacity is larger than that of the agents it barters with, and therefore i's expected utility is smaller than u^*_i.

Therefore, with any combination of $c'_{u,i}$ and k'_i, i's expected utility is smaller than, or equal to, u^*_i. For any agent $i \in N$, strategy s_i that results in i specifying $c_{u,i}$ and k_i provides i with highest expected utility. By Definition 3.6, this is a Bayesian-Nash equilibrium. $\square$

Theorem 4.14 does not hold if an agent cannot expect to barter with equally-ranked agents exclusively, as the following theorem shows.

Theorem 4.15. *In a stable configuration of a smaller network in which every agent has Equation 4.1 as its utility function, and agents can specify their upload capacity and number of upload slots, an equilibrium is not reached.*

Proof. We use proof by example.

Consider a network consisting of an agent i and 11 other agents. Agent i has a physical upload capacity of 200 kB/s, which it divides evenly over its k_i upload slots. Agents 1 through 11 have a slot capacity of $5, 10, 20, 30, \ldots, 100$ kB/s. We expect that in a stable configuration of such a small swarm, an agent is connected to the agents with slot capacity centered around its own slot capacity. Table 4.2 shows the outcome, and i's utility. We find that here, it is beneficial for i to choose k'_i strategically. Maximum expected download rate is obtained with 6 upload slots; however, this is

Upload slots	Slot capacity	Barters with	Download rate
1	200	100	100
2	100	90, 100	190
3	67	50, 60, 70	180
4	50	30, 40, 50, 60	180
5	40	20, 30, 40, 50, 60	200
6	33	10, 20, 30, 40, 50, 60	210
7	29	5, 10, 20, 30, 40, 50	155

Table 4.2: *Agent i has physical upload capacity of 200 kB/s, and joins a network of 11 other agents that have a slot capacity of* $5, 10, 20, 30, \ldots, 100$ *kB/s, respectively. For every number of upload slots* k_i*, we specify i's slot capacity, the slot capacities of the agents that i connects to, and resulting expected utility.*

based on the assumption that when 6 slots are employed, *i* will connect to the six agents with slot capacity centered around its own slot capacity, i.e., agents with slot capacities 10 through 60. If the agent with slot capacity 60 has all its upload slots assigned to other agents, *i* is forced to barter with the agent with slot capacity 5 instead, which would result in $u_i = 155$. Using 2 upload slots is a safer choice, as that results in a marginally smaller download rate of 190, but with high probability because *i* then has a slot capacity of 100 kB/s which makes it the best-ranked agent. □

From the proof for Theorem 4.15 we find that the utility obtained by the user can vary substantially: from a download rate of 100 with one upload slot, to 210 with six upload slots. However, the gain fully depends on the number of agents in the swarm and the distribution of bandwidth over these agents, both of which are difficult to predict before joining the swarm. In general, the smaller the swarm, the more skewed the bandwidths are distributed over the agents, and the more gain can be obtained by selecting an optimal number of upload slots. But, as the proof above shows, specifying another number of upload slots may result in very low utility. Joining a smaller swarm with any number of upload slots holds the risk that this number yields suboptimal utility. The larger the swarm, the smaller this risk, and according to Theorem 4.14, it is no longer a risk when the network is large, according to Definition 4.9. For the smaller swarm, a user may be best of by joining the swarm first with the default number of upload slots, observing the upload rates of the other agents in the swarm, and then calculating the optimal number of upload slots. However, this will be too much of an effort for most users, especially considering that agents join and leave the network continuously, which means that the calculations have to be repeated multiple times. In Chapter 5, we elaborate on this issue and suggest a BitTorrent client that dynamically performs such calculations.

4.2.2 Long-Term Seeding Incentive

With the utility for both average and advanced users directly tied to the download rate, it follows directly that neither of these users obtain any utility when the download is finished. Figure 3.2 shows two examples of utility where we assume that full utility is obtained when the download finishes. Another option is Figure 4.12, where utility accumulates until the download completes and full utility is reached.

From all these examples, we find that BitTorrent offers no long-term seeding incentive to average and advanced users. When their downloads complete, these users obtain no utility from seeding the downloaded file; instead any user that has a utility function with a negative upload component, such as Equation 4.2, has an incentive to go offline directly after the download finishes.

In the following section, we discuss the BarterCast reputation mechanism that has been added to the Tribler client to provide users with a long-term seeding incentive.

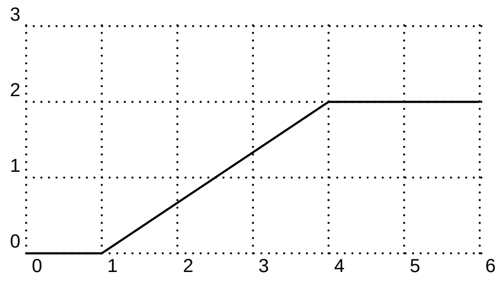


Figure 4.12: *Download starts at* $t = 1$ *and finishes at* $t = 4$*. After* $t = 4$*, no data is downloaded, so no additional utility is obtained.*

4.2.3 Conclusions

We started this section by defining two different classes of lazy free-riders: average users, and advanced users. We show that a user has room for manipulation if its utility function incorporates an upload component. However, in most practical situations, we find that this room is small and in practice it is not used.

Related literature has found that if specifying upload capacity is the only action available to users, specifying full upload capacity is a Bayesian-Nash equilibrium. Instead, we find that it is a dominant strategies equilibrium, which is a stronger equilibrium as it does not depend on the assumption of a common prior that is shared amongst all agents in the network.

If agents can specify their number of upload slots in addition to specifying their upload capacity, we show that specifying full upload capacity and the default number of upload slots is a Bayesian-Nash equilibrium. However, this does require that the network is large, because it depends on whether agents can expect to be bartering with equally ranked agents exclusively. In smaller networks, where that expectation is not met, we show that an equilibrium is not reached and that agents can gain or lose substantial amounts of utility, dependent on their settings. Unfortunately, it is difficult for an agent to determine which settings will provide maximum utility, and for most users, the effort of performing these calculations will outweigh the possible gain in utility by deviating from the default settings.

Finally, and unsurprisingly, we show that current BitTorrent does not provide an incentive for users to stay online after their download has finished. An additional mechanism is needed to provide this incentive, and this is the subject of the next section.

4.3 BarterCast

Our results in the previous section show that BitTorrent needs additional mechanisms to provide users with a long-term seeding incentive. One such mechanism is the BarterCast reputation mechanism. In this section, we investigate whether a user has an incentive to use BarterCast in a network where all agents use BarterCast.

We introduce and describe BarterCast in Section 4.3.1. In Section 4.3.2, we investigate whether using BarterCast is an equilibrium strategy. In Section 4.3.3, we draw conclusions. We verify the theoretical results from this chapter with experiments in the following chapter.

4.3.1 Introduction

In Section 4.2.2, we showed that even though BitTorrent provides incentives for users to upload while downloading, there is no incentive for users to stay online and seed the file after their download has completed. BitTorrent relies on altruistic peers to keep content available, which is problematic especially for content in which few peers are interested.

In Section 2.1.1 we gave an overview of proposed solutions to this problem, which are all based on reputation systems that keep track of an agent's contribution to the network with respect to its consumption. Centralized reputation systems exist in the form of private communities that ban peers who upload little compared to what they download. A study by Andrade et al. [5] finds that in such communities, more peers share their content than in public communities, which results in higher download performance and more available content. The downside of such communities is that they have a central point of failure, considerable administration and management overhead, and require user's trust in an unknown authority with respect to privacy sensitive information [28]. As an alternative, distributed reputation systems have been proposed, but none of these have been successfully deployed in practice because research has focused on systems that cannot be manipulated, thereby sacrificing practical feasibility for attractive theoretical properties.

Taking another approach, Meulpolder et al. [28] introduce BarterCast. Here, attractive theoretical properties may be sacrificed for practical feasibility. BarterCast is a fully distributed reputation mechanism that is deployed in practice in the Tribler network. This is possible because it does not compute a globally consistent reputation score for any agent. Instead, reputation in BarterCast is modeled after reputation in human communities, where every person has a subjective reputation for every other person in the community, that is based on direct experience and information obtained from other people. Because it is based on incomplete information, such reputation is not globally consistent.

In BarterCast, the data volume that an agent has received from another agent in the past is taken as that agent's direct experience. Every agent i maintains a private history as a set of triples *(j, up, down)* that represents the amount of data measured in kB that i exchanged with j. This private history is shared with all other agents in the network known to i. From the private histories that i receives from the other agents, it constructs a subjective sharing history from which it determines the net contribution of peers that i did not interact with itself. It does this by constructing a graph, where the agents are the vertices. Edges between vertices are directed and have a capacity, where the capacity of the edge between i and j represents the data volume that i has uploaded to j. If i evaluates an agent j that it had no direct experience with, it calculates the maximum flow over the graph from j to i, using the Ford-Fulkerson algorithm (see Meulpolder et al. [28] for details). This way, j's contribution is bound by the reputation of the agents that reported j's contribution to i. Note that an agent a may exaggerate another agent b's reputation to an agent c, but this has limited effect because c bounds b's reputation by its direct experience with a.

The sharing history is private to i and not shared with other agents. Because i is not necessarily informed of all j's up- and downloads, the net contribution $b_i(j)$ of j according to i is subjective. From $b_i(j)$, the subjective reputation value $R_i(j)$ of agent j according to i is calculated as follows:

$$R_i(j) = \frac{\arctan(b_i(j))}{\pi/2} \tag{4.3}$$

The arctan-function is used for two reasons. First, it limits the resulting reputation between -1 and 1. Second, it has the effect that the difference between, for example, 0 and 100 MB is more significant than the difference between 1000 and 1100 MB. This ensures that a modest contribution of a new or neutral peer significantly effects its reputation.

Agents use their subjective reputations of the other agents when they unchoke other agents. Meulpolder et al. [28] describe how this reputation can be used by agents under two different policies to enforce participation. The first is the *ban-policy*, under which an agent does not upload to agents with a reputation below a certain threshold (the article considers -0.3, -0.5 and -0.7). The second policy is the *rank-policy* under which all agents are served, but those with high reputation first. The idea is that this motivates agents to seed a downloaded file, as that will boost their reputation and make for faster subsequent downloads. Based on simulations and measurements in real swarms, Meulpolder et al. find that BarterCast is effective in distinguishing free-riders from altruists. In the following, we investigate whether individual users receive best performance in a swarm if they enable BarterCast.

4.3.2 Incentive to Use BarterCast

Introduction

In this section, we include BarterCast in the BitTorrent model we developed in Chapter 3.2. In the remainder of this section, by BitTorrent we refer to the BarterCast-enhanced BitTorrent network, while by *plain BitTorrent* we refer to BitTorrent without BarterCast.

The BarterCast reputation mechanism we described in the previous section is fully distributed: every agent calculates its own subjective history and shares its private history with the agents it knows. Because agents are autonomous, it is possible for an agent to send messages that do not conform to the BarterCast specification, or to send false reports. However, as we motivated in Section 3.3, in this thesis our focus is on lazy free-riding agents, i.e., agents that will not develop or install clients with such behavior. These agents use the interface to enable or disable the BarterCast mechanism. If this is enabled, we assume that the BarterCast mechanism is faithfully executed.

With BarterCast added to our network, agents in a file-sharing network with BarterCast have an extra action $x_{i,b} \in X_{i,b}$ available that models the enabling or disabling of the BarterCast mechanism.

Equation 3.5 is the outcome function that takes all agents' actions and from that calculates an outcome that determines which file chunks are exchanged by which agents. In Section 3.2.5, we described that in plain BitTorrent, this outcome function is implemented distributedly by the agents, with every agent uploading file pieces to its best-reciprocating agents. Chapter 4.1 explained how this implementation leads to stratification and identifies equilibria of the agents' strategies.

With BarterCast added to our network, the distributed implementation of the mechanism that determines the outcome changes. Agents that use BarterCast no longer unchoke their best-reciprocating agents, but instead unchoke the agents that best comply with their participation policy.[7] Note that besides the implementation of the mechanism and the set of actions available to an agent, our model remains unchanged.

Preliminaries

In Section 4.3.1, we mention that Meulpolder et al. [28] consider two policies: the ban-policy and the rank-policy. We investigate the incentive for a user to employ BarterCast with either of these two policies in a network where all agents use that policy. Our goal is to show that using BarterCast with one of these policies is an equilibrium strategy, because then BarterCast's intended long-term seeding incentive can be achieved.

BarterCast is developed to provide a long-term seeding incentive, and keeps a reputation score of an agent's participation in multiple swarms. In our analysis, we therefore consider a setting where the agents have been active before in other swarms, in which they built up a reputation score. We focus on one swarm only, and keep any agent's reputation score constant, because the data volume exchanged in this swarm is negligible compared to its previously exchanged volume. We assume that all agents in the swarm use the same policy.

As in Section 4.2, we model the agents to have one of a distinct number of upload capacities. We assign reputation scores to the agents according to some normal distribution, so that there is no correlation between an agent's upload capacity and its reputation score. We address this later in this section.

[7]We assume that there is only one policy, or, equivalently, that agents cannot select the policy used.

Under the ban-policy, agents with a reputation below some threshold are not served, while the other agents are treated as in plain BitTorrent. The effects of this on the outcome are easily predicted: the agents with a too low reputation are not served by the other agents, and can only barter with each other. In a network where all agents, the seeders included, use BarterCast with the ban-policy, those agents will not finish their downloads. For the agents with reputation above the threshold, nothing changes. An agent that switches to plain BitTorrent bartering cannot improve on its completion time: if its reputation is too low, it cannot download from the other agents, while if its reputation is sufficient, it barters as it would with the ban-policy enabled. As a result, using BarterCast with the ban-policy is clearly a Bayesian-Nash equilibrium strategy.

Whether using BarterCast with the rank-policy is an equilibrium strategy as well is more difficult to predict, and we focus on that in the remainder of this section.

Ranking and Stratification

The outcome in plain BitTorrent fully depends on the ranking of slot capacities. In Section 4.1.2, we showed that even though agents do not actually maintain such a ranking of the other agents in the network, the outcome is equivalent to the outcome in a network where agents do have such a rank. We also showed (Section 4.1.2) that this ranking is a global ranking. This global ranking is the main cause of stratification in plain BitTorrent networks.

Agents that use the rank-policy no longer rank agents on their slot capacity, but instead on their reputation score. From this point on, we refer to the ranking on slot capacity as the *capacity ranking*, and to the ranking on reputation score as the *reputation ranking*. Note that with the reputation ranking used with the rank-policy, it is not the downloading agent that ranks its uploaders, but the uploading agent that ranks its downloaders. An uploading agent i selects, from all agents that uploaded to i over the past round, the k_i agents with highest reputation, and reciprocates to those agents. Those agents do the same, and for each of those agents, if i is among its uploaders with highest reputation, it reciprocates to i and a bartering relationship is formed.

For now, we assume that an agent's reputation in BarterCast is global, i.e., $R_i(j) = c$ for all agents $i \in N$, $i \neq j$ and $-1 < c < 1$, and drop this assumption later in this section. Then, all agents rank the other agents using the same metric, and all agents have the same reputation ranking over the other agents. As in plain BitTorrent, this is a global ranking.

Following our reasoning in Section 4.1.2, we expect the network configuration to progress towards a stable configuration. However, in this context, this means that agents with similar reputation scores, and not similar slot capacities, are in bartering relations with each other. The network is stratified on reputation scores.

In plain BitTorrent, the resulting stratified stable configuration allowed us to identify equilibria with regard to upload capacity and number of upload slots used. In the following, we investigate whether the outcome of the BarterCast-enhanced BitTorrent provides users with an incentive to use BarterCast.

Equilibrium

Above, we showed that using the rank-policy in BarterCast-enhanced BitTorent, a stable configuration is reached which is stratified on reputation scores. In other words, all agents are bartering exclusively with agents with similar reputation.

Because we expect no correlation between reputation score and upload capacity (as motivated earlier in this section), in the stable configuration, an agent expects to be bartering with agents randomly drawn from all agent classes present in the swarm. In Sections 4.1.3 and 4.2, we show that this causes unfairness with respect to uploaded data volume and completion times: faster agents upload much more than they download, and could finish their download sooner if they could barter

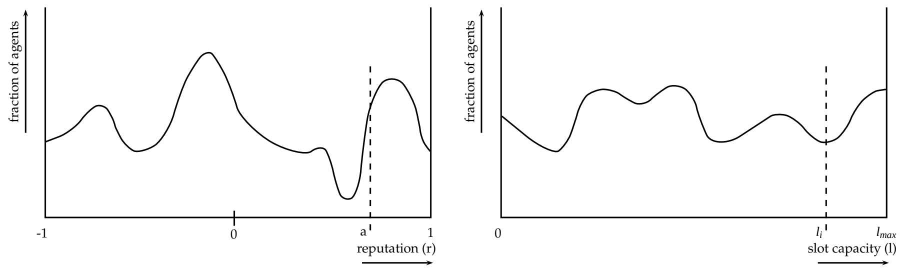


Figure 4.13: *Distribution of the agents over the reputation scores (left) and slot capacities (right).*

with other fast agents exclusively. The reverse holds for the slower agents. BitTorrent's outcome under the rank-policy is therefore beneficial for the slower agents, at the expense of the faster agents. We now investigate whether agents have an incentive to switch from using the ban-policy to plain BitTorrent bartering, in an attempt to improve their completion time.

In a network where all agents use the rank-policy, an agent *i* with high reputation score can barter with any agent *j* it wants, because *j* will always reciprocate. Agent *i* can improve its download rate by turning off its rank policy and switching to plain BitTorrent bartering, thereby favoring agents with high upload capacity. An agent with low reputation, on the other hand, can only barter with other agents with low reputation. In order to improve on its completion time, it needs to find good bartering partners among the other low reputation agents. Whether this is possible depends on the fraction of lower reputation scores in the swarm, and the size of the swarm. The lower the agent's reputation, the more time it needs to explore the network to find suitable bartering candidates.

In order to formally proof this, we first define a large BarterCast network.

Definition 4.10. A large BarterCast network *is a network consisting of so many agents that an agent can expect to be bartering with agents with similar reputation exclusively.*

Theorem 4.16. *In a stable configuration of a large BarterCast network, where:*

1. *all agents use the rank policy,*
2. *the reputation ranking does not fully correspond to the capacity ranking,*

using the rank policy is not a Bayes-Nash equilibrium strategy for an agent with high reputation.

Proof. Assume a stable configuration of a large BarterCast network for which the conditions mentioned hold. Denote by *i* an agent with high reputation. In the stable configuration, *i* barters with agents with similar reputation. Because of condition 2, *i* expects these agents to have different upload capacities. If *i* turns off its rank policy and switches to plain BitTorrent bartering, *i* can unchoke fast agents only. Those agents reciprocate because they use the rank policy and *i* has high reputation. Therefore, *i* improves its expected utility and by Definition 3.6, using the rank policy is not a Bayes-Nash equilibrium strategy. □

We verify this result experimentally in the following chapter.

Three questions arise from Theorem 4.16. The first is what exactly constitutes a high, or high enough, reputation. Second is whether this is a sustainable strategy. Third is the question how large the expected gain is from disabling the rank policy.

To determine what a high (enough) reputation is for the manipulation to be successful, we first recognize that the manipulation succeeds for an agent with the reputation of 1 because it can barter with any agent in the swarm. The lower the agent's reputation, the lower the probability that it can find bartering partners with higher slot capacity than with the rank policy enabled, over all its upload slots. Figure 4.13 (left) shows some distribution $R(r)$ of the reputation scores over the agents. For an agent with reputation score a, the aforementioned probability is then given by Equation 4.4, which gives the fraction of the agents that that agent can barter with:

$$\int_{-1}^{a} Rt(r)\,dr \tag{4.4}$$

A user that knows its reputation score can thus calculate the probability that switching to plain bartering will decrease its download completion time. Note that this is an upper bound, because an agent needs time to explore the network in order to find the better bartering partners.

To answer the second question, we recognize that in order to gain from disabling the rank policy, first the user needs to build up a high reputation, then turn off the rank policy in order to download faster. Following our analysis in Section 4.1.3, an agent downloads the same volume it uploads if it barters with agents with equal slot capacity exclusively. If such an agent has high reputation, its reputation remains high and it can successfully continue downloading with disabled rank policy. However, if it barters with faster agents, it will download more than it uploads, causing its reputation to drop. That requires the user to enable the rank policy again to boost its reputation.

Consider the right part of Figure 4.13, which shows some distribution $L(l)$ of the agents over the slot capacities (l) of the agents. The probability that switching to plain bartering is a sustainable strategy is the probability that an agent with slot capacity $l_i = \frac{c_{u,i}}{k_i}$ barters with agents with lower slot capacity. This probability is given by

$$\int_{0}^{l_i} L(l)\,dl \tag{4.5}$$

This shows that the higher an agent's slot capacity, the longer it can profit from switching to plain bartering. In the above, we have seen that faster agents increase their reputation at a higher rate because they barter with slower agents more. Combined, we find that switching to plain bartering is attractive for fast agents with high reputation. These are exactly the agents that we expect to profit from switching.

This leaves us with the last question: what is the expected gain for an agent? With BarterCast enabled, we approximate agent i's download rate by $k_i\bar{l}$, where $\bar{l}$ represents the weighted average of slot capacities in the swarm. Denote i's reputation by a. Then, Equation 4.4 represents the fraction of the agents that i can barter with. Then, i's gain is bounded by:

$$l_{max} \int_{-1}^{a} D(r)\,dr - \bar{l} \tag{4.6}$$

This shows that the larger an agent's reputation, the higher its gain.

Subjective Reputation

In the above, we assumed that all agents have the same ranking over the agents: a global ranking. However, as described in Section 4.3.1, BarterCast is inherently subjective and globally inconsistent.

An agent i using BarterCast bases its reputation of another agent j on its direct experience with j, and information about j obtained from others. Meulpolder et al. [28] reason that by approximation every peer has a reputation score for every other peer in the network. We therefore ignore the possibility that i has no information on j at all.

This leaves the possibility that i's reputation of j is either too high or too low. Meulpolder et al. show, based on simulations, that the average of all agents $i \in N \setminus \{j\}$ of j is a good representation of j's real behavior. More research is needed to determine the variance of j's reputation among all other agents, but lacking that, our assumption of a global ranking is plausible.

Relationship Between Reputation And Upload Capacity

At the beginning of this section, we assume that there is no relationship between an agent's reputation and its upload capacity. We are not aware of any research that investigates a possible relationship, but our results in Section 4.1.3 suggest that such a relationship is plausible.

Faster agents upload more than they download, and the difference becomes larger if they are not in equal relationships, as is the case when BarterCast with the rank policy is added to BitTorrent. Therefore, when a fast agent completes its download, it has boosted its reputation. A slow agent, on the other hand, uploads less than it downloads and its reputation decreases. After download completes, that agent needs to seed the file in order to boost its reputation again. In line with our finding in Section 4.3.2, this shows that the long-term seeding incentive offered by the rank policy is stronger for the slower agents.

The above also shows that it is easier for a fast agent to obtain a high reputation. That implies that there is a correlation between reputation and relative slot capacity, rendering our assumption of no relationship between the two invalid. That, in turn, would decrease the difference in download rate for a fast agent between bartering under the rank policy and plain bartering, possibly moving enabling the rank policy more towards an equilibrium strategy. We leave analyzing this possibility for future research.

4.3.3 Conclusions

BarterCast is a reputation mechanism that is added to the Tribler network to provide agents with a long-term seeding incentive. It keeps a reputation score for every agent, and allows agents to base their unchoking decisions on that. Meulpolder et al. [28] suggest two unchoking policies that are based on the BarterCast reputation: the ban-policy and the rank-policy. In this section, we investigate whether users have an incentive to use BarterCast in a network where all other users do so.

We find that using the ban-policy is a Bayesian-Nash equilibrium, as no user can benefit from not using it if all other users do. Using the rank policy, on the other hand, is not an equilibrium strategy for the fast agents – these can improve on their download completion time by switching to plain bartering. For these fast agents, we find that switching is a sustainable strategy as well, as those agents continue to keep their reputations high. Additionally, we find that the long-term seeding incentive using the rank policy is stronger for the slower agents than for the faster agents.

We verify experimentally whether these findings hold in practice in the following chapter.

Chapter 5

Experiments

In the previous chapter, we used our model from Chapter 3 to predict the outcome of the BitTorrent mechanism, i.e., to predict which agent classes an agent is likely to be bartering with. This allowed us to both predict and explain observed phenomena like stratification and segmentation, skewed upload/download ratios, and download completion times. For some types of users, under some circumstances, we show that the BitTorrent file-sharing game is in a Nash equilibrium. Using the BarterCast reputation mechanism with the rank policy is an equilibrium strategy for the slower agents, but not for the fastest. In this chapter, we verify these results with simulations.

Section 5.1 describes the setup of our experiments. Section 5.2 investigates our claims on stratification in BitTorrent networks. This is followed by an analysis on the formation of bartering relationships in Section 5.3. Section 5.4 verifies the outcome of our prediction on download completion times. In Section 5.5, we consider the special case of small swarms. In Section 5.6, we investigate whether the Nash equilibria we identified in Section 4 exist in practice. Section 5.7 investigates whether using BarterCast is an equilibrium strategy. Finally, in Section 5.8, we draw conclusions.

5.1 Experiment Setup

For our experiments, we use TriblerSim 1.0, a simulator that is developed by Michel Meulpolder at the Tribler group of Delft University of Technology [30]. This simulator features a full implementation of the BitTorrent protocol and simulates BitTorrent swarms on a single computer.

We run our experiments on two types of swarms: swarms with 5% seeders, and with 45% seeders. With our experiments on the swarms with 5% seeders, our goal is to demonstrate the validity of our model and our claims in Section 4 on stratification and bartering relationships. In our model, we did not consider seeders. We add seeders to the network in our experiments to ensure that all file pieces are available in the swarm. To prevent agents from downloading from the seeders extensively, all seeders have very low upload capacity.

The second types of experiments, on swarms with 45% seeders, are motivated by a study by Dán and Carlsson [13], who performed measurements on 1,690 BitTorrent trackers and investigated about 330,000 swarms. They find that on one particular date in 2008, 21 million leechers and 17 million seeders participate in 3.3 million unique swarms – roughly 45% seeders per swarm.

Their study also finds that that are few very large swarms, and many small swarms; the distribution of swarm sizes is given in Figure 5.1. Based on this, we run our simulations on swarms of 20, 50, and 200 agents.

As we explained in Section 4.1.2, different agents may have different upload capacities. For every swarm, we define the same six classes of upload capacities and distribute the agents evenly over these classes. The composition of the swarms is given in Table 5.1. Agents with more upload

Leechers (%)	Swarm size	Upload capacity (KB/s) 256 s	256 l	512 s	512 l	1024 s	1024 l	2048 s	2048 l	4096 s	4096 l	8192 s	8192 l
45	20	1	2	1	2	2	2	1	2	1	2	2	2
	50	3	5	3	5	4	5	3	5	3	5	4	5
	200	13	20	13	20	14	20	13	20	13	20	14	20
5	20	1	3	0	3	0	3	0	3	0	3	0	3
	50	2	8	0	8	0	8	0	8	0	8	0	8

Table 5.1: *Swarm sizes and number of seeders (s) and leechers (l) per upload capacity, in swarms with 5% and 45% seeders.*

Upload capacity (KB/s)	Upload slots	Slot capacity
256	3	85
512	5	102
1024	8	128
2048	10	205
4096	12	341
8192	14	585

Table 5.2: *Number of upload slots used with given slot capacity, and the resulting slot capacity. Note that both upload capacity and slot capacity increase at every step, but slot capacity increases at a slower rate.*

capacity have more upload slots, in line with most BitTorrent implementations (see Section 3.1.1). In our simulations, we use the same settings as the Vuze client. Refer to Table 5.2 for these settings, and to Section 4.2.1 for more information on this.

We specify upload capacity only, and assume that an agent's download capacity is unlimited. Even though this is a simplification, in reality download capacity is a number of times larger than upload capacity and is not considered a bottleneck.

In the swarms with 45% seeders, all agents enter the swarm simultaneously. Leechers have no pieces of the file, while seeders have the full file available. Because, as we found in Section 4.2.2, leechers have no long-term seeding incentive, a leecher leaves the swarm as soon as its download completes.

In the swarms with 5% seeders, the simulator is in steady-state mode. Here, all agents are created with a random number of file pieces available. A leecher leaves the swarm when its download completes, and its place is taken by another leecher with the same upload capacity but no file pieces. In our results in the remainder of this section, we only consider agents that finish the download.

There is one swarm only. In this swarm, one of two files can be exchanged; one of 512 MB and the other of 2048 MB. In the following, we refer to the former as a *small file*, and to the latter as a *large file*. Each file is divided into pieces of 1024 kB each. All experiments are run ten times and the results of these runs are aggregated.

5.2 Stratification

The main result of Section 4 is Claim 4.10, which states that stratification occurs in general in a BitTorrent network. In order to verify this claim, we introduce the *Stratification Index*. The stratification index takes the average over all agents of the number of stable relationships an agent is in, divided by the number of bartering slots. If a network has a stratification index of 1, then

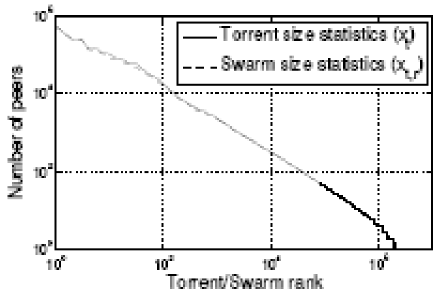


Figure 5.1: *Swarm size as function of swarm rank. Figure taken from [13].*

all agents are in equal relationships (a bartering relationship with an agent with the same upload capacity) over all their bartering slots, and the network is fully stratified. A lower index means that some agents are in unequal relationships and that there are bartering relationships between agent classes.

We now formally define this index. First, we define $e_i(t)$ as the number of equal relationships that agent i is in at time t, and $n(t)$ as the number of leechers present in the network at time t. Recall from Section 3 that k_i is the number of upload slots for agent i.

Definition 5.1. *The* Stratification Index *of a BitTorrent network at any time $t \in T$ is given by*

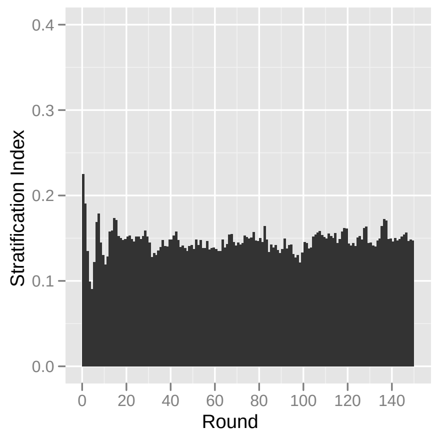
$$\frac{1}{n(t)} \sum_{i=0}^{n(t)} \frac{e_i(t)}{k_i}$$

.

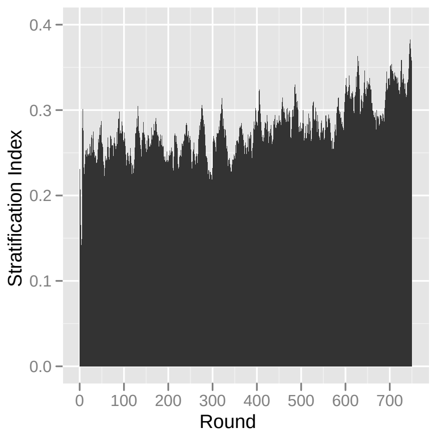
Figures 5.2 (a) and (b) show the stratification index of the networks with 5% seeders. As mentioned in the previous section, agents leave these networks when their download completes, and their place is taken by a new agent with the same upload capacity. From our model, we expect the stratification index to decrease at such an event, because relationships are broken. After the event, we expect the index to rise again until the next event. Indeed, we find (in all figures in Figure 5.2) that the index drops or raises up to 30% in a few rounds.

Surprisingly, the index in the network with the large file is consistently some 80% higher than in the network with the small file. Our model provides no explanation for this phenomenon. As we do expect from our model, the trend of the index increases slightly over time, indicating a progression towards a stable relationship. However, even after 750 rounds, the index is just over 0.3, indicating that only about $\frac{1}{3}$ of the slots is used for equal relationships. This is not the strong stratification that our model predicts. One possible explanation for this is that because of the Rarest Piece First selection strategy, our assumption of a global ranking on upload capacity alone is not valid. Another explanation is that the difference in upload slots used (14 for the fastest, versus 3 for the slowest) causes the faster agents to connect to so many agents that they must resort to lower-ranked agents. We investigate this further in the following section.

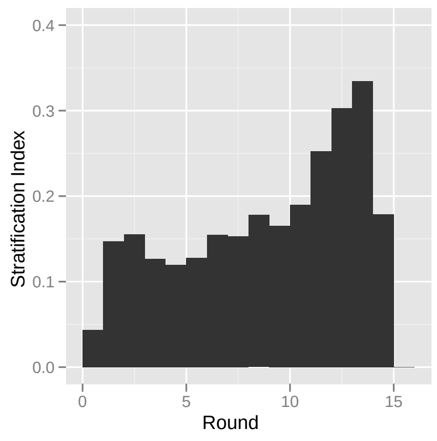
Figures 5.2 (c) through (f) show the stratification index of the networks with 45% seeders, where agents leave the network as their download completes, but no new agents join. In the larger networks with the small file, the index has a strong increasing trend as we expect from our model. However, the results in the networks with 5% seeders suggest that progression towards a stable configuration is not as strong as the model predicts, and there is another reason for the stratification index to rise to rise this high in the networks with many seeders. From our model, we expect the faster agents to finish their download sooner. We validate this in Section 5.4. This would provide an additional



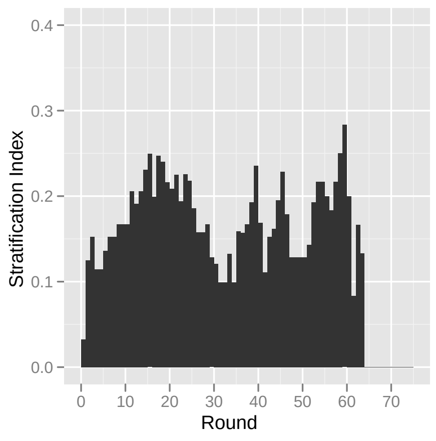
(a) 5% seeders, swarm size 50, file size 512 MB



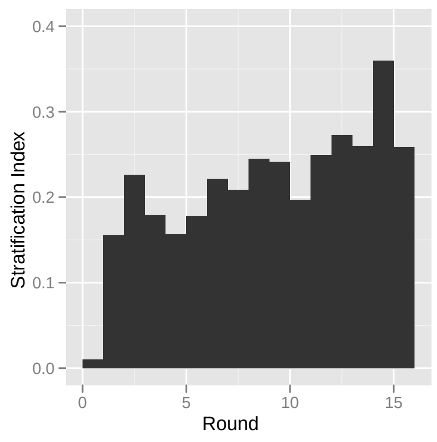
(b) 5% seeders, swarm size 50, file size 2048 MB



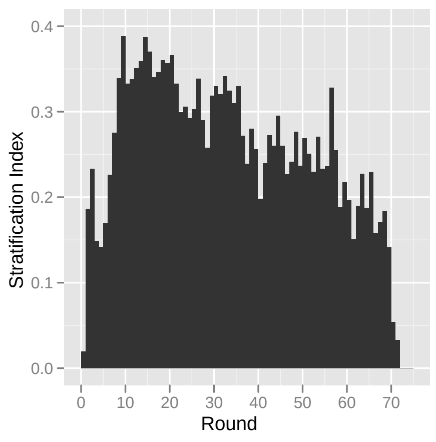
(c) 45% seeders, swarm size 50, file size 512 MB



(d) 45% seeders, swarm size 50, file size 2048 MB



(e) 45% seeders, swarm size 200, file size 512 MB



(f) 45% seeders, swarm size 200, file size 2048 MB

Figure 5.2: *Stratification index per round for different networks.*

reason for the index to increase near the end of file exchange in these networks: because fewer agent classes are present, any agent is more likely to be bartering with an equal agent. This does not hold in the swarms with 5% seeders because there all agent classes remain present.

In the networks where the large file is exchanged, the index more than doubles in the first 20 rounds, similar to the networks with the smaller file, but then halves over the next 20 rounds. This is when the fastest agents start leaving the network (again, see Section 5.4). When this happens, the remaining slower agents need to find new bartering partners, and explore the network, thereby breaking stable relationships. The index then rises quickly, until the next class of agents leaves the network, etc. This shows that dynamic aspects as agents leaving the network have a significant impact on the its stratification, as we predicted in Section 4.1.2.

Figure 5.3 provides another look at stratification. Here, we display the fraction of data that is uploaded from every agent class to all agent classes over all runs.

From our model, we expect that every class allocates most data to itself. Instead, we find that the three fastest classes group together and allocate most data to each other. Similarly, the slowest three classes allocate most data among themselves. Agents in both groups hardly seem to distinguish their own class at all. The slowest agents with upload capacity of 256 KB are the exception, as they hardly receive any data from the two classes above them, even though they upload the majority of their data to these classes. They are forced to get their data from the seeders in the network. These results are statistically significant with 95% confidence level, based on an independent two-sample t-test with equal variance.[1]

Figure 5.3 shows that in the swarms with the small file, all classes upload most of their data to the faster classes. With the larger file, the results are more in line with the model, as the majority of the data is uploaded to the own class. This suggests that the small file is downloaded too fast for the agents to find stable relationships before the download completes.

In line with our earlier finding, Figure 5.3 shows that stratification is not as strong as predicted by our model.

5.3 Bartering Relations

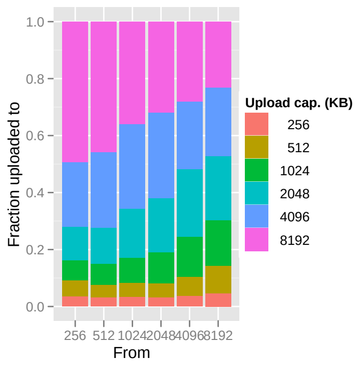
In the previous section, we found that stratification does occur, but not as strong as we predicted. Stratification relies on stable relationships between equal agents, and in this section we investigate the relationships in the different networks.

Figure 5.4 shows the average number and average length of relationships in three different networks. In the first two networks, a large file is exchanged in a network with 45% seeders in swarms of 50 and 200 agents. As the trend in these figures is the same, we perform the third experiment is in a swarm of 200 agents only. Here, a small file is exchanged in a swarm with 5% seeders.

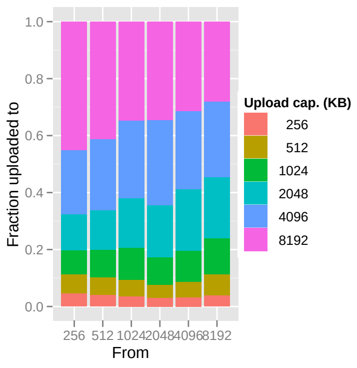
First we discuss the experiments with 45% seeders in Figures 5.4 (a) through (d). Based on our model, we expect that agents have stable relationships with agents of their own class, which would result in few, but long-lasting, equal relationships. In practice, we find that that is not true. In the swarm of 200 agents (figures (c) and (d)), we find that the two fastest classes (agents with upload capacities of 4096 and 8192 kB/s) group together and have relatively few, but longest relationships with each other. This is remarkable, because the fastest agents have 14 upload slots, and there are 20 leechers in each class. Based on these numbers, we would expect every agent to be able to barter with its own class exclusively.

In the swarm of 50 agents (figures (a) and (b)), the three fastest classes barter most among each other, which makes sense because the three classes together those hold 15 leechers. Because the fastest agents cannot find all their relationships in their own class, they resort to bartering with lower

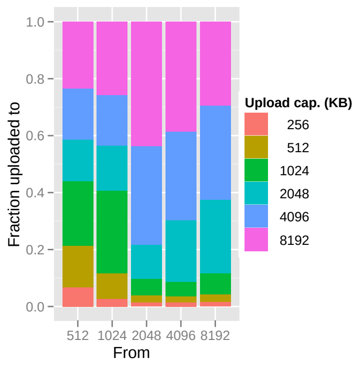
[1]This is what we mean by *statistically significant* in the remainder of this chapter.



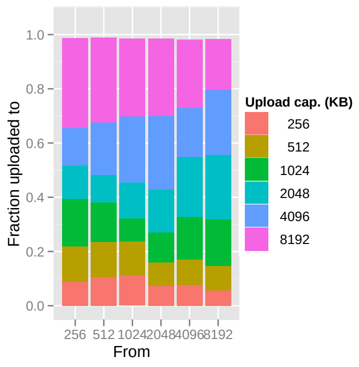
(a) 5% seeders, small file, swarm size 50



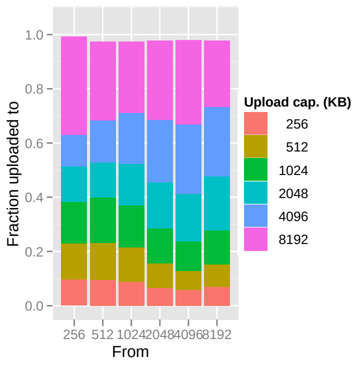
(b) 5% seeders, small file, swarm size 200



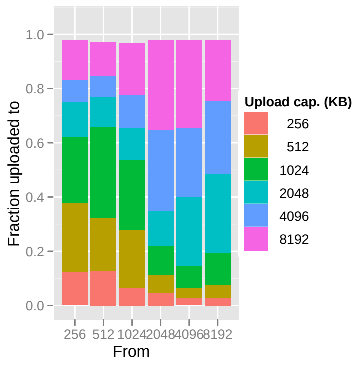
(c) 5% seeders, large file, swarm size 50



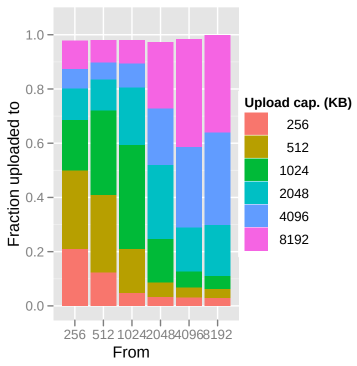
(d) 45% seeders, small file, swarm size 50



(e) 45% seeders, small file, swarm size 200

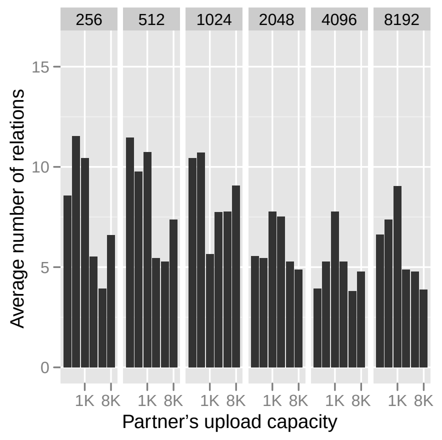


(f) 45% seeders, large file, swarm size 50

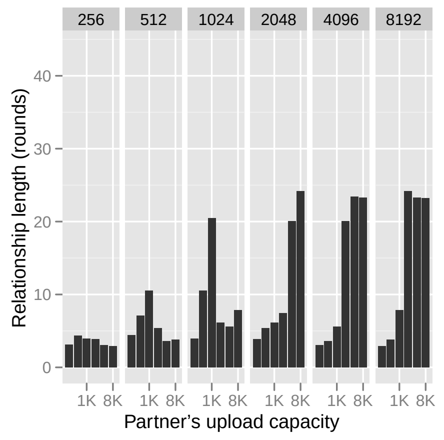


(g) 45% seeders, large file, swarm size 200

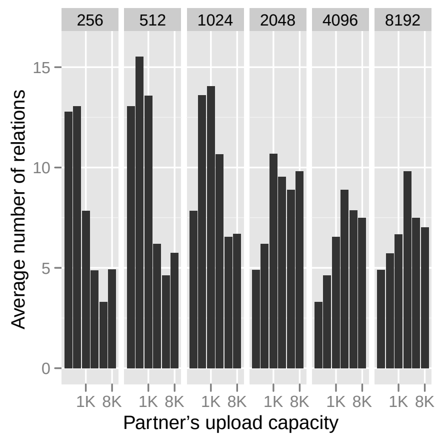
Figure 5.3: *Average fraction of data uploaded to all agent classes over all rounds, broken down by uploading agent class.*



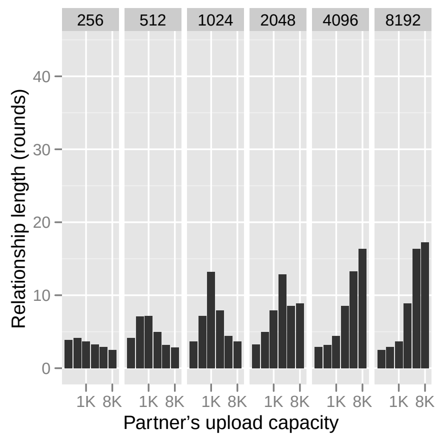
(a)



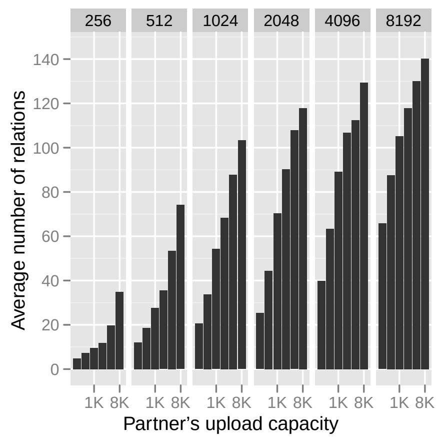
(b)



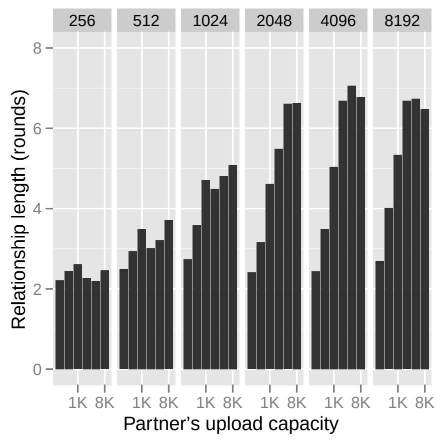
(c)



(d)



(e)



(f)

Figure 5.4: *Average number (left) and length (right) of relationships in three different networks. The top figures show the results for a network of 50 agents, 45% seeders, and a large file. In the middle figures, parameters are the same except for the swarm size which is 200. The bottom figures show the results for a network of 200 agents, 5% seeders, and a small file. Note that the units are different for these figures.*

classes. Agents in the lower classes happily reciprocate, and as a result assign their upload slots to a class other than their own. These results are significant.

The agents with upload capacities of 1024 (and 2048 in the large swarm) do have longest relationships with equal agents. This is because the best-ranked agents barter mostly among themselves, so those agents are the best-ranked agents among the remaining agents. These results are statistically significant.

For the remaining slowest agents, we find that they have short relationships with any class. The slowest agents have only few upload slots available, and whenever they are optimistically unchoked by a faster agent, they break an equal relationship reciprocating to that agent. As a result, they are unable to form long-lasting relationships. This also explains why the slowest agents have the largest average number of relationships. Note that in this experiment, no agents join the network. When the faster agents leave the network as their download completes, the slower agents remain and relationships become more equal. This means that the averages are skewed over time, and average relationship length is for the slowest agents is shorter when all agent classes are present.

We find that all agent classes have significantly largest number of relationships with agents of their own class, or one class higher or lower. This suggests that stratification does not happen strictly within one class, but between an agent's own class, and one class higher and lower.

Next, we discuss the experiment in the network with 5% seeders. Here, we find that every class has more relationships with the faster classes. The fastest three classes have the longest relationships amongst these three classes, and again the fastest agents do not seem to distinguish between the fastest three classes. The slowest agents have short relationships with all classes. This is statistically significant.

The difference between the two is that in the latter network, new agents join the swarm as agents complete their download. In the former two networks, the faster agents leave the network sooner than the slower agents, forcing the slower agents to barter with each other. In the latter network, the slow agents are unable to form relationships because there are always more attractive faster agents present.

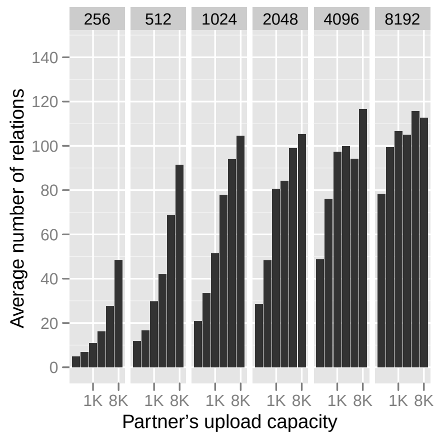
In the above, we found that one possible explanation that stratification is not as strong as our model suggests is that the faster agents connect to slower agents because they have so many upload slots. To verify whether this is the case, we ran an experiment in a network of 50 agents, 5% seeders, and a small file, where all agents have 5 upload slots. The results of this are in Figure 5.5 and show that then, all agents have longest relationships with their own classes. However, the results are not entirely convincing, because the fastest agents have equal relationship length with their own class and one class lower, and the class of 2,048 KB has most relationships with the fastest two classes. Still, the results are more in line with our model.

Another possible explanation for the observed behavior is the Rarest Piece First piece selection strategy (see Section 3.1). This could render a slower agent more attractive than a faster agent because it has rarer file pieces. In order to verify this, we replaced the Rarest Piece First strategy with a random piece selection strategy. Figure 5.5 shows the results of this for a network of 5% seeders, 50 agents, and a small file as well. We find that that the difference is small: all agent classes have most and longest relationships with the fastest agents, instead of with their own class.

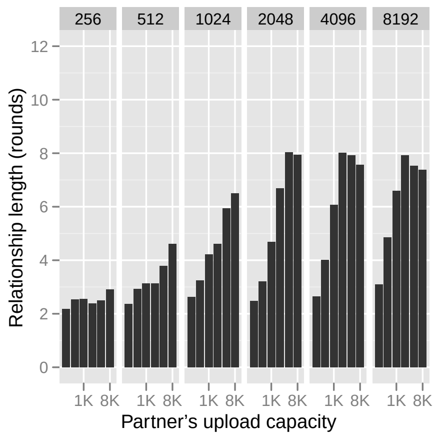
We conclude that the major reason for the observed low stratification is the large number of upload slots for the faster agents.

5.4 Download Completion Times

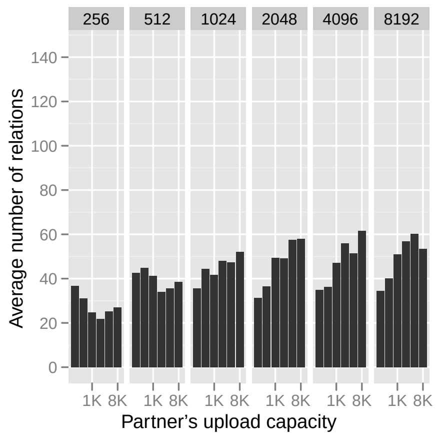
In Section 4.1.3, we calculate the download completion time for different agent classes based on the assumption of a stable configuration. We repeated these calculations for the networks from our experiments, and list the results in Table 5.3. The actual results from our experiments are provided



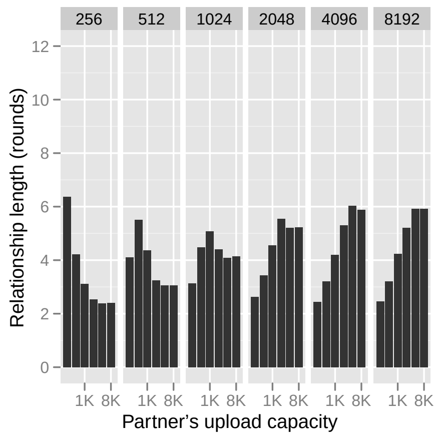
(a) Number of relations



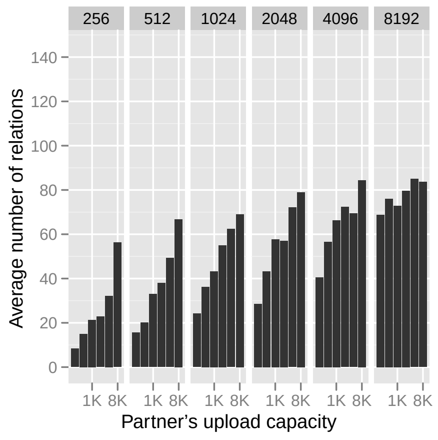
(b) Length of relations



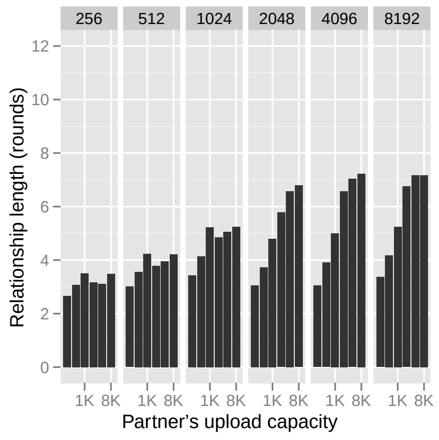
(c) Number of relations



(d) Length of relations



(e) Number of relations



(f) Length of relations

Figure 5.5: *Number and length of relationships in a network with 5% seeders, swarm size 50, and file size 512 MB. Upper figures show results with Rarest Piece First piece selection strategy. Middle figures show results with random piece selection strategy. Bottom figures show results when all agents have 5 upload slots.*

	File size	
Upload capacity (KB/s)	512 MB	2048 MB
256	2000	8000
512	1000	4000
1024	500	2000
2048	250	1000
4096	125	500
8192	62.5	250

Table 5.3: *Prediction of download completion (s) time for different upload capacities and file sizes, based on the assumption of a stable configuration.*

in Figure 5.6. The difference between the calculated and actual results is dramatic, and indicates that the network configuration is far from stable.

Because of its large network size of 200 agents, we would expect that agents in the network from Figure 5.6 (b) would find equal relationships, resulting in a stable configuration. However, the fastest agents are four times slower than predicted, while the slowest agents are almost five times faster. This follows naturally from the network configuration not being stable, as explained in Section 4.1.3.

There are two other observations from Figure 5.6. First, we notice the dramatic impact that seeders have on the download performance: compare Figure 5.6 (a) with (d), or (b) with (e) and find that a download can be completed 20 times faster with 45% seeders instead of 5%. This result is statistically significant.

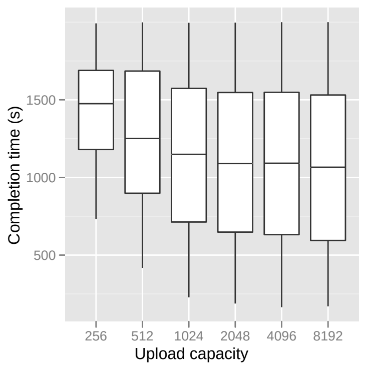
The second observation is that the differences in download completion times are much larger in the networks with many seeders: in figure (f), the fastest agents finish their download 2.8 times as fast as the slowest, while in figure (d), this factor is only 1.5. This is also statistically significant. This is an incentive for agents to specify their full download capacity in swarms with many leechers.

In the networks with 5% seeders, the fastest three classes have roughly equal completion time, but that of the slowest classes is different. In the other networks, however, we find that the difference in completion times of the fastest and slowest two classes are not always statistically significant. This is explained by our finding in the previous section that agents form relatively many relationships with neighboring classes.

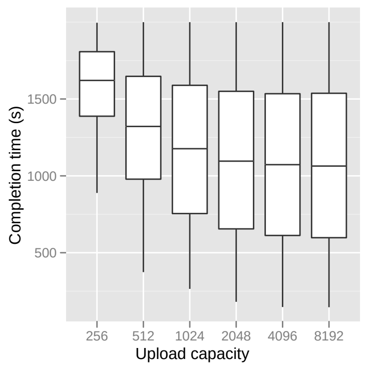
5.5 Small Swarms

In the above, we presented the results for relatively large swarms of 50 and 200 agents. In the latter swarms, we expected every agent to be bartering with agents from its own class only, but we found that this is not the case. In this section, we discuss results for very small swarms. As we described in Section 4.2.1, in a small swarm all agents barter with each other and stratification cannot occur.

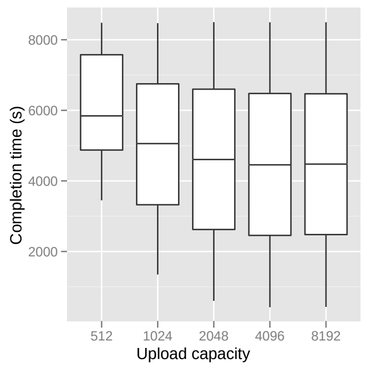
The results of these experiments are in Figure 5.7. As expected, the stratification index is very low: below 0.1. The fastest agents are connected to all other agents because of their large number of upload slots. As a result, all classes reciprocate to these agents the most. This is most dramatic for the slowest agents, because they have only few upload slots, and therefore upload the majority of their data to the fastest agents. Except for the slowest agents, all agents take the same time to finish their download. This is exactly what we expect from our model.



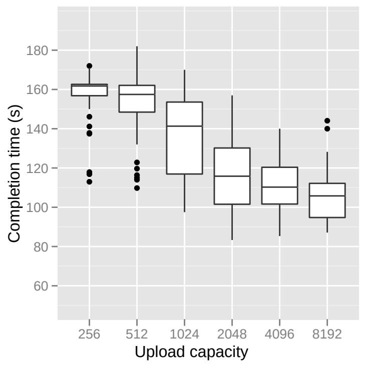
(a) 5% seeders, small file, swarm size 50



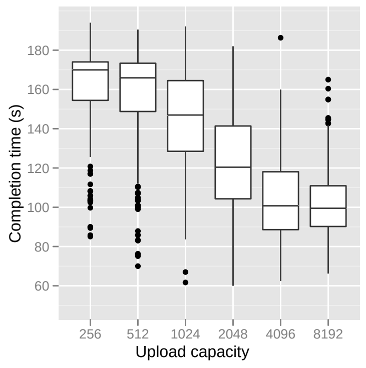
(b) 5% seeders, small file, swarm size 200



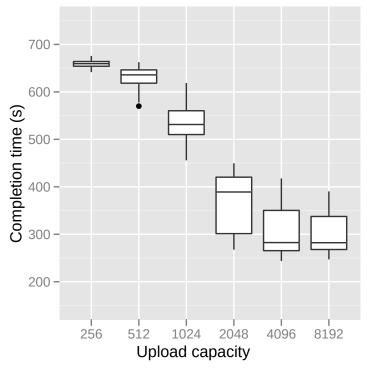
(c) 5% seeders, large file, swarm size 50



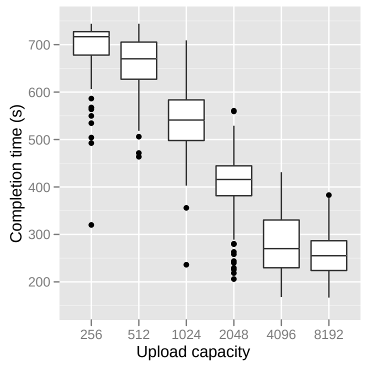
(d) 45% seeders, small file, swarm size 50



(e) 45% seeders, small file, swarm size 200

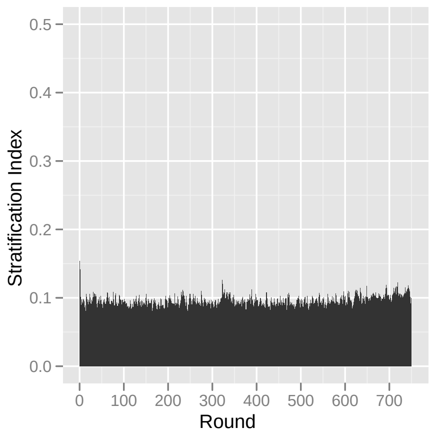


(f) 45% seeders, large file, swarm size 50

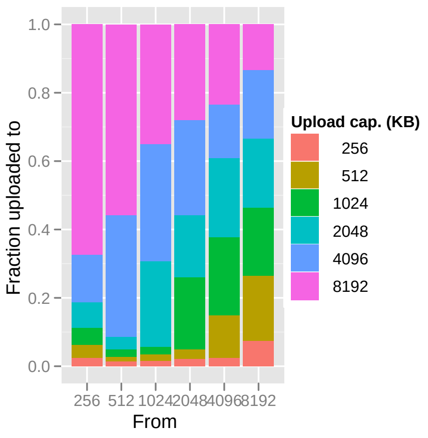


(g) 45% seeders, large file, swarm size 200

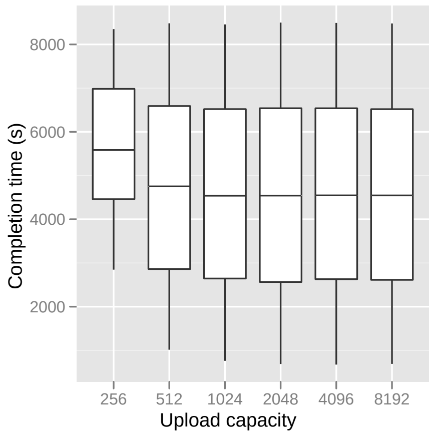
Figure 5.6: *Mean download completion time per agent class in different networks. Note that the units vary per file and swarm size.*



(a) Stratification index



(b) Fraction of data uploaded to all agent classes



(c) Completion time

Figure 5.7: *Results of a small swarm of 20 agents, with 20 agents, in which a large file is exchanged.*

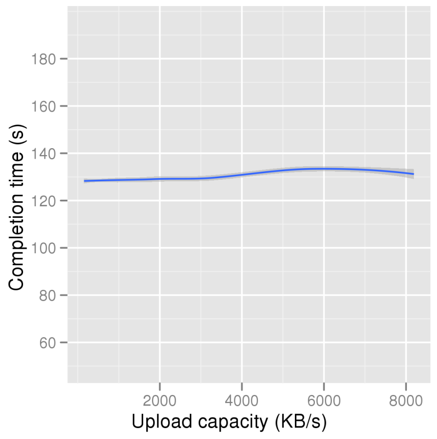
5.6 Equilibria

Theorem 4.12 states that in a stable configuration of a large network[2], specifying full upload capacity is a dominant strategy. Figure 5.8 shows the download completion time for an agent in a swarm of 20 and 50 agents as a function of its upload capacity.

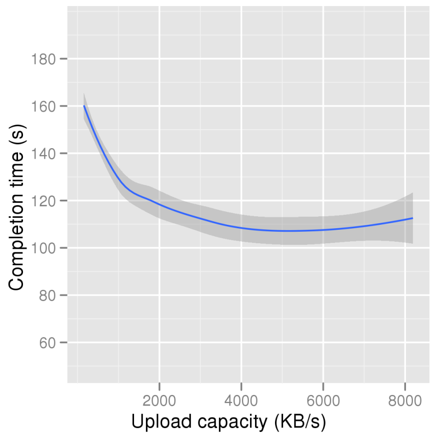
Figure 5.8 (a) shows the result for a swarm of 20 agents. As expected from our earlier results in this section, a user gets no increase in utility because all agents finish roughly at the same time. In the larger network of figure (b), however, we find that a user with an upload capacity larger than 4000 kB/s can specify an upload capacity of 4000 kB/s without an increase in completion time. As we found before, the network configuration is not stable which results in specifying full upload capacity not being a dominant strategy.

As we explain in Section 4.2, average users only specify the upload capacity and do not change

[2]See Definition 4.9

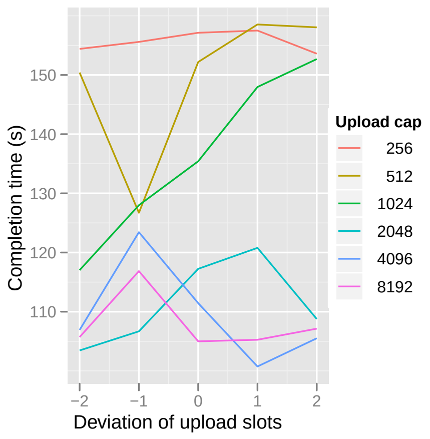


(a) Swarm size 20

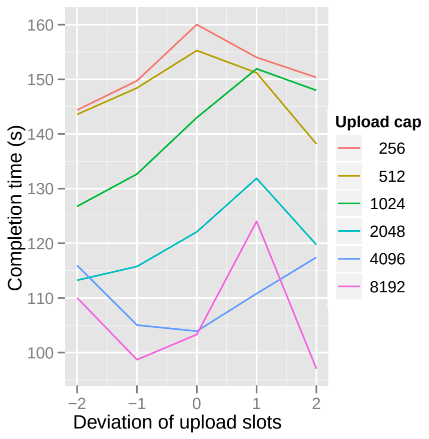


(b) Swarm size 50

Figure 5.8: *Download completion time as a function of upload capacity in a network with 45% seeders, in which a small file is exchanged. Data from all ten runs aggregated.*



(a) Swarm size 50



(b) Swarm size 200

Figure 5.9: *Download completion time as a function of the deviation of the number of upload slots specified. A deviation of 0 represents the suggested setting from Table 5.2. A deviation of -2 represents two slots less, and a deviation of +2 represents two upload slots more. Experiments are performed in a network with 45% seeders, in which a small file is exchanged. Data from all ten runs aggregated.*

other settings. Advanced users, on the other hand, may change any setting, including the number of upload slots. With Theorem 4.14, we prove that specifying the suggested number of upload slots is a Bayes-Nash equilibrium strategy in a stable configuration of a large network.

Figure 5.9 shows the download completion times for an agent in a swarm of 50 and 200 agents, who deviates from the specified number of upload slots by one or two. We find that the results are wildly varying, depending on swarm size and upload capacity. For most combinations, however, it is possible to obtain lower download completion time (thereby increasing utility) by specifying the right number of upload slots. From our limited experiment, this increase can be up to 15%. This shows that when a configuration is not stable, specifying the suggested number of upload slots is not a dominant strategy. In an unstable configuration, agents barter with agents from other ranks. Deviating from the default number of upload slots allows some agents to barter with additional agents, while it allows other agents to barter with higher-ranked agents – both increase an agent's utility. However, the results show that it is also possible that because of the deviation, an agent barters with fewer, or lower-ranked agents, which decreases its utility.

These results show that an advanced user can increase its utility by carefully selecting the upload capacity and number of upload slots. The problem here is that it is difficult to predict before joining the network what number of upload slots will provide the best utility, as we explained in Section 4.2.1, and the cost of performing these calculations manually will outweigh the gain in utility for many users. However, a BitTorrent client that dynamically adapts its configuration to the network observed can achieve better download performance for its user. Such a client could calculate which agents it could connect to for a certain range of upload slots, and select the number that provides best utility accordingly. More research is needed to determine whether this strategy would lead the agents to an equilibrium, or whether it forces them to keep changing this number, in a network where all agents use this client.

5.7 BarterCast

In Section 4.3, we described the BarterCast reputation system that is added to Tribler to provide agents with a long-term seeding incentive. Agents can use the reputation kept by BarterCast to decide on which agents to unchoke. The reputation score can be evaluated using different policies, and we described the rank-policy in detail. Our main result of that section is Theorem 4.16, which states that in a network where all agents use the rank-policy, an agent with high reputation can improve on its download performance by disabling that policy and switching to plain BitTorrent bartering instead. In this section, we verify this result with simulations, along with two other results from Section 4.3.

5.7.1 Simulation Setup

The setup for the experiments in this section builds on the setup described in Section 5.1. The unchoking mechanism of the agents (both leechers and seeders) is extended so that an agent either uses the rank policy, or plain bartering. In our experiments, an agent uses one policy only during its lifetime.

We simulate an agent's participation in a single swarm. Using BarterCast, an agent's contributions in multiple swarms are considered, and the agent's contributions in the single swarm we simulate are negligible compared to the contributions it did during its lifetime. This implies that an agent's reputation remains almost constant in a single swarm. In our simulation, we therefore keep any agent's reputation constant. Instead of fully implementing BarterCast into the simulator, this allows us to simulate the rank policy with a simpler setup, as we describe in the following.

Every agent knows its own contributions, and can calculate its own reputation score using Equation 4.3. We refer to this reputation as the agent's *real reputation*. We randomly provide every agent with a real reputation between -1 and 1. BarterCast does not keep global reputation scores. Instead, every agent has its own subjective view on the reputations of the other agents. We simulate this by providing each agent i with a reputation score for every other agent j, which is randomly drawn from the normal distribution around j's real reputation with a standard deviation of 0.2 (10% of the interval between -1 and 1).

The simulations are run with the simulator in steady-state mode (see Section 5.1). The swarm consists of 103 agents. There are three seeders to guarantee that all pieces are available, each with an upload capacity of 256 KB/s. The 99 leechers in the swarm are equally divided over the three upload capacities of 512, 2048, and 8192 KB/s. Each of these 102 agents uses the rank-policy. Finally, there is a single agent that uses plain bartering,[3] to verify our claim that this agent can improve its download rate if it has high reputation.

For every experiment, the single agent has a fixed upload capacity of 512, 2048 or 8192 KB/s (the same classes as the other agents), and a real reputation of -1, -0.85, -0.7, -0.5, 0, 0.5, or 0.95. The simulation for each combination of upload capacity and reputation score is run ten times, and the results of these runs are aggregated.

The file exchanged is 1 GB, and we simulate 3,500 seconds to allow even the slowest agents to finish their downloads. Contrary to our setup described in Section 5.1, the number of upload slots for an agent does not depend on its upload capacity, but is fixed on five slots instead. This is to eliminate any effects the number of upload slots can have on completion time.

5.7.2 Expected Results

We verify three hypotheses. First, to investigate whether agents have an incentive to have a high reputation, we examine whether download completion times decrease with increasing reputation, as we expect from Section 4.3. In the remainder of this section, we refer to this incentive as the *reputation incentive*.

Second, we verify Theorem 4.16, which states that an agent with high reputation improves on its completion time if it switches from using the rank policy to plain bartering, as that allows it to barter with the fastest agents, instead of with a random selection of agents. This is to investigate whether agents have an incentive to actually use BarterCast for their unchoking decisions, and we refer to this incentive as the *unchoking incentive*.

If BarterCast provides agents with both the unchoking incentive and the reputation incentive, then every rational agent will use BarterCast and and strive for a high reputation value. The latter it can only achieve by uploading more than it downloads, which means that BarterCast is successful in providing a long-term seeding incentive.

The final hypothesis we verify is that, because all agents barter with a random selection of agents, there is no correlation between upload capacity and completion time. This would justify our reasoning in Section 4.3 that the capacity ranking used in BitTorrent is replaced by the reputation ranking when BarterCast is added to the network.

5.7.3 Completion Time and Reputation

We start by examining our first hypothesis: whether in this network, download completion times are decreasing with increasing reputation. Figure 5.10 shows the completion time for the agents that use the rank policy only. The blue line shows the average completion time, and clearly verifies our hypothesis.

Download completion times decrease most for increasing lower reputations. The difference between minimal and neutral reputation is a factor of 25. If an agent's reputation increases from 0 to 0.5, it has a 41% decrease of completion time, while the decrease is only 7% if its reputation increases from 0.5 to 1. There is a strong incentive to improve on a negative reputation, but the incentive to improve on an already positive reputation is not nearly as strong.

[3]Recall from Section 4.3 that we refer to BitTorrent's default unchoking mechanism as *plain bartering*. We say that agents *use the rank-policy* if they unchoke agents based on their BarterCast reputation score evaluated with the rank-policy.

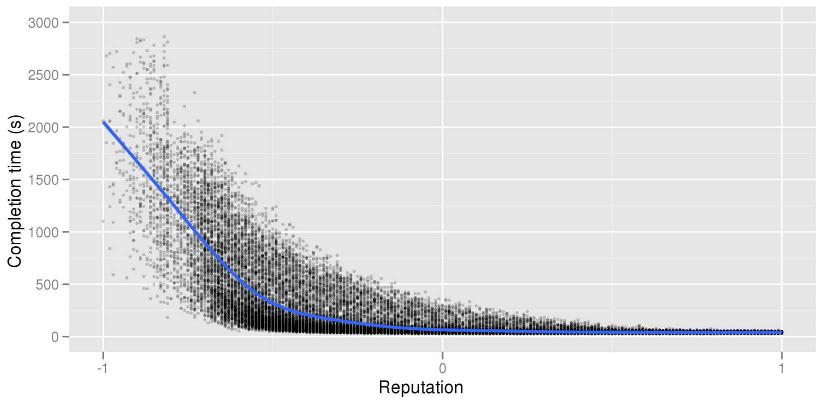


Figure 5.10: *The blue line shows the average download completion time as a function of reputation value for agents using the rank policy. As we expect, completion time is monotonically decreasing with increasing reputation, although differences are not as significant for reputations above 0.*

5.7.4 Improving Completion Time by Switching to Bartering

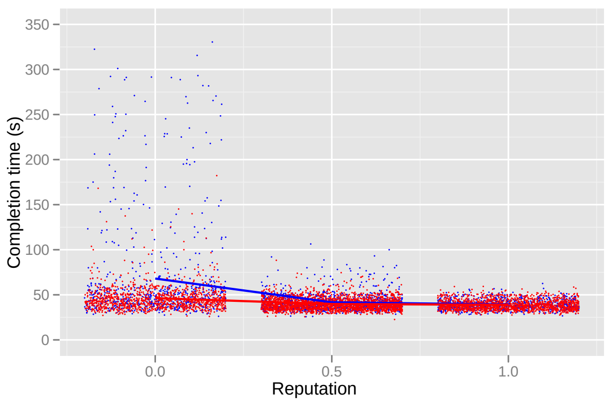
We now verify the second hypothesis: that an agent with high reputation can improve its completion time by switching to plain bartering in a network with different agent classes. The results from our experiments are given in Figure 5.11.

Shown in that figure are scatter plots of the completion time for the seven real reputation values for which we ran simulations (see Section 5.7.1), jittered around the corresponding reputation value to increase readability. Blue dots show completion times for the agents using the rank policy, and red dots or plain bartering agents. Blue and red lines connect the averages of the corresponding values.

Our simulations ran for 3,500 seconds, which proved too short as none of the plain bartering agents with a real reputation value of -1 and only a fraction of those with reputation of -0.85 finished their download completely in that period. For the other real reputation values, 3,500 seconds was sufficient. For the agents with reputation of -1 and -0.85, we extrapolated completion times by considering their downloaded volume and the corresponding download time. We did this for both bartering agents and agents using the rank policy. Because agents need time to explore the network, extrapolating results from agents that downloaded only a small fraction of the file may produce skewed results. Therefore, we only considered agents that downloaded at least 512 MB of the 1,024 MB file. Because completion times depend on reputation value (see the previous section), the number of data points for each evaluated reputation score varies: from 1017 for a reputation of -1 to 2258 for a reputation of 1. For the lowest reputations of -1 and -0.85, the original simulations provided so few points that we ran those simulations an additional 10 times and aggregated the results.

The top figure shows the completion time for agents with a reputation between 0 and 1. Confirming Theorem 4.16, an agent with real reputation of 1 improves its completion time by switching to bartering, but only by 2%. This result is not statistically significant, based on an independent two-sample t-test with equal variance and 95% confidence level (p-value is 0.18). For larger files, we expect a bigger improvement because the agent has more time to explore the network and to profit from its better bartering partners.

Peers with a reputation of 0 and 0.5 improve more on their completion time if they switch from using the rank policy to plain bartering: up to 31% for an agent with a real reputation of 0. These results are statistically significant, with p-values below 10^{-6}. The results contradict Theorem 4.16,



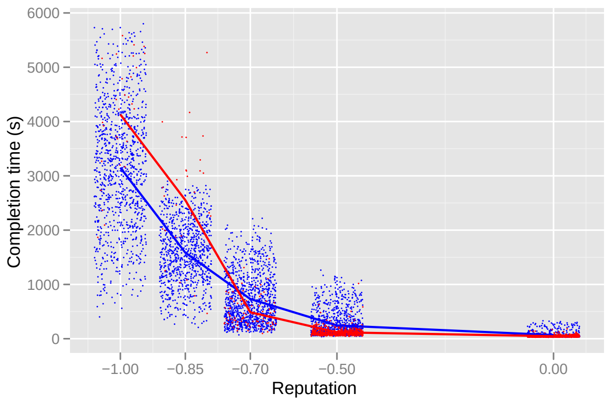


Figure 5.11: *The blue line shows the download completion time for agents using the rank policy, while the red line shows completion time for an agent switching to plain bartering. Confirming Theorem 4.16, the bartering agent with high reputation improves on its completion time, but by a meager 2% only. Contradicting that theorem, the improvement holds for agents with a reputation over -0.7, even though the theory predicts that these agents do not profit from switching to bartering. An agent needs a real reputation of below -0.7 for that to happen.*

which predicted improvements only for agents with high reputation. This is explained from the fact that the file is exchanged in under 70 seconds, or seven rounds. Agents using the rank policy query the reputation system in order to decide who to unchoke. Bartering agents, on the other hand, optimistically unchoke all agents they are aware of, and as a result they are faster in exploring the network. We expect that larger files will result in higher completion times for plain bartering agents with neutral reputation.

The bottom figure in Figure 5.11 shows the results for agents with negative reputation. All differences between the averages for bartering or using the rank policy here are statistically significant, with p-values below 10^{-6}. From this figure, we find that an agent needs a real reputation below -0.7 before using the rank policy gives it better performance than bartering. However, the difference is considerable: 32% for a reputation of -1. Bartering agents with the lowest reputations hardly receive any pieces in return for their optimistic unchokes. However, if their reputation is -0.7 or above, their reputation is high enough to be served. We expect the pivot point, that is now between -0.85 and -0.7, to shift to a higher reputation value with increasing file size, because then the agents using the

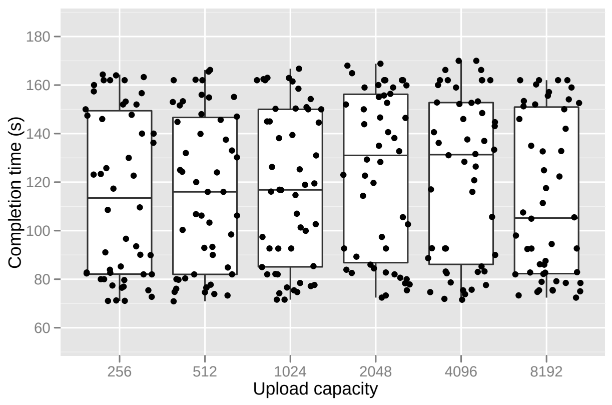


Figure 5.12: *Completion time by peer class in a network where all agents use BarterCast with the rank-policy, and reputation scores are normally distributed over the agents. Download completion time does not depend on upload capacity.*

rank policy have more time to explore the network to find higher reputed agents.

5.7.5 Upload Capacity and Download Completion Time

We now verify the final hypothesis we formulated in Section 5.7.2: whether download completion time depends on an agent's upload capacity. Because we want to compare our result with a similar experiment in Section 5.4, we use a different setup than the one we described in the previous section. In this experiment, the swarm consists of 27 leechers and 23 seeders, evenly divided over six peer classes as in Section 5.4, exchanging a 512 MB file. The reputation scores are randomly distributed over the agents. All agents use the rank-policy. The results are given in Figure 5.12.

Download completion times are roughly equal for each of the peer classes. The figure shows different median completion times; however, none of these differences are statistically significant, with a p-value of at least 0.16.

If we compare Figure 5.12 with Figure 5.6 (d), which shows the completion times in the same network, with all agents using plain bartering, we find that the rank-policy indeed changes the outcome of the network, as completion time no longer depends on upload capacity.

5.8 Conclusions

In this section, we presented the results from our experiments on the validity of the outcome of our model. The major result is that while stratification does occur in practice, it is not as strong as we expect with only about 30% of the bartering slots used for equal relationships. This means that the network configuration is far from stable. We do find that the fastest agent classes barter significantly with neighboring fast classes, which suggests that stratification is stronger in the fastest classes. However, contrary to what our model predicts, we find that these fastest classes distinguish little amongst each other. The main reason for this is their large number of upload slots, which allows them to barter to their own class, as well as one class lower.

The network not nearly reaching a stable configuration has major impact on user utility. We show in Section 5.4 that the fastest agents are four times slower than predicted, while the slowest agents are almost five times faster. This has a downside and an upside: it makes the network attractive to slower agents, but provides no incentive to the faster agents to specify full upload capacity. Even for the slower agents, the difference in download completion time is often not statistically significant from that of a neighboring class. Thus, even the slowest agents have no incentive to specify their full upload capacity. However, when many seeders are present, fast agents' utilities increase dramatically. If download capacity were linked to upload capacity, this would provide that incentive.

An unstable network configuration also results in specifying the suggested number of upload slots not being a dominant strategy. As a result, an advanced user can gain up to 15% (in our experiments; possibly more in other networks) in download performance by manipulating the number of upload slots. Figure 5.9 shows that the actual benefit of manipulating upload slots is difficult to predict, and may result in increased or decreased performance. Therefore, this may not be a practical form of manipulation for advanced users. However, it does show that using a (not yet existing) BitTorrent client which dynamically adapts its configuration on the network it is bartering in in order to achieve the best download performance can result in increased performance. Whether an equilibrium exists in a stable or unstable network with multiple such clients (and whether the agents' actions will converge to that equilibrium if it exists) is an interesting question for future research.

We verified whether using the rank-policy, download completion time is independent of upload capacity. We find that this is indeed the case, and that completion time decreases with increasing reputation. However, where the difference in completion time for an agent with lowest and neutral real reputation is very large, for neutral and high reputation it is not nearly as large, and for reputation values of 0.5 and 1 it is negligible. This means that there is a strong incentive for agents to have positive reputation, but the incentive to improve on an already positive reputation is small.

We verified whether an agent with highest reputation of 1 can improve on its download time if it switches to plain bartering in a network where all other agents use the rank policy. Our results confirm this, although the gain is 2% only and not statistically significant. We expect this gain to increase with increasing file size, as then the agents have more time to profit from the faster bartering partners. Contradicting our theoretical results, the increase holds for all other agents with a real reputation of -0.7 and above, and those results are statistically significant.

In conclusion, we find that BarterCast with the rank policy provides a reputation incentive, i.e., agents have an incentive to improve on their reputation by sharing more than they download. However, this incentive strongly decreases once reputation has reached a positive value. For this incentive to work, all agents in the network need to use the rank policy for their unchoking decisions. From our simulations, we find that agents with a real reputation of -0.7 and above improve on their completion time by switching to plain BitTorrent unchoking instead. BarterCast with the rank policy does not provide all agents with an unchoking incentive. This raises the question whether the reputation incentive is still provided if some fraction of the agents does not use BarterCast, and for what fraction the incentive disappears. We leave this for future research.

Meulpolder et al. [28] find that the rank policy is not very effective in preventing free-riding. The results above confirm this finding. The rank policy could be more effective if the incentive to improve on positive reputations were stronger, and if there are no users that have an incentive to turn off BarterCast. How to accomplish this is an open question for future research.

Chapter 6

Conclusions and Recommendations

The central question of this thesis is formulated in Section 1.1: do BitTorrent and BarterCast provide incentives to lazy free-riding users to share? To answer this question, we used game theory to model agents in a generic P2P file-sharing network. We applied this model to BitTorrent, and used the model to predict the outcome of the BitTorrent network. Following that, we extended the model with BarterCast and predicted the outcome for the extended network as well. This allowed us to answer the central question. We verified the results from our analysis experimentally.

In this chapter, we review our work in this thesis. In Section 6.1 we discuss our conclusions, and in Section 6.2 we identify viable directions for future research.

6.1 Conclusions

In this section, we answer our central research question.

We start by investigating BitTorrent's incentives. We used the model we developed in Chapter 3 to investigate the effects of different lazy free-riding strategies, and verified our results experimentally. We come to the following conclusions:

1. BitTorrent offers an incentive to share to users whose actions are limited to specifying their upload capacity, as they achieve best results from the network if they dedicate their full upload capacity to the network (Theorem 4.12);

2. For users whose actions additionally include specifying the number of upload slots, BitTorrent offers an incentive to share only in a network that is large enough that, for every agent, there are multiple agents with similar upload capacity (Theorems 4.14 and 4.15);

3. Experiments show that the two theoretical results above are not as strong in practice, because there peers join and leave the network continuously. Conclusion 1 holds for the slower fraction of the agents only; the faster agents are not given an incentive to dedicate their full upload capacity to the network. Even in large enough networks, Conclusion 2 could not be verified experimentally.

Prior work, such as Qiu and Srikant [39], Gai et al. [20], and Fan et al. [14], found a result that is similar to Conclusion 1, but with an important distinction: in their work, the result depends on the knowledge and beliefs that an agent has of the other agents in the network. In our work, the result is independent of any prior knowledge or beliefs, which makes it a stronger one.

Conclusions 2 and 3 show that BitTorrent does not provide an incentive to share to all users. Users can significantly improve on their download completion times by selecting the optimal number of upload slots. This optimal number depends on the upload capacities and number of upload slots the other users in the network selected. These are unknown before joining the swarm, and a miscalculation can result in significantly larger completion times. This limits the applicability of this lazy free-riding strategy.

Next, we study BarterCast's incentives. We distinguish two different incentives: to get a good reputation score by uploading more data than downloading (the reputation incentive), and to use information provided by BarterCast to determine who to upload to (the unchoking incentive). BarterCast only provides an incentive to share if it provides both the reputation and the unchoking incentive. We come to the following conclusions:

4. BarterCast provides an unchoking incentive to low-reputed agents, but not to highly reputed agents (Theorem 4.16), as the latter can improve on their completion time by not using BarterCast;

5. We show experimentally that Conclusion 4 is partially true in practice: the unchoking incentive is provided to the 15% lowest reputed agents, while the remainder of the agents is better off not using BarterCast;

6. We show experimentally that BarterCast provides a reputation incentive because download time decreases with increasing reputation.

It is BarterCast's goal to provide a long-term seeding incentive, so that agents continue sharing a file long after they have downloaded it. In a network where all users decide who to upload to based on BarterCast's information, it is the reputation incentive that provides long-term seeding incentive. However, because BarterCast offers no unchoking incentive, not all users will be using BarterCast. This limits the applicability of the reputation incentive, and thereby decreases BarterCast's long-term seeding incentive.

In conclusion, we find that both BitTorrent and BarterCast are not fully incentive compatible for lazy free-riders. The efforts needed for lazy free-riding in BitTorrent are substantial because a user needs to calculate and specify its optimal settings, while the benefits are unsure. In practice, we expect that these two factors combined deter users to free-ride lazily, and as such can be seen as an additional incentive not to free-ride.

6.2 Recommendations

From our work follow, in our opinion, some interesting issues that could be addressed in future work:

- In Section 3.2, we model P2P file-sharing as a static one-shot game, and extend this to a dynamic game. In experiments, we find that the dynamic aspects of P2P file-sharing affect the outcome more than we expect. It would be interesting to investigate whether using elements from online or repeated mechanism design in our model would make for a better fit between theory and practice;

- In this thesis, our focus is on lazy free-riders. However, our work identifies two novel die-hard free-riding strategies, in Sections 4.1.3 and 5.6. For both strategies, the effectiveness needs to be evaluated, as well as their effect on the network and the other users' completion times;

- BarterCast is the first successfully deployed distributed reputation mechanism. In this thesis, we considered lazy free-riding in BarterCast only: using BarterCast, or not. We expect that BarterCast will be used in more settings than file-sharing alone, and that it will be subject to more die-hard free-riding attempts. We propose to assess BarterCast's vulnerability to die-hard free-riding and to investigate its incentive compatibility in order to identify possible room for improvements;

- We evaluated the rank policy in combination with BarterCast, and found that although peers have a strong incentive to have a reputation that is at least neutral, the incentive to improve on a positive reputation is not that strong. In addition, some agents have an incentive to not use the rank policy. Intuitively, the rank policy seems more fair and better applicable than the more successful ban-policy. It would be interesting to identify how the rank policy can be adjusted so that the incentive to improve on a positive reputation is stronger, whether that incentive still holds if some fraction of the agents does not use BarterCast, and how all agents can be given an incentive to use BarterCast;

- One of the motivations for considering lazy free-riders in this thesis is that even though die-hard free-riding clients for BitTorrent are available, their use is not wide-spread. In the Kazaa network, on the other hand, a successful die-hard free-riding client caused the demise of the network. It would be interesting to determine why die-hard clients do not catch on in BitTorrent. This study will be psychological more than technical, but is likely to provide BitTorrent's system designers with valuable insights on the effectiveness of their product.

Bibliography

[1] Vuze community forums. Retrieved May 30, 2010, from http://forum.vuze.com.

[2] BitTorrent still king of P2P traffic. Retrieved May 30, 2010, from http://torrentfreak.com/bittorrent-still-king-of-p2p-traffic-090218/, February 2009.

[3] Good settings - VuzeWiki. Retrieved May 30, 2010, from http://wiki.vuze.com/index.php/Good_settings, 2009.

[4] Eytan Adar and Bernardo A. Huberman. Free riding on Gnutella. *First Monday*, 5, October 2000.

[5] Nazareno Andrade, Miranda Mowbray, Aliandro Lima, Gustavo Wagner, and Matei Ripeanu. Influences on cooperation in bittorrent communities. In *P2PECON '05: Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, pages 111–115, New York, NY, USA, 2005. ACM.

[6] Moshe Babaioff, John Chuang, and Michal Feldman. *Algorithmic Game Theory*, chapter Incentives in Peer-to-Peer Systems, pages 593–611. Cambridge University Press, 2007.

[7] Yoram Bachrach, Ariel Parnes, Ariel Procaccia, and Jeffrey Rosenschein. Gossip-based aggregation of trust in decentralized reputation systems. *Autonomous Agents and Multi-Agent Systems*, 2008.

[8] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and improving a BitTorrent network's performance mechanisms. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–12, April 2006.

[9] Kevin L. Brown and Yoav Shoham. *Essentials of Game Theory: A Concise, Multidisciplinary Introduction*. Morgan and Claypool Publishers, 2008.

[10] Jonathan S. K. Chan, Victor O. K. Li, and King-Shan Lui. Performance comparison of scheduling algorithms for peer-to-peer collaborative file distribution. *Selected Areas in Communications, IEEE Journal on*, 25(1):146–154, 2007.

[11] B. Cohen. Incentives build robustness in BitTorrent. In *Proc. of the Workshop on Economics of Peer-to-Peer Systems*, 2003.

[12] Bram Cohen. The BitTorrent protocol specification, version 11031. Retrieved May 30, 2010, from http://www.bittorrent.org/beps/bep_0003.html, January 2008.

[13] György Dán and Niklas Carlsson. Dynamic swarm management for improved BitTorrent performance. In *Proc. International Workshop on Peer-to-Peer Systems (IPTPS '09)*, Boston, MA, USA, April 2009.

[14] Bin Fan, Dah M. Chiu, and John Lui. The delicate tradeoffs in BitTorrent-like file sharing protocol design. In *ICNP '06: Proceedings of the Proceedings of the 2006 IEEE International Conference on Network Protocols*, pages 239–248, Washington, DC, USA, 2006. IEEE Computer Society.

[15] Joan Feigenbaum, Michael Schapira, and Scott Shenker. *Algorithmic Game Theory*, chapter Distributed Algorithmic Mechanism Design, pages 363–384. Cambridge University Press, 2007.

[16] Joan Feigenbaum and Scott Shenker. Distributed algorithmic mechanism design: recent results and future directions. In *DIALM '02: Proceedings of the 6th international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 1–13, New York, NY, USA, 2002. ACM.

[17] Michal Feldman, Kevin Lai, Ion Stoica, and John Chuang. Robust incentive techniques for peer-to-peer networks. In *EC '04: Proceedings of the 5th ACM conference on Electronic commerce*, pages 102–111, New York, NY, USA, 2004. ACM.

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

[19] Anh-Tuan Gai, Dmitry Lebedev, Fabien Mathieu, Fabien de Montgolfier, Julien Reynier, and Laurent Viennot. Acyclic preference systems in P2P networks. In *Euro-Par 2007 Parallel Processing*, pages 825–834. Springer Berlin / Heidelberg, 2007.

[20] Anh-Tuan Gai, Fabien Mathieu, Fabien D. Montgolfier, and Julien Reynier. Stratification in P2P networks, Application to BitTorrent. In *Proceedings of ICDCS'07, International Conference on Distributed Computing Systems 2007*, Toronto Canada, 2007. IEEE Computer Society.

[21] Philippe Golle, Kevin Leyton-Brown, Ilya Mironov, and Mark Lillibridge. Incentives for sharing in peer-to-peer networks. In *WELCOM '01: Proceedings of the Second International Workshop on Electronic Commerce*, pages 75–87, London, UK, 2001. Springer-Verlag.

[22] David Hales, Rameez Rahman, Boxun Zhang, Michel Meulpolder, and Johan Pouwelse. BitTorrent or BitCrunch: Evidence of a credit squeeze in BitTorrent? In *WETICE '09: Proceedings of the 2009 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*, pages 99–104, Washington, DC, USA, 2009. IEEE Computer Society.

[23] Ramayya Krishnan, Michael D. Smith, and Rahul Telang. The economics of peer-to-peer networks. *The Journal of Information Technology Theory and Application (JITTA)*, 5(3):31–44, 2003.

[24] Rakesh Kumar and Keith Ross. Optimal peer-assisted file distribution: Single and multi-class problems. In *IEEE Workshop on Hot Topics in Web Systems and Technologies (HOTWEB)*, 2006.

[25] Arnaud Legout, Nikitas Liogkas, Eddie Kohler, and Lixia Zhang. Clustering and sharing incentives in BitTorrent systems. *SIGMETRICS Perform. Eval. Rev.*, 35(1):301–312, 2007.

[26] Dave Levin, Katrina Lacurts, Neil Spring, and Bobby Bhattacharjee. BitTorrent is an auction: analyzing and improving BitTorrent's incentives. In *SIGCOMM Comput. Commun. Rev.*, volume 38, pages 243–254, New York, NY, USA, 2008. ACM.

[27] 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. *IEEE/ACM Trans. Netw.*, 14(5):978–991, October 2006.

[28] M. Meulpolder, J. A. Pouwelse, D. H. J. Epema, and H. J. Sips. Bartercast: A practical approach to prevent lazy freeriding in P2P networks. In *Proc. of the 6th International Workshop on Hot Topics in Peer-to-Peer Systems (Hot-P2P'09) in conjunction with IPDPS 2009*, May 2009.

[29] M. Meulpolder, J. A. Pouwelse, D. H. J. Epema, and H. J. Sips. Modeling and analysis of bandwidth-inhomogeneous swarms in BitTorrent. In S.N., editor, *Proc. of IEEE P2P 2009*, pages 232–241, Los Alamitos, USA, September 2009. IEEE Computer Society.

[30] Michel Meulpolder. TriblerSim 1.0. Retrieved May 30, 2010, from `http://tribler.org/trac/wiki/P2PSimulator`, 2010.

[31] Seth James Nielson, Scott A. Crosby, and Dan S. Wallach. A taxonomy of rational attacks. In Miguel Castro and Robbert van Renesse, editors, *IPTPS*, volume 3640 of *Lecture Notes in Computer Science*, pages 36–46. Springer, 2005.

[32] Noam Nisan. *Algorithmic Game Theory*, chapter Introduction to Mechanism Design (For Computer Scientists), pages 209–242. Cambridge University Press, 2007.

[33] Noam Nisan and Amir Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35(1-2):166–196, April 2001.

[34] David C. Parkes and Jeffrey Shneidman. Distributed implementations of Vickrey-Clarke-Groves mechanisms. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 261–268, Washington, DC, USA, 2004. IEEE Computer Society.

[35] Michael Piatek, Tomas Isdal, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. Do incentives build robustness in BitTorrent? In *Proceedings of 4th Usenix Symposium on Networked Systems Design and Implementation (NSDI)*, Cambridge, MA, USA, April 2007.

[36] Michael Piatek, Tomas Isdal, Arvind Krishnamurthy, and Thomas Anderson. One hop reputations for peer to peer file sharing workloads. In *NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 1–14, Berkeley, CA, USA, 2008. USENIX Association.

[37] J. A. Pouwelse, P. Garbacki, Wangand, J. Yang, A. Iosup, D. Epema, M. Reinders, M. R. van Steen, and H. J. Sips. Tribler: A social-based based peer to peer system. In *5th Int'l Workshop on Peer-to-Peer Systems (IPTPS)*, Feb 2006.

[38] Johan A. Pouwelse, Pawel Garbacki, Dick H. J. Epema, and Henk J. Sips. The Bittorrent P2P file-sharing system: Measurements and analysis. In Miguel Castro and Robbert van Renesse, editors, *IPTPS*, volume 3640 of *Lecture Notes in Computer Science*, pages 205–216. Springer, 2005.

[39] Dongyu Qiu and R. Srikant. Modeling and performance analysis of BitTorrent-like peer-to-peer networks. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 367–378, New York, NY, USA, 2004. ACM Press.

[40] Stefan Saroiu, Krishna P. Gummadi, and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Multimedia Computing and Networking (MMCN)*, San Jose, CA, USA, January 2002.

[41] Sven Seuken. Distributed online mechanism design: Improving incentive compatibility in peer-to-peer file-sharing networks. Unpublished draft version, 2007.

[42] J. Shneidman, D. Parkes, and L. Massoulie. Faithfulness in internet algorithms. In *Proc. SIGCOMM Workshop on Practice and Theory of Incentives and Game Theory in Networked Systems (PINS'04), Portland, OR, USA*, September 2004.

[43] Jeffrey Shneidman and David C. Parkes. Specification faithfulness in networks with rational nodes. In *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 88–97, New York, NY, USA, 2004. ACM.

[44] Jeffrey Shneidman, David C. Parkes, and Margo Seltzer. Overcoming Rational Manipulation in Distributed Mechanism Implementations. Technical Report TR-12-03, Harvard University, Cambridge, MA, USA, 2003.

[45] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, New York, 2009.

[46] Michael Sirivianos, Jong H. Park, Rex Chen, and Xiaowei Yang. Free-riding in BitTorrent networks with the large view exploit. In *IPTPS*, 2007.

[47] Jie Tang. Informativeness and incentive compatibility for reputation systems. Bachelor thesis, Harvard College, Cambridge, MA, USA, 2008.

[48] Dimitrios Vassilakis and Vasilis Vassalos. An analysis of peer-to-peer networks with altruistic peers. *Peer-to-Peer Networking and Applications*, 2(2):109–127, 2009.