A TAXONOMY-BASED PERSPECTIVE OF THE DESIGN TRADE-OFFS FOR BITTORRENT-LIKE PROTOCOLS

WANG YOUMING

(B.Eng.(Hons.), NUS)

A THESIS SUBMITTED FOR THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2013

DECLARATION

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

[illegible]

Wang Youming
29 Jan 2013

Acknowledgements

First and foremost, I would like to thank my supervisor Prof. Ben Leong, who has taught me and guided me in many ways for the past four years. He taught me in the Facebook class (CS3216), mentored me in the CVWO project, employed me to work as a research assistant on the TFTTP project and supervised me to complete this Master thesis in the end. I have learned many useful lessons from him, not only about research, but also more importantly about life. His words are insightful, his conduct is a live demonstration of his core belief. His passion for teaching and spirit of striving for excellence have deeply affected me. I think I am going to pursue a career in education sector as well, since to inculcate my knowledge to my students through teaching is one of the greatest delights of my life. I am deeply grateful to Ben for helping me to find my calling.

I also owe my gratitude to Prof Teo Yong Meng for guiding me in many ways in my research work. His strong background and experience of network system research has broaden my thinking and helped me to conduct my work in a more systematic ways.

I would like to thank my colleagues Dr. Su Wen and Cristina. Both of them are more senior than I and have more research experience than I do. They pointed out my limitation in my thinking and experiment design and suggest many useful improvements, and helped me to better adapt to the research environment.

I would like to thank my friends Guo Xiangfa and Liu Xiao who has made my NUS life more memorable. I also would like to thank Wang Wei, Xu Yin, Gong Jian, Yu Guoqing, Leong Wai Kay, Daryl Seah and Ali Razeen for being my wonderful and helpful lab mates.

I owe my deep gratitude to my parents for loving me and praying for me, especially during my life in Singapore. They are wonderful parents who I cherish deeply and dearly. I would like to thank my newly married wife Kang Pei for accompanying me for the past one and half years through my joy and sorrow, my wellness and sickness. Her presence has brought much delight to my life. I thank God for His blessing by bringing her into my life.

Last but certainly the most importantly, I would like to thank my Savior and Lord Jesus Christ. His love to me surpasses knowledge and is everlasting and ever fresh. I would like to dedicate my whole life to experience His love and love Him in return.

Table of Contents

List of Tables

List of Figures

Abstract

In recent years, BitTorrent (BT) has become the most popular peer-to-peer file sharing protocol. However, in spite of its popularity, the protocol has many vulnerabilities that can be exploited by strategic peers. Some recent work studied the trade-offs involved in BitTorrent algorithm, but the exploration of the design space has not been comprehensive. In the dissertation, we propose a new taxonomy-based approach for analyzing the trade-offs in a practical implementation of the BT protocol and investigate these trade-offs in the protocol design space. Finally, we propose two key design principles we gleaned from our experience working with various BT clients: (i) keeping promises and (ii) keeping information up-to-date.

Chapter 1

Introduction

BitTorrent (BT) [5] has in recent years become the predominant means for peer-to-peer (P2P) content distribution on the Internet. A number of BT variants have also been proposed over the past few years to address various issues like fairness [16] and strategic peers [13, 14]. Given its importance to file sharing, it is important to understand how different elements in the protocol will affect performance.

To the best of our knowledge, Fan et al. [6] were the first to propose a mathematical model that allows us to tradeoff performance for fairness in BT by adjusting the ratio of regular unchokes to optimistic unchokes in BT protocol. We found that in addition to this ratio, there are many other mechanisms that can affect the trade-offs between performance and fairness that are not captured in their model. We believe that because the implementation of the BT protocol is inherently complex, the trade-off between performance and fairness cannot be adequately captured with a limited mathematical framework, such as the one proposed by Fan et al.

1.1 Our Approach

Therefore, we propose a new taxonomy-based approach for analyzing the trade-offs that takes into consideration the practical implementation of the BT protocol. To this end, we analyzed a number of the available BT variants, including BitTyrant [14], BitThief [13], PropShare [11], FairTorrent [16] and Azureus 3.0.4 [1], and compared them to the default BT protocol [5] to come up with a taxonomy, based on the following four key decisions made by the various protocols:

1. **Number of connections** , i.e. how many peers does each node connect to?
2. **Number of unchokes**, i.e. how many peers does a node service simultaneously?
3. **Peer selection strategy**, i.e. how does a node decide which peer(s) to unchoke?
4. **Uplink bandwidth allocation**, i.e. how much data is to be uploaded to each unchoked peer?

The resulting taxonomy is shown in Figure 1.1.

We modified the Azureus BT client to comply with the behaviour of original BT protocol to act as the baseline comparison and added additional code to record key activities, like choke messages. We also augmented the client with additional command-line arguments to allow us easily change various parameters and modified the client to support both seeding mode and non-seeding mode, where nodes leave immediately upon completing a download. We did the same for the other clients like BitTyrant [14], PropShare [11] and FairTorrent [16]. We also modified the choke/unchoke algorithm to include new one,

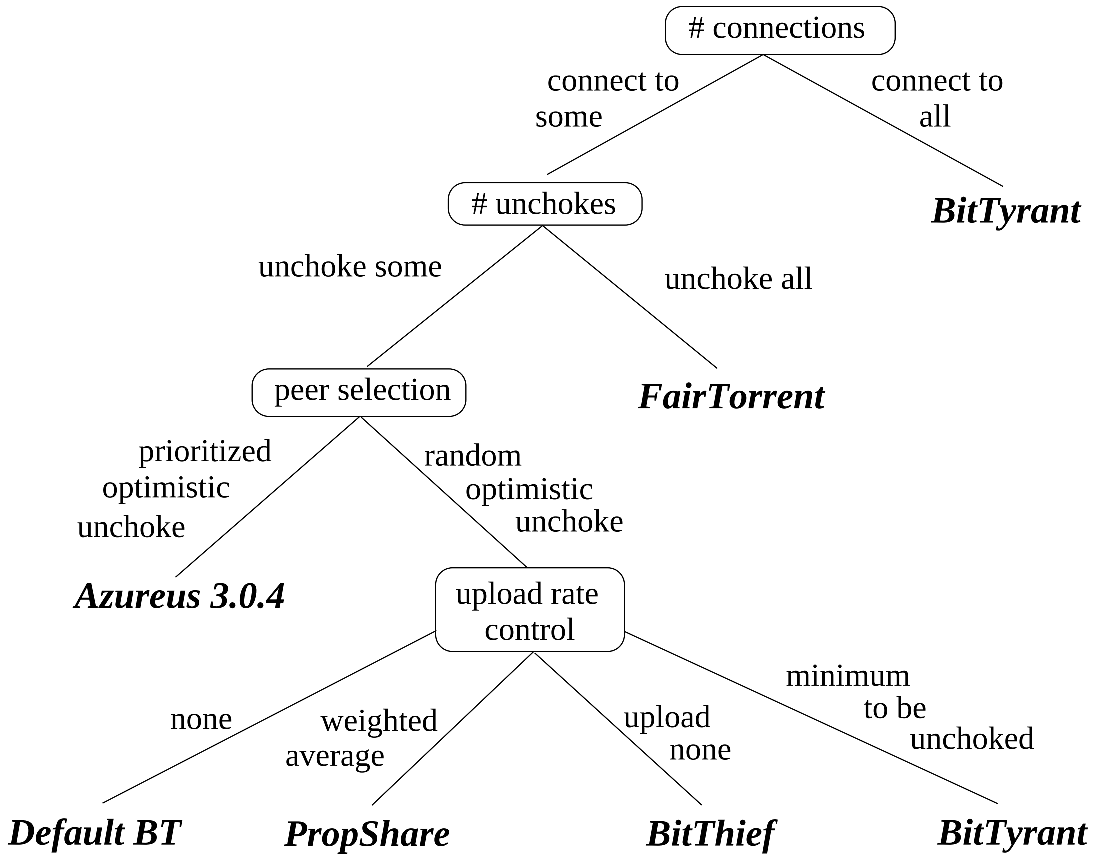


Figure 1.1: Taxonomy of BT variants.

like unchoking algorithm that was based on deficit to allow us to compare the performance of different algorithms for peer selection strategy.

We conducted experiments on PlanetLab using 100 nodes and 3 servers. We chose a wide range of upload capacity for our nodes in order to mimic the heterogeneous environment in real world. We looked into the possible options for each decision by gathering from previous works and our own proposed ideas. We collected logs from each node of each experiment and wrote scripts to process them to give us data we like to analyze. We plotted various parameters, like upload rate, client matching, utilization, etc. to help to visualize the interval mechanics of each option and compare their differences in term of fairness and performance. We investigated fairness and performance at both the systematic and at the individual level. Though we realized that some of the protocol decisions are related to one another, we try to separate them as much as possible so that we can analyze and study them individually to give us some useful insight.

1.2 Contributions

By systematically studying the differences between the various BT variants with our taxonomy-based approach, this dissertation makes the following contributions:

1. **Detailed Investigation of Protocol Design Space**. Our detailed and systematic exploration of the design space for the BT protocol reveals more design knobs than those suggested by Fan et al. [6], including different peer selection strategies and data upload control. In particular, we show that the peer selection can have significant impact on performance and fairness.

2. **Design Principles**. From our experience working with the various BT variants, we also articulate two key principles that we found are important to achieve good performance:

    - Keep promises, i.e. requests should be serviced promptly;
    - Keep the neighbour information up-to-date.

1.3 Report Organization

The rest of this dissertation is organized as follows: in Chapter 2, we provide an overview of the related work in the literature. In Chapter 3, we describe each level of taxonomy framework along with an associated measurement study. In Chapter 4, we present the key principles along and investigate how they can affect practical performance. Finally, we discuss future work and conclude in Chapter 5.

Chapter 2

Related Work

In this chapter, we first present a general overview of previous studies that reveal some of its key vulnerabilities of the BitTorrent protocol. Next, we describe several prominent strategic BT clients in chronological order. Finally, we highlight some studies which focus on a high-level understanding of the BT protocol design space.

2.1 Analysis, Simulation and Measurement Studies

There are a large number of analysis, simulation and measurement studies on BT performance in the literature. Legout et al. [10] claimed that rarest first and choke algorithm is enough to encourage reciprocation and prevent free-ridiing and later showed experimentally that clustering and good sharing incentive in BT systems [9]. The inherent weaknesses of the BT protocol has also been extensively studied [17, 7, 2, 12].

Thommes et al. found that peer selection and unchoking techniques in default BT implementation can induce substantial unfairness and proposed the use of a conditional optimistic unchoke to reduce the altruism introduced in unnecessarily optimistic unchoke [17]. They also suggested multiple connection chokes and variable number of unchokes to allow more flexibility on how many peers to unchoke and who to unchoke in order to improve fairness.

Jun et al. modelled the incentives of BT as an iterated *Prisoner's Dilemma* problem and showed with PlanetLab experiments that free riders complete downloads as early as those who contributes to the swarm [7]. To address such unfairness, they proposed that a restriction be imposed on the differences of upload amount and download amount for each link to a certain bound at all times.

Bharambe et al. found that BT's rate-based Tit-For-Tat (TFT) policy can give rise to unfairness across nodes in term of total data served in heterogeneous environment [2]. They proposed a pairwise block-level TFT which reduces unfairness, which is essentially the equivalent to the scheme proposed by Jun et al. [7]. The resulting trade-off is a reduction in utilization, which is especially severe among faster peers. This is because the faster peers are more likely to stop uploading to its neighbours whenever the block-level TFT constraint is not satisfied.

Liogkas et al. studied the effect of *selfish* BT clients, which attempt to download more than their fair share [12]. They identified three exploits, downloading only from seeds, downloading only from fastest peers and advertising false pieces. Their experimental results showed that BT proved to be quite robust against these exploits. However, the paper only studied each exploit individually, therefore the effect of benefits may be greater if all exploits are employed at the same time.

2.2 Strategic BT Clients

Many different BT clients have been invented to exploit or fix the various strategic vulnerabilities. BitThief [13] is a free-riding BT client that attempts to download data without contributing to the swarm by uploading data at all. BitThief tries to establish much more connections than the official client which allows it to obtain data from more seeders and get more optimistic unchokes from leechers. BitThief can achieve a high download rate, which showed that the basic piece exchange mechanism is ineffective at restraining free-riding peers.

Piatek et al. studied three different instances of altruism in BT-like protocols, namely the matching period, regular unchokes and optimistic unchokes [14]. To take advantage of the altruism, they propose a BT variant called *BitTyrant* that uses greedy peer set size (i.e. number of connections) which was proposed in BitThief [13] and greedy uplink allocation. Instead of treating unchoked peers equally, by not limiting on how much data can be uploaded to unchoked peers, BitTyrant attempts to upload only the minimum amount of data to each unchoked peer so as to secure and maintain the peer's reciprocation. In other words, the BitTyrant client seeks to maximize the total data download rate by actively managing the data uploaded to each peer. Carra et al. subsequently showed that the performance gain of BitTorrent over BT is due to the increased number of connections established by BitTyrant peers, rather than to the alleged active upload management [3]. However, this study was limited to simulation. In our work, we verified that the performance of BitTyrant is not as good as that claimed in the original BitTyrant paper [14] through experiments on PlanetLab.

Laoutaris et al. developed an uplink allocation algorithm that can shorten the download time by improving uplink utilization by dynamically managing the number of unchokes in real-time [8]. While keeping the upload capacity of the peer is fully utilized, they try to minimize the number of unchokes by uploading to the nodes with high upload capacity and low availability of pieces. This minimizes the risk of under-utilization of neighbours. However, since Laoutaris et al.'s protocol requires the peers to be cooperative, their scheme may not be realistic in a real-world scenario.

PropShare was proposed to address the loopholes in original BT algorithm which were exploited by BitThief and BitTyrant [11]. PropShare controls the rate of data upload by assigning each peer with an upload limit equal to the weighted average of the data received from the previous few rounds. Levin et al. showed that PropShare is Sybil-proof and collusion-resistant. However, the PropShare client needs to know its initially available upload capacity and only thereafter can it allocate a preset upload quota for each connection. Furthermore, the upload quota of each connection may not be fully utilized, which would result in wasted bandwidth. Nevertheless, PropShare outperforms BitTyrant when they are in the same swarm and BitTyrant cannot game PropShare. This is because PropShare clients do not use any upload threshold to decide who to unchoke, so there is no way for BitTyrant to determine what minimum value to upload in order to win a bid for reciprocation.

FairTorrent [16] is an innovative algorithm similar to PropShare that tries to address the problem of unfairness in original BT protocol without the need of neighbours' bandwidth estimation, risk of under-utilization and complicated parameter tuning in previous attempts by other works. Basically it does not choke any connections, but instead prioritizes uploads according to difference of number of bytes uploaded and downloaded from any peer, which is called *deficit*. The general idea is that the request from the peer which has the least deficit will be served first. This approach can achieve fairness naturally, however we will show in Section 4.1 that it can result in starvation.

2.3 BT Protocol Design Space

Fan et al. proposed a mathematical framework to study the fairness and performance of a P2P file sharing network [6]. They showed that there is a fundamental trade-off between performance and fairness. However, they only investigated performance and fairness from a theoretical perspective, and the actual algorithm for various BT-variants are not fully explored. For example, the paper assumes that each peer divides its uploading capacity equally among its neighbours. This is certainly not the case for BitTyrant, PropShare and FairTorrent. The paper presents only one design knob to tune fairness and performance based on original BT, which is by tuning number of regular unchokes and optimistic unchokes. In a practical BT implementation, the design knobs are certainly more complicated that this. Furthermore, Fan et al. did not seem to understand original purpose of optimistic unchokes. While an optimistic unchoke is altruistic since it will give to others first, its purpose is to explore the available peers to identify those that can reciprocate at a faster rate than current set of peers that are unchoked by regular unchokes. Optimistic unchokes are therefore not altruistic by design, but rather, the altruism is a side-effect. Therefore, the scenario where all the peers use only optimistic unchokes only to serve other people is not realistic in an actual real-world environment.

Xia et al. surveyed existing BT performance studies by adopting some general approaches in categorizing existing works and summarizing the design issues, their effectiveness and possible improvements [18]. Their survey includes works from analysis, measurement and simulation studies. However, Xia et al. categorized all design issues under either piece exchange and overlay topology which is unnecessarily broad. There is no apparent relationship between the two categories. In contrast, our work considers four factors that correspond directly to the BT protocol implementation, to systematically organize the design issues in a step-by-step manner, which we believe aids in our understanding and appreciation of the mechanics involved in the BT protocol and facilitates future design of new BT-related protocol. In addition, some of the claims summarized by Xia et al.'s survey paper are mutually contradictory and the authors made no attempt to verify the correctness of the claims. Furthermore, there is no clear focus of the paper, so the issues covered are much broader and the resulting discussions on each issue are inevitably very brief. In our work, we focus mainly on performance and matching among peers, which allows us to focus on fewer issues but in the process, investigate each issue in greater depth.

Chapter 3

Investigating the Protocol Design Space

In this chapter, we first describe our framework for the proposed taxonomy of protocols. We make the following assumptions in our discussion:

- The bandwidth bottleneck is in the uplink instead of the downlink at the nodes. This assumption is consistent with previous work [6].
- Peers will attempt to fully utilize the upload capacity as long as they can achieve good download rate.

Next, we investigate how the following parameters affect BT performance by running experiments on the PlanetLab testbed [15]:

- Number of connections
- Number of Unchokes
- Peer Selection Strategy
- Uplink Bandwith Allocation

3.1 Overview of BT-like Protocols

In this section, we give a brief introduction to BT protocol and explain the terms that are used in this dissertation.

A P2P file sharing network is formed by peers that want to download and/or upload a common file. The file is divided into fixed size *pieces* (typically 256 KB each), and each piece is further divided into sub-pieces which is called *blocks*, typically of 16 KB in size. The peers usually simultaneously download and upload blocks of the file from one another. The peers that have the complete file are called *seeds* and they effectively act as servers by uploading pieces to other peers.

When a peer joins a BT-based file sharing network, it obtains a list of peers from the tracker and connects to some of them. The peers exchange their *bitfield*, a bit map that records what file pieces each peer has. Based on the bitfield information, the peers can request missing pieces from other peers. *Choking* is the mechanism used to limit the number of simultaneous upload. By *unchoking* a remote peer, the local peer informs the the remote peer that it can now request pieces from it and serves the requests accordingly. The set of unchoked peers can be divided into *regular unchoke* peers and *optimistic unchoke* peers. Nodes record how much data they download from each peer every ten seconds, which we refer to as a *time interval*. Regular unchoke peers are chosen from the remote peers that upload the most data blocks to the local peer during the latest time interval according to the original protocol specification. *Optimistic unchoke* peers usually chosen randomly by a node in an attempt to find remote peers that can upload data to it at a faster rate than its current set of unchoked peers. Seeds and the optimistic unchoke help to bootstrap new peers without any file blocks to exchange with others.

There are basically two major strategies involved in the BT protocol, namely *peer selection strategy* and *piece selection strategy*.

The peer selection strategy refers to how a node decides on which peers to unchoke. In the BT protocol, the owner of the data decides which peers to unchoke (upload) and will upload blocks according to the requests received from the peers, while the unchoked peer only decides what piece to request. The goal of peer selection is (i) to efficiently utilize available upload capacity and (ii) to obtain maximum reciprocation from other peers. Hence, a node needs to pick enough peers to fully utilize its upload capacity and also pick wisely in order to maximize reciprocation from the peers.

The decision on which peer to download data from is usually passive. In the original BT protocol, a peer can only request up to four pieces from neighbours when they are unchoke, so the peer does not really have a choice about where it wants to download data from. In fact, it need not. The more peers that unchoke a peer, the better off is its situation. Just like in real-life, a person needs not be concerned when there are many benevolent people around who want to share their wealth.

Hence, once a node is unchoked, the remaining question is: what piece(s) should it try to download. The de facto piece selection strategy in original BT protocol is Local Rarest First (LRF). Since piece requests are usually pipelined, two requests are often sent initially. More requests can be sent later if the upload rate is found to be high.

3.2 Experimental Setup

To understand how various parameters affect the performance of the BT algorithm, We conducted measurements on PlanetLab [4, 15] with BT, Azureus and FairTorrent. We used Azureus version 3.0.4 as the BT client, but we modified the Azureus client to make it conform to original BT protocol as much as possible. For FairTorrent, we used the implementation provided by Sherman et al. [16]. In all our experiments, the size of the file to be downloaded is 100 MB, which is divided into blocks of 16 kB with 16 blocks forming a piece. In each experiment, unless specified explicitly, we used 100 nodes to simultaneously join the system and start downloading the file from the seed. Peer bandwidth are set to be heterogeneous, we adopt a uniform distribution, with the same number of peers having bandwidth 50KB/s, 75KB/s, 100KB/s, 125KB/s and 150 KB/s. This allows us to study the basic performance of BT clients in a heterogeneous swarm which serves as a good starting point for study of other more complicated distributions in future work. For most experiments, we conduct two variants: a *non-seeding* round, where the peers will leave after completion of download, and a *seeding* round, where the peers will stay and become seeds after completion of download.

Choice of the Upload Bandwidth for Server: Before presenting the results from our experiments, we shall explain the methodology used to choose an appropriate upload bandwidth for the server. In Figure 3.1, we plot the time taken for the fastest client to complete its download and also the time taken for the initial seed to give out every single block of the downloaded file. It is clear that when the server bandwidth is less than 175KB/s, the time required by the server in issuing out all the fresh blocks imposes a lower bound on the finish time of the fastest client. As the server bandwidth increases, the finish time of the client is likely less affected by the server capacity but more by the bandwidth distribution of peers in the system. We observed that though unique pieces finish time constantly decreases as server upload bandwidth increases, the best client finish time no longer improves with increasing server capacity when server bandwidth exceeds 270KB/s. Given this observation, we used 300KB/s as our server bandwidth for all our experiments.

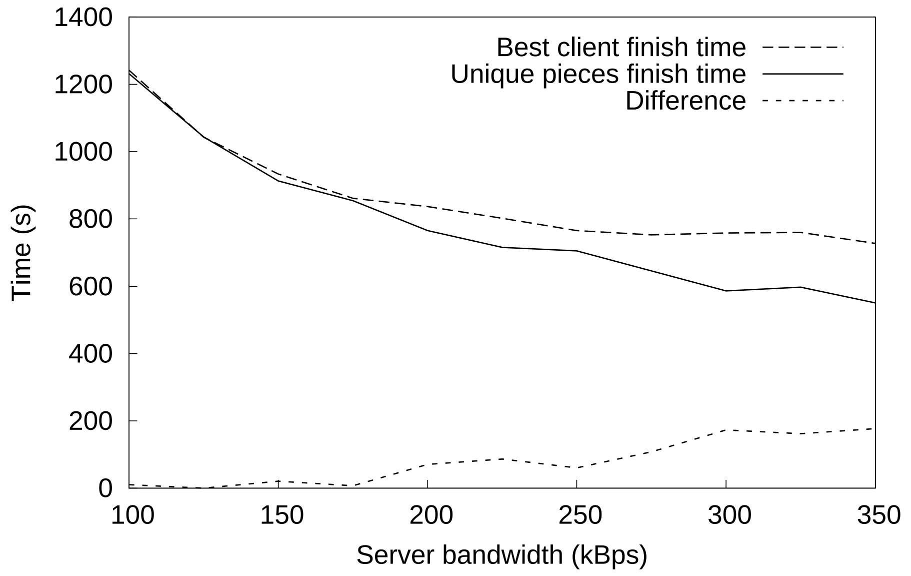


Figure 3.1: Plot of finish times against server upload bandwidth. Best client finish time is the download completion time of the fastest client in the system. Unique pieces finish time is the time needed by the server to issue out all pieces of the downloaded file at least once to some peer in the system.

3.3 Number of Connections

Peer set size is defined as number of connections that a peer maintains in the official BT protocol documentation. Maintaining connections with remote peers serves two purposes. The first is to exchange useful information regarding current pieces in possession with one another through bitfield and “have” messages. This allows a peer to calculate the availability of each piece and request local-rarest-first piece from other peers. The second is that from the peer set, a node can try to find matching peers and unchoke them. If the peer set size is too small, there may not be enough peers of compatible upload bandwidth within the group and the peer may not be able to find matching ones and will have to work with mismatched peers. Figure 3.2 shows that the average download time is roughly constant when number of connections is more than or equal to 30. We plot the upload utilization for different numbers of connections for seeding case in Figure 3.3. It shows that a small peer set

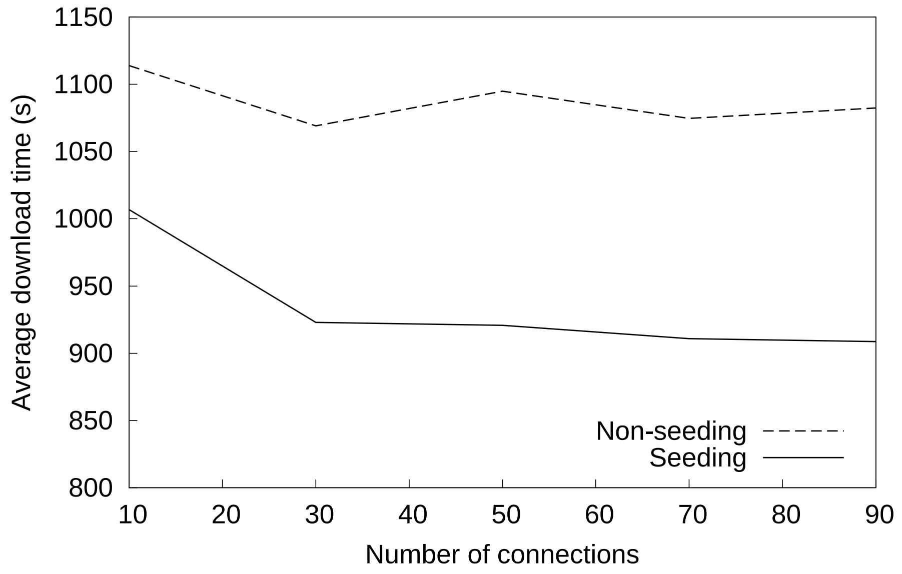


Figure 3.2: Average download time of BT peers when varying the number of connections.

size (as small as ten) can cause peers to become uninteresting to other peers and consequently result in a drop in the upload utilization. Therefore, we conclude that the local-rarest-first principle is effective in maintaining high availability of the local peers to others, and allow BT to utilize upload capacity efficiently.

In Figure 3.4, we plot the proportion of peer-bandwidth matching for all the peers' regular unchokes over time. If a node and its unchoked peer have the same upload capacity, we consider them to be *exactly matched*. If the absolute difference of node's upload capacity and its unchoked peer differs by no more than 25KB/s, we consider them to be *roughly matched*. We plot the graph only for experiment running time up to 700 s because after this time, some peers will complete their download and start leaving the system and this adversely affects the matching among peers of similar bandwidths.

Figure 3.4a shows that for smallest peer set size (i.e. ten), the percentage of exactly matched regular unchokes only increases slightly initially and stays constant for the rest of time. It is because the peer set size is too small, and

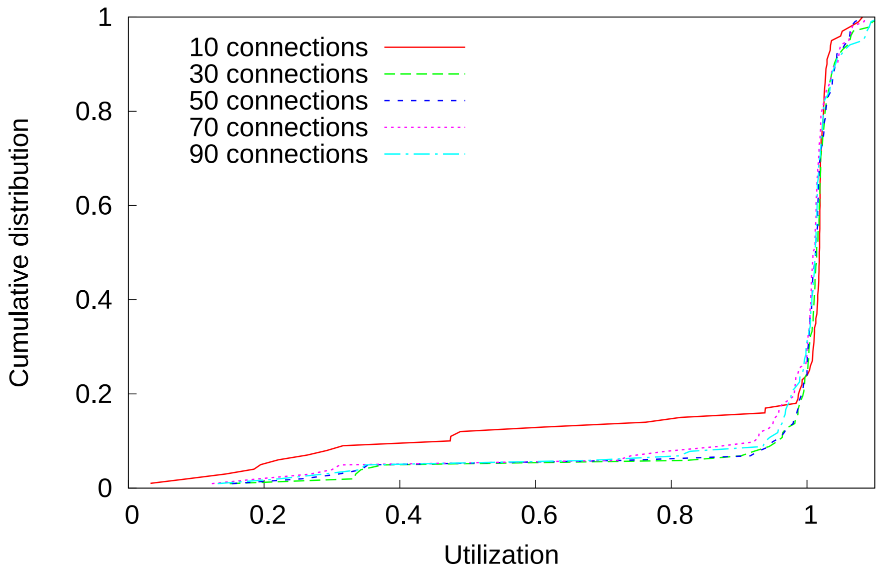
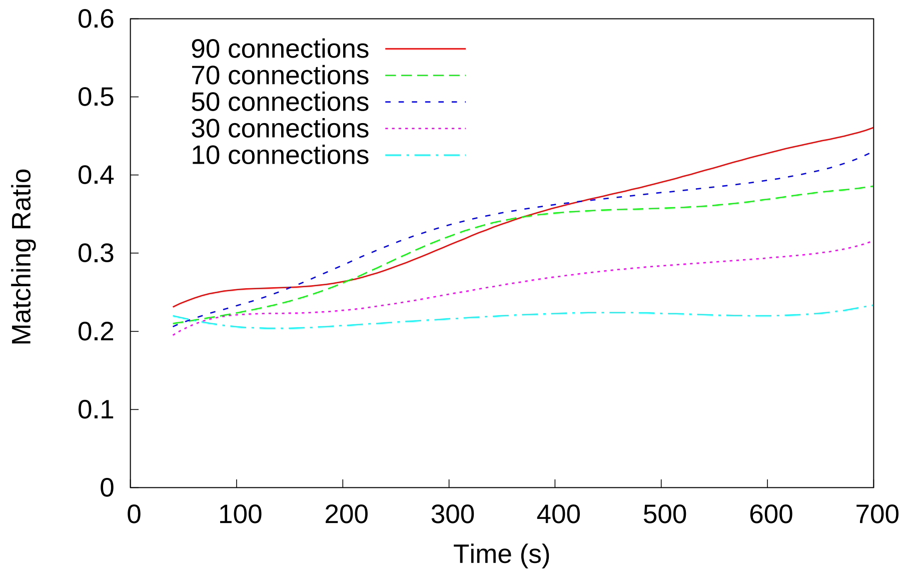
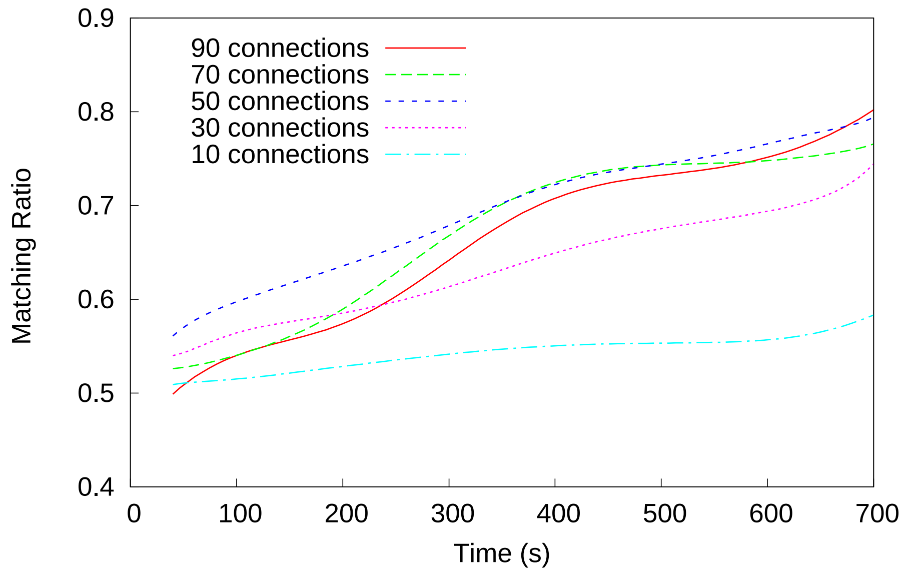


Figure 3.3: Comparison of upload bandwidth utilization for different numbers of connections.

there are limited neighbours for the local peers to explore for better matching, so after a short while, the local peer would have found the best ones in its peer set and continues to unchoke the same set of peers for the rest of time. With more connections, the matching percentage generally increases over time, since nodes have access to a large set of peers and nodes will gradually find better peers over time. Since the bandwidth used in our experiments do not differ too much, it is expected some peers will be content to exchange file blocks with peers of similar bandwidths. For example, a peer with 50KB/s upload capacity may pair with another peer with 75KB/s and another peer with 100KB/s upload capacity might pair with one with 75KB/s or 125KB/s upload capacity. In Figure 3.4b, we see that the results for roughly matched peers are similar to that for exact matching.



(a) Exactly matched.



(b) Roughly matched.

Figure 3.4: The percentage of matched regular unchokes over time for different peer set size.

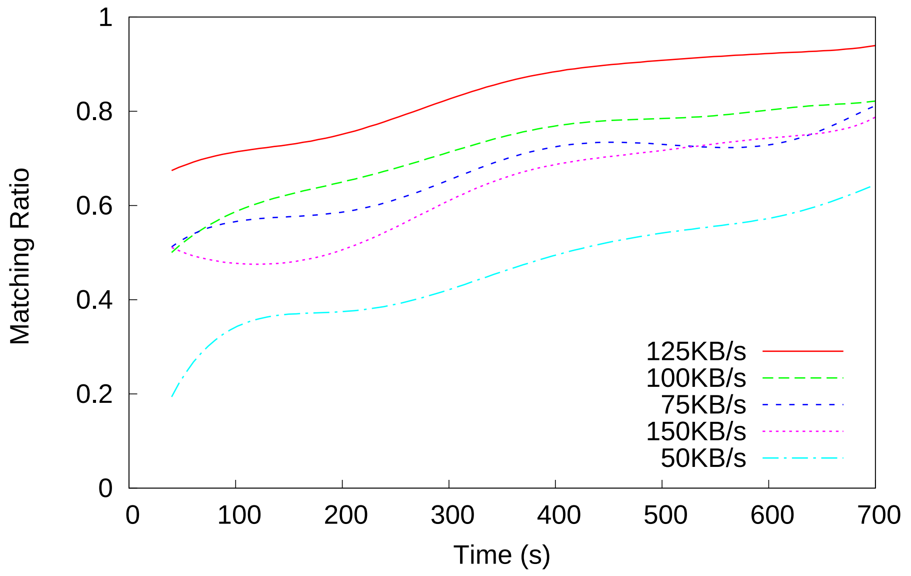


Figure 3.5: The percentage of roughly matching regular unchokes for each bandwidth groups over time for peer set size = 90.

Current BT protocol implementations usually use 50 connections as the default. For our experiment setup, this setting is sufficient to include enough peers of similar bandwidth in the peer set. However, due to the uncertainty and heterogeniety in real-life swarms, we would not claim that is the one-fit-all solution for all scenarios. We recommend that the download rate from each regularly unchoked peers be measured periodically. If there is no improvement over the matching after some time, we might want to increase the peer set size and allow more connections to be established. To investigate the behaviourial difference of peers for different bandwidth groups, we plot the matching ratios for the different groups in Figure 3.5, which shows the matching ratio for a peer set size of 90.

Since we are comparing rough matchly peers, those with bandwidth 75 KB/s, 100 KB/s, 125 KB/s can roughly match to peers of three bandwidth groups, but for peer of bandwidth 50 KB/s and 150 KB/s, it can only match to two bandwidth groups, so it is fair to compare peers of bandwidth 75 KB/s, 100 KB/s, 125 KB/s as one group, and peers of bandwidth 50 KB/s and

150 KB/s as another group. From the both groups, we observe that, in general, the peer matching ratio increases with higher bandwidth. Peers of bandwidth 125 KB/s constantly match better than those of 100KB/s, and peers of bandwidth 100 KB/s than those of 75 KB/s. It is also true for 150 KB/s and 50 KB/s. This is due to the fact that the faster peers will gradually match among themselves first, and slowly give up the slower peers it previously matched. When the slower peer realize that it has been abandoned by the faster ones, will try to match itself with other slower ones, and then stabilize. This process will go on until most the peers' regularly unchoked ones stabilize. So the faster peers will stabilize first, then the slower ones, and in the end, the slowest ones.

While we might expect the the matching proportion for the different bandwidth groups to eventually converge to a similar value, we found that this is not true in practice, as shown in the Figure 3.5. The reason is that previously we only focussed on the regular unchokes. When a fast peer has found its matching peers and its regularly unchoked peers stabilize, it still uses optimistic unchoke to explore better ones. Doing so is disastrous to slower peers since they will very likely give up a currently better-matched peer in one of the regular unchoke slots and replace it with a faster peer. However, when after a while, the fast peer realizes that the slow peer cannot upload as much data and will chokes the slow peer and in order to try another peer. So the slow peer will be abandoned and it has to go and look for other peers to unchoke and upload to. Similarly, the slower peers are constantly offered optimistic unchokes by the fast peers, which may not seem like a bad thing, except that they may disrupt the existing stable matchings of the slow peers.

3.4 Number of Unchokes

The active set size is defined as the number of unchokes in official BT protocol documentation. In the original BT algorithm, it is a fixed value: four regular unchokes and one optimistic unchoke. In later versions, number of regular unchokes is changed to a dynamic value that equals to $\lfloor\sqrt{0.6 * C}\rfloor$ where C is the upload capacity and the number of optimistic unchokes is set to be two. Later, in Azureus, a popular BT variant implemented using Java, it is still a fixed number - three regular unchokes and one optimistic unchokes by default. We expect that a higher upload capacity will require more unchoked peers in order to fully utilize the available capacity. In BitTyrant, it is a dynamic number based on the some calculation of the upload/download ratio of the neighbours and the local peer's own upload capacity. PropShare unchokes all the neighbors which has been uploaded some data to the local peer during the last four rounds and also some peers who did not upload to the local peer recently in order to know new peers.

Therefore, there are basically three classes of unchoke strategies: (i) a fixed number, (ii) a dynamic number purely based on the local peer's upload capacity, and (iii) a dynamic number based on the remote peers who uploaded to the local peer in the recent past (c.f. PropShare) or the ratio of the upload and download rate of each remote peer (c.f. BitTyrant). Due to the complexity of last category (since both PropShare and BitTyrant also involve data capping at the same time), we only study the effect of first two categories in this section.

We ran multiple experiments by varying the fixed number of unchokes for all nodes from 4 to 40 with step of 3 for both seeding and non-seeding cases. We plot in Figure 3.6 the average download time of all nodes in each experiment when number of unchokes changes. We found that, in general, when there are more uploads, the average download time increases. We believe it is

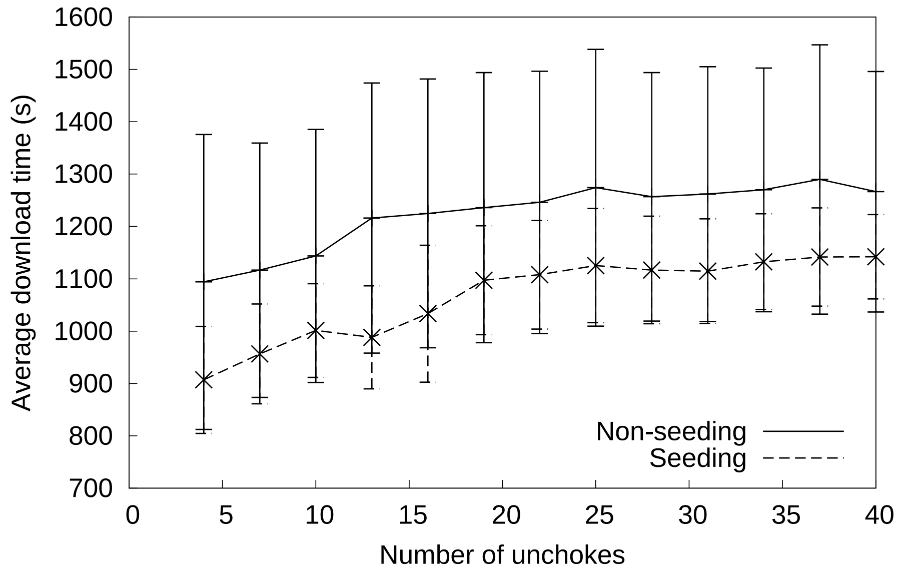


Figure 3.6: Average download time of BT peers when varying the fixed number of unchokes from 4 to 40 with step of 3. Error bars indicate the standard deviation. The client upload capacities are heterogeneous

because when there are too many unchokes, the pipe of each established TCP connection would be rather thin, and that would reduce the efficiency of data transmission for each connection. Furthermore, more unchokes would mean that each unchoked peer would get slower download rate, and therefore take a longer time to receive a complete piece from each peer. That may hurt the availability of pieces to others. In order to find the best number of unchokes for our distribution setup, we ran another set of experiments by varying the fixed number of unchokes for all nodes from 1 to 10 with step of 1 for both seeding and non-seeding cases. We plot the result in Figure 3.7. It shows that the value is around 3 to 4 which is very close to the default unchokes (5) in original BT protocol.

We also ran experiments with the original BT algorithm (denoted with “BTold”) with a fixed number of unchokes (four in our case) along with the latest version of the BT algorithm with number of unchokes varying according to the upload capacity (denoted with “BTnew”) . We plot the total data

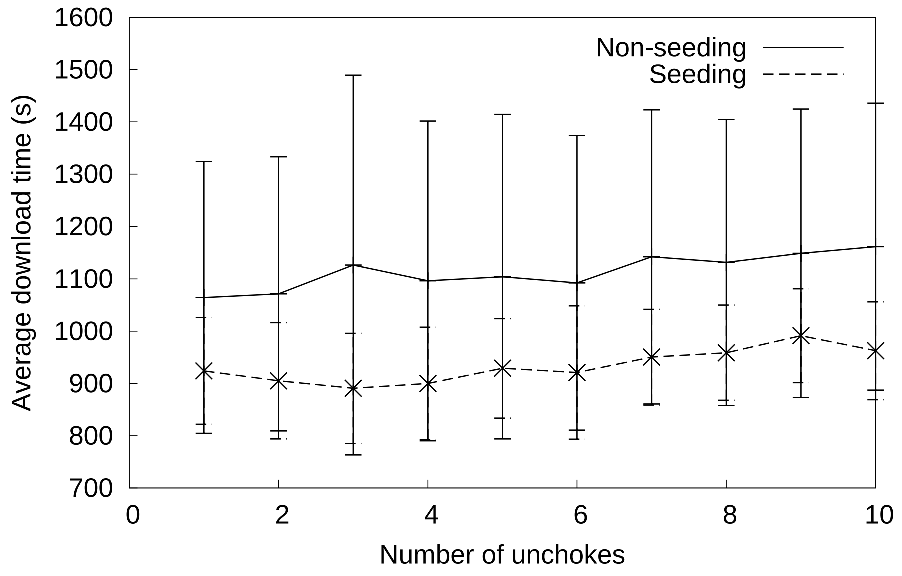
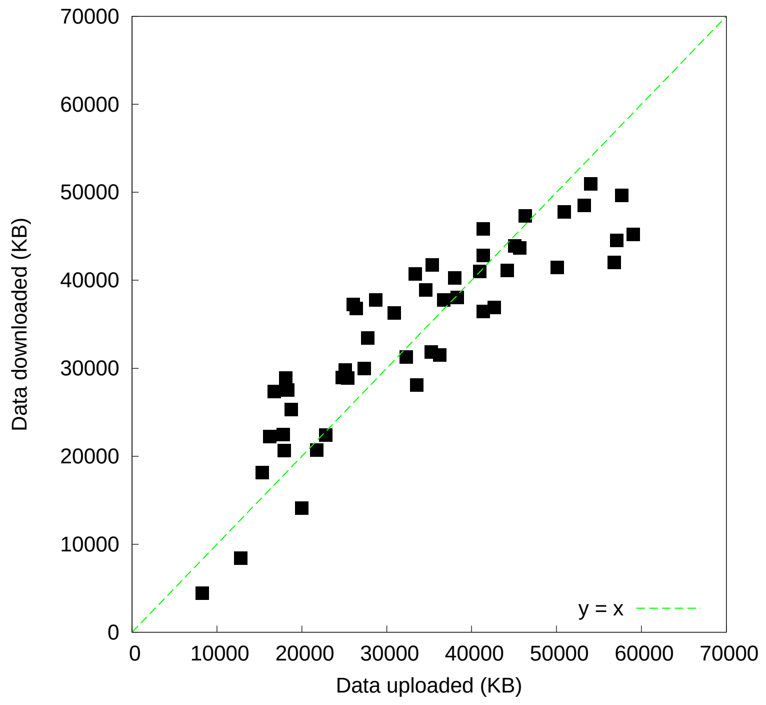
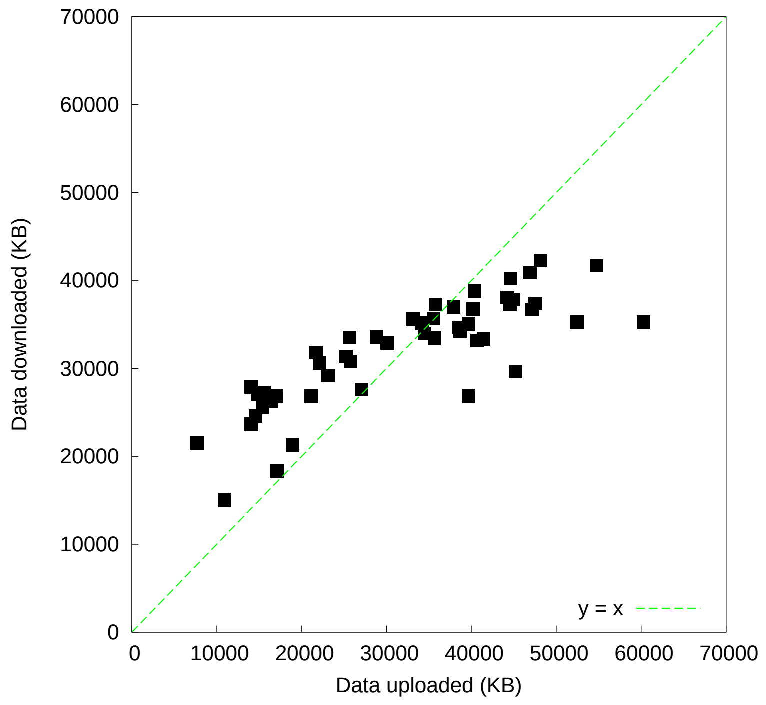


Figure 3.7: Average download time of BT peers when varying the fixed number of unchokes from 1 to 10, with step of 1. Error bars indicate the standard deviation.

uploaded and downloaded for each peer after 400 s in Figure 3.8. We pick the time at 400 s because at that point, all the peers are busy exchanging blocks with one another, yet none of them have completed the download. We found that for BTold, the peers had equitable upload download ratio for all bandwidth groups, while for BTnew, the faster peers (who uploaded more) contribute more than they downloaded, and the slower peers received more than they uploaded. Therefore, fixed number of unchokes for all peers actually achieves a better matching than a strategy where the number of unchokes varies based on upload capacity.



(a) All BTold clients



(b) All BTnew clients

Figure 3.8: The matching graph of upload amount vs download amount for all peers when time = 400s.

As mentioned, BTnew sets number of regular unchokes to a dynamic value that equals to $\lfloor\sqrt{0.6 * C}\rfloor$ where C is the upload capacity and the number of optimistic unchokes is set to be two. We define equal-split to be the average upload rate for a peer to each unchoked neighbours, which is the total upload capacity divided by the number of unchokes. Table 3.1 shows a comparison of number of unchokes and corresponding equal-split rates for BTold and BTnew clients respectively for different upload capacities from 50KB/s to 150KB/s. There are many more regular unchokes slots for fastest peers in BTnew (9) than in BTold (4), so it takes much longer time to find 9 peers with matching rate than in BTold case. Furthermore, since number of regular unchokes increases with upload capacity for BTnew protocol, the difference of equal-split rates for the faster peers and slower peers becomes smaller, so it becomes even harder to match among the same bandwidth groups.

Table 3.1: Equal-split rate of BTold vs BTnew

Upload Capacity (KB/s)	50	75	100	125	150
BTold (# of unchokes)	5	5	5	5	5
BTold (equal-split rate)	10	15	20	25	30
BTnew (# of unchokes)	7	8	9	10	11
BTnew (equal-split rate)	7.14	9.38	11.11	12.5	13.64

3.4.1 Number of Optimistic Unchokes

The purpose of optimistic unchoke is allow nodes to find better peers with which to exchange data. It is important to study how we should determine the number of optimistic unchokes and how to pick peers to be unchoked wisely.

We plot the average download time of BT peers when fixing the total unchokes to be 4 and 6, and varying the number of optimistic unchokes from 0 to the max number for non-seeding case in Figure 3.9. It shows that for

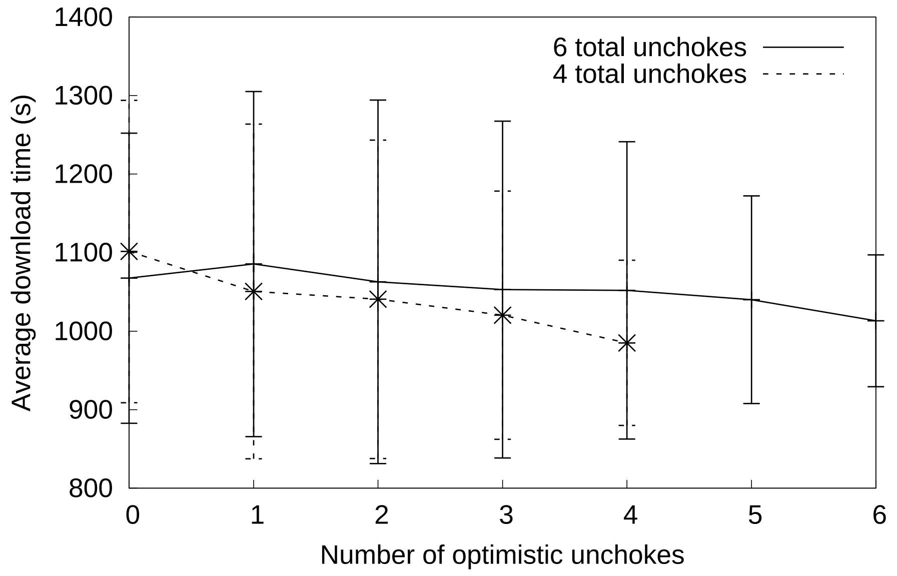
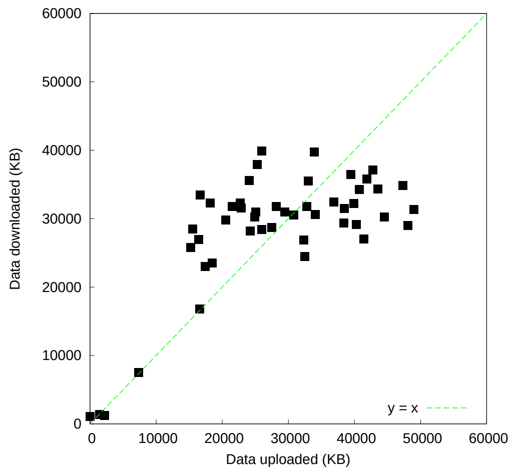


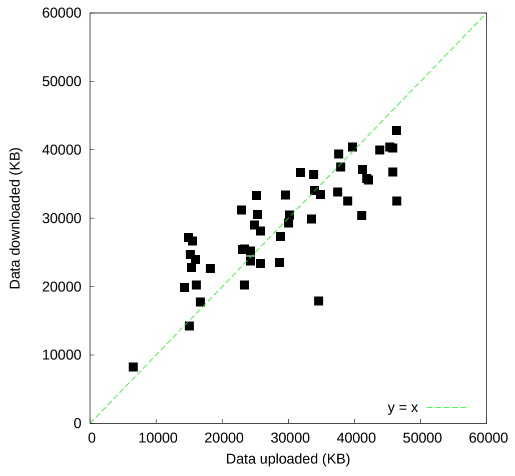
Figure 3.9: Average download time of BT peers when varying the number of optimistic unchokes for nonseeding case. Error bars indicate the standard deviation.

non-seeding case, when the number of optimistic unchokes increases, the average download time decreases. However, since the standard deviation becomes smaller, it suggests that the difference of average download time between faster nodes and slower nodes become smaller.

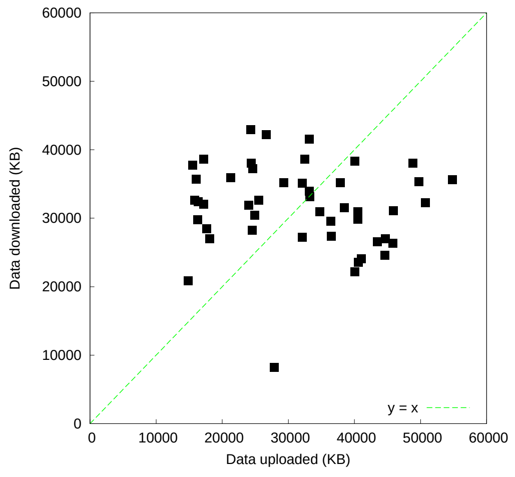
We plot in Figure 3.10 a comparison of total data downloaded and uploaded for each peer when all nodes run default BT clients with 4 total unchokes at time of 400 s. It shows that when the number of optimistic unchokes is two, we achieve the best matching among peers. The matching for 4 optimistic unchokes is worse than that with no optimistic unchoke since the former treats all peers equally, and the latter can at least choose the best ones among all the peers it exchanges data with initially. Though the latter has a very small set to choose from, it is still better than choosing randomly (four optimistic unchokes). So the reduction in average download time achieved in Figure 3.9 is due to the sacrifice of faster peers who upload to the other nodes without demanding reciprocation.



(a) Zero optimistic unchoke



(b) Two optimistic unchokes



(c) Four optimistic unchokes.

Figure 3.10: The matching graph of upload amount vs download amount for all peers when all nodes run BT clients when time = 400s.

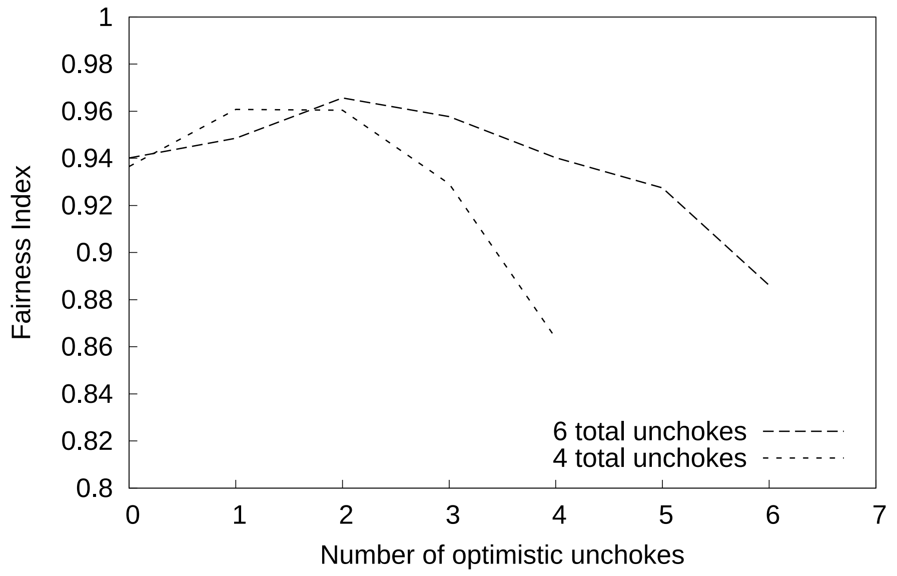


Figure 3.11: Average download time and fairness index of BT peers when varying the number of optimistic unchokes.

We adopt the fairness index proposed by Fan et al. [6] to quantify the fairness of our system and plot in Figure 3.11 the average download time and fairness index when the number of total unchokes is fixed at 4 and 6 respecitively, and the optimistic unchokes is varied. It shows that as the number of optimistic unchokes is increased, the fairness index increases until a maximum value, and then it decreases. This agrees with the simulation results of Fan et al. [6].

It shows the advantage and the drawback of the optimistic unchoke. The main purpose of an optimistic unchoke is to allow a node to find better peers. When it is zero, peers can choose from the peers they exchange data with initially (for a fixed number of unchoke N, they have information for N peers they upload to and average N peers they download from, and in total they have less or equal to 2N peers to choose from), so the fairness index is very low. When number of optimistic unchokes initially increases, the optimistic unchokes start allowing a node to better peers. Unfortunately, when it increases even more, we start seeing the drawback of the optimistic unchoke, which is its altruistic nature in uploading to others first. If number of optimistic unchokes occupies more than a certain proportion of the total unchokes, a peer will end up giving out too much upload capacity for nothing.

In addition, we also observe that when we have a larger number of total unchokes, we will need more optimistic unchokes to achieve better fairness. That is to be expected since with more unchokes, we need more time to find the matching peers to fill the unchoke slots, and more optimistic unchoke may help to achieve that faster. However, too many optimistic unchokes might cause the peers to be excessively altrustic and affect fairness. Our results suggest that as a good rule of thumb, the number of optimistic unchokes should be slightly less than half of the total unchokes (one for four unchokes, two for six unchokes).

Next, we plot in Figure 3.12 the average download time of BT peers when the total unchokes is fixed at 4 and 6, and vary the number of optimistic unchokes from 0 to the max number for seeding case. It shows when number of optimistic unchokes increases more than half of the total unchokes, the average download time starts to increase. The reason is that when there are more optimistic unchokes, the peers tend to be more altruistic and the faster peers will take longer to finish download and become seeds. Whenever a node finishes and becomes a seed, there are less peers to share the total upload capacity, so the average download rate received by each leecher will increase. So it is generally better to allow faster nodes to finish earlier and become seeds.

A simple example can help to better illustrate this concept. Assuming there are 100 fast nodes of upload capacity of 80 KB/s, 100 slow nodes of upload capacity of 20 KB/s and a server of upload capacity of 100 KB/s in the swarm. To make our analysis simpler, we ignore the contribution of the server in our calculation. If all peers use all unchokes for optimistic unchokes, then each

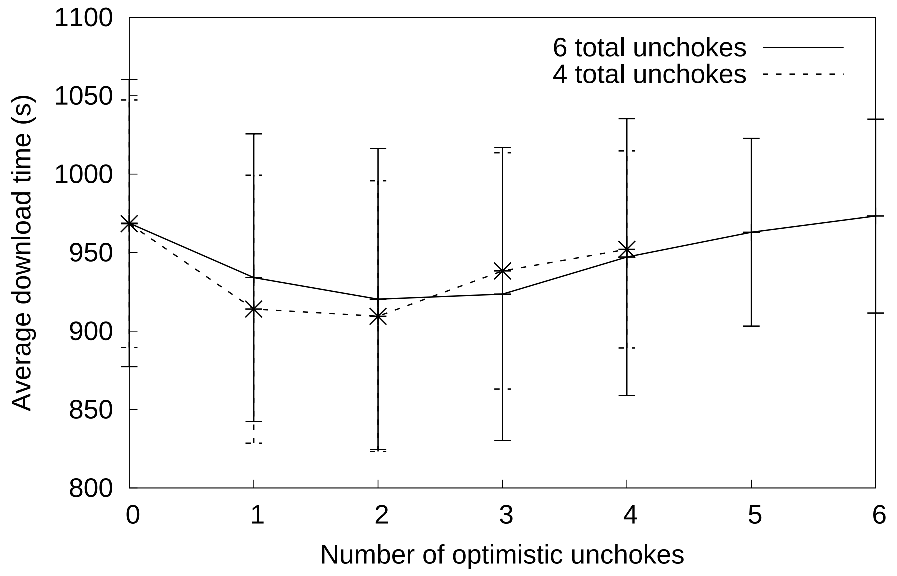


Figure 3.12: Average download time of BT peers when varying the number of optimistic unchokes for seeding case. Error bars indicate the standard deviation.

one will roughly get 50KB/s download rate. If we assume the file size to be 10000 KB, then it will take 200 s to download the file for each peer. However, if we allow fast nodes to exchange blocks among themselves through perfect matching, the fast nodes will roughly take 125 s to download the file and become seeds. Each slow peer would receive $20 \times 125 = 2500$ KB during the period through data exchange among themselves and have 7500 KB left to download for the file. The average download rate each slow peer now receives is 100KB/s since one fast node (a seed) will upload to one slow node on average, so in total it only takes $125 + 7500 / 100 = 200$ s to download the file for the slow nodes. So on average, the download time is only $(125 + 200)/2 = 162.5$ s which is much smaller than 200 s in our previous calculation. Therefore, it is beneficial to keep number of optimistic unchokes small from a system point of view.

3.5 Peer Selection Strategy

Peer selection strategy involves how to choose peers from the neighbours for optimistic unchokes and regular unchokes respectively.

3.5.1 Choice of Peers for Optimistic Unchokes

The original BT protocol chooses randomly among all the neighbours. Azureus implements a new strategy which gives higher priority to peers who have reciprocated in the past. It ranks all the peers based on deficits (data downloaded to the local peer - data uploaded by the local peer) in a descending order with the first peer being the one with the highest deficit. Then it generates a random floating point number x from 0 to 1, and calculates a position value using $\frac{1}{\frac{0.2}{x}+0.8} * peer\_size$. It uses the position value to pick the peer to unchoke from the ordered list. We plot the function in Figure 3.13, and it shows that the strategy favors the peers who are nearer to the end of the peer list (nearer to 1), and thus have lower deficits.

We plot in Figure 3.14 the percentage of exactly and roughly matched regular unchokes over time for the two peer selection strategies: (i) random optimistic unchokes and (ii) factor of reciprocation consideration. We found that when we consider factor of reciprocation in choosing peer for optimistic unchoke, the percent of matched regular unchokes is slightly higher for the duration of the download process. We also plot in Figure 3.15 the percentage of exactly matched optimistic unchokes over time for the two different strategies for peers with upload capacity of 100 KB/s and 150 KB/s.

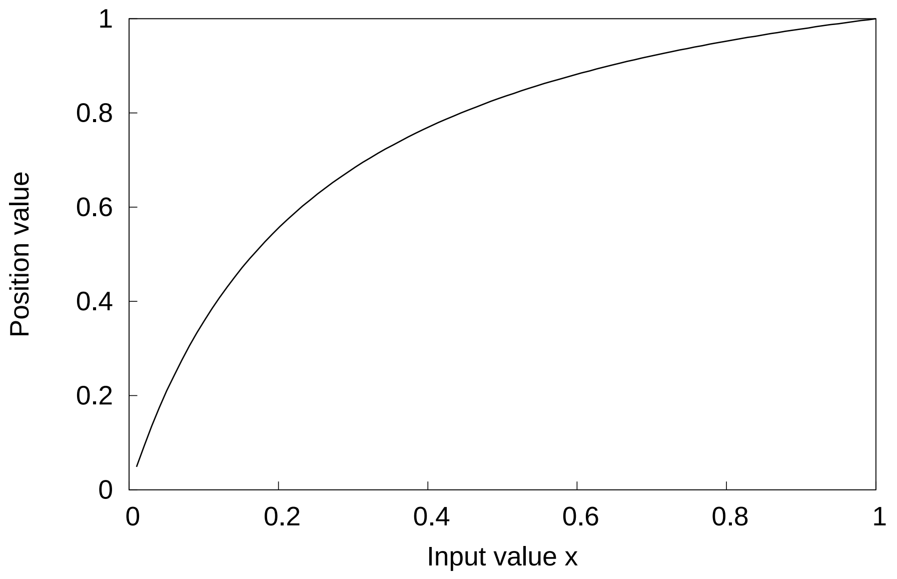


Figure 3.13: The function that Azureus uses to calculate and locate the peer(s) from the peer list ordered according to descending order of deficit for optimistic unchokes

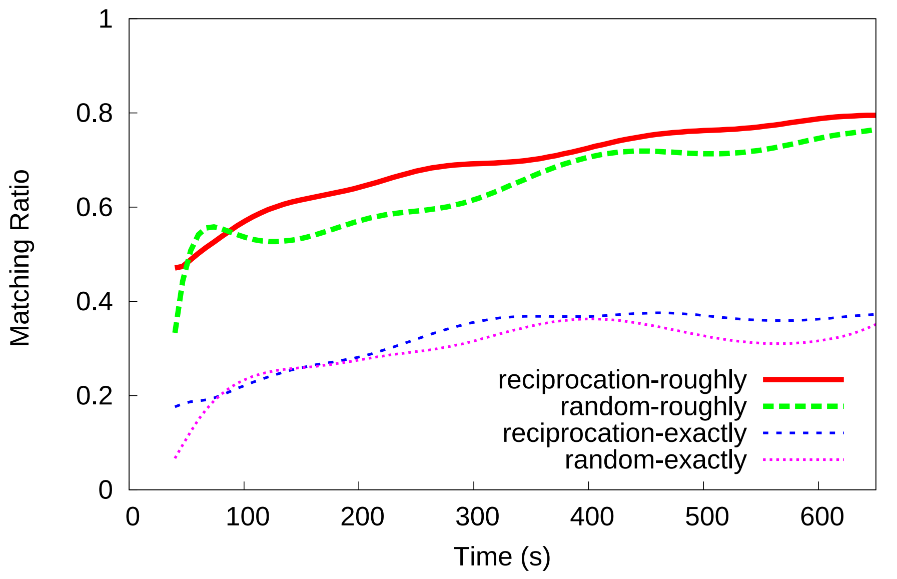


Figure 3.14: The percentage of exactly and roughly matched regular unchokes over time for random optimistic unchokes and factor of reciprocation consideration.

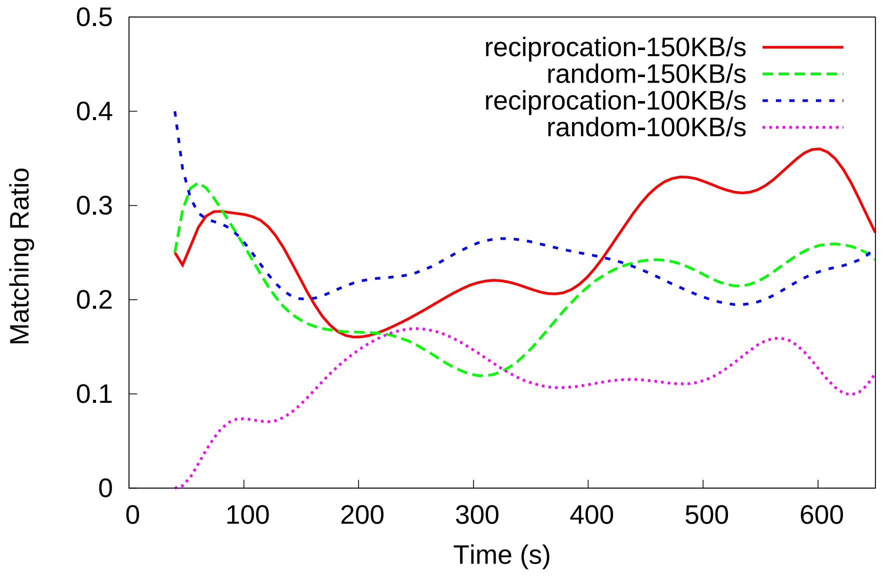


Figure 3.15: The percentage of exactly matched optimistic unchokes over time for random optimistic unchokes and factor of reciprocation consideration for peers with upload capacity of 100 KB/s and 150 KB/s.

We found that the high (150 KB/s) and middle (100 KB/s) upload capacity peers are more likely to pick peers from their own bandwidth groups if we use the factor of reciprocation peer selection strategy. Doing so is helpful because low bandwidth peers who would less likely be mistakenly unchoked by the higher bandwidth peers, and thus are more likely to stay matched with peers in its own group. We plot in Figure 3.16 the percentage of exactly matched regular unchokes over time for random optimistic unchokes and factor of reciprocation consideration for peers with upload capacity of 50 KB/s and found that slow peers (50 KB/s) are more likely to pick peers from its own groups for regular unchokes.

However, it does not tell the whole story. We plot the percentage of exactly matched optimistic unchokes over time for random optimistic unchokes and factor of reciprocation consideration for peers with upload capacity of 50 KB/s in Figure 3.17. It shows that this strategy does not prevent the slow peers from choosing faster peers for the optimistic unchokes. This is due to the nature of

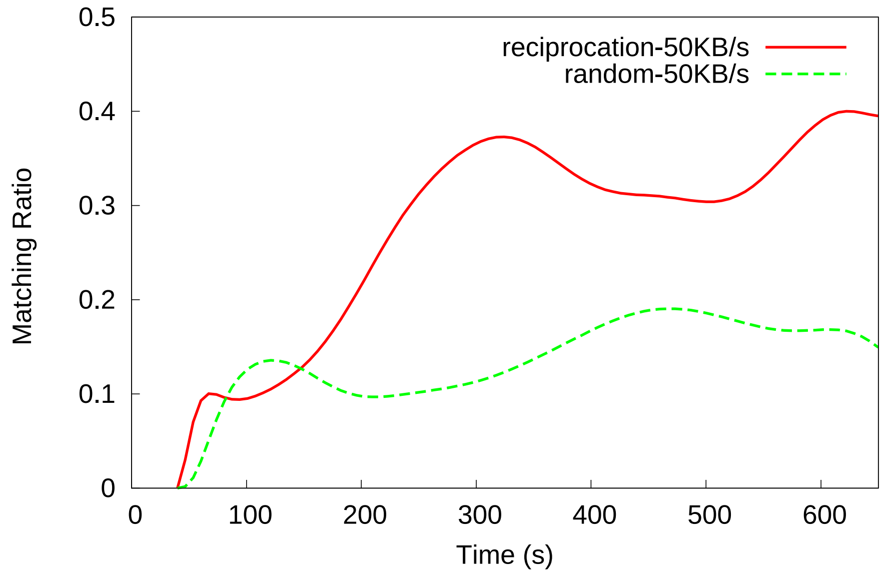


Figure 3.16: The percentage of exactly matched regular unchokes over time for random optimistic unchokes and factor of reciprocation consideration for peers with upload capacity of 50KB/s.

the strategy used, since a peer is more likely to choose peers who have lower deficit, a slow peer is more likely to choose faster ones who optimistically unchoke it and upload to it recently.

3.5.2 Choice of Peers for Regular Unchokes

In the original BT algorithm, peers for regular unchokes are chosen from the remote peers who can upload the most data to the local peer in the past period. BitTyrant changes it to download/upload ratio to maximize the return it can get from its upload. Some papers [2, 16] suggest using deficit (difference between download and upload amount) to find the peers the local peer owes the most and serve them. In our experiment, we compare selection based on download rate received and deficit. We ran experiments of all nodes running BT clients that use download rate and deficit respectively and calculated the average download time and fairness index for both seeding and non-seeding cases. The results in Table 3.2 shows that the average download time of

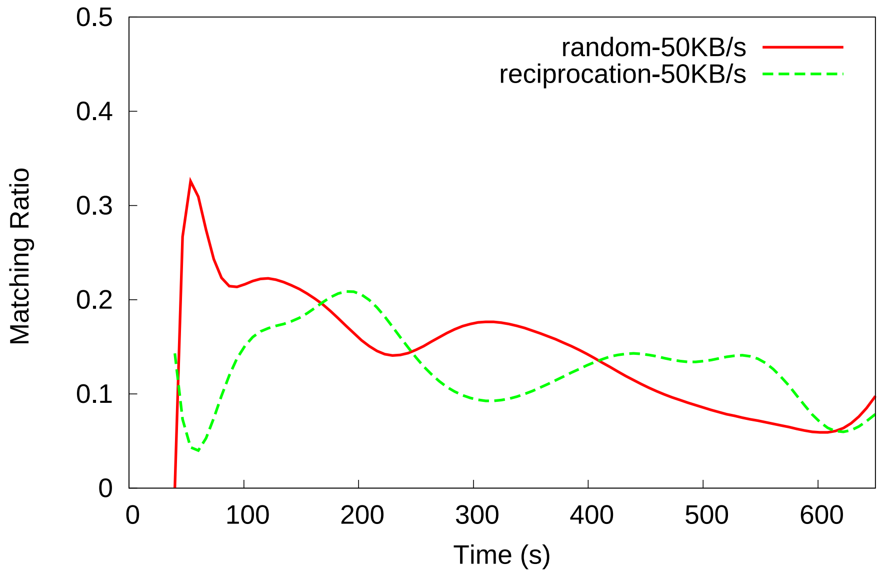
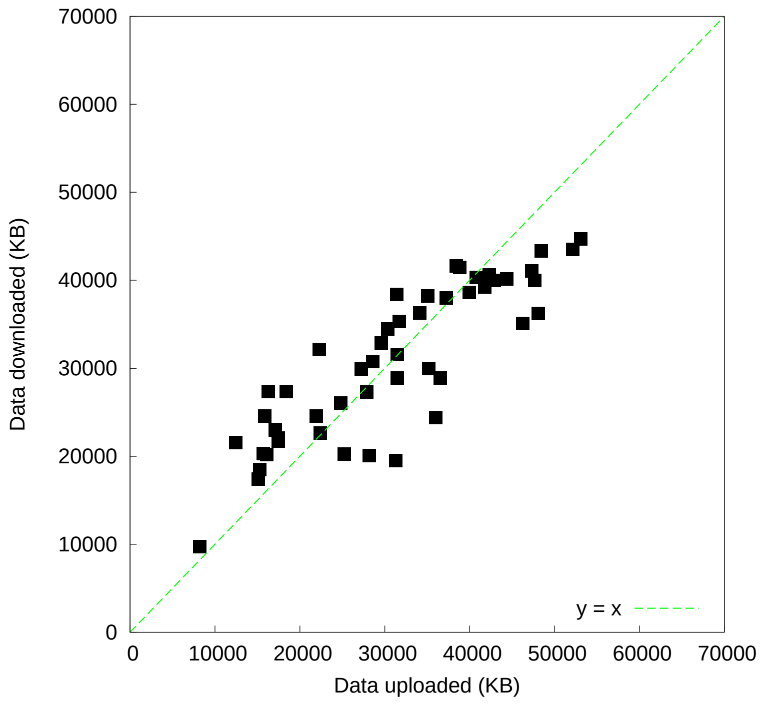


Figure 3.17: The percentage of exactly matched optimistic unchokes over time for random optimistic unchokes and factor of reciprocation consideration for peers with upload capacity of 50 KB/s.

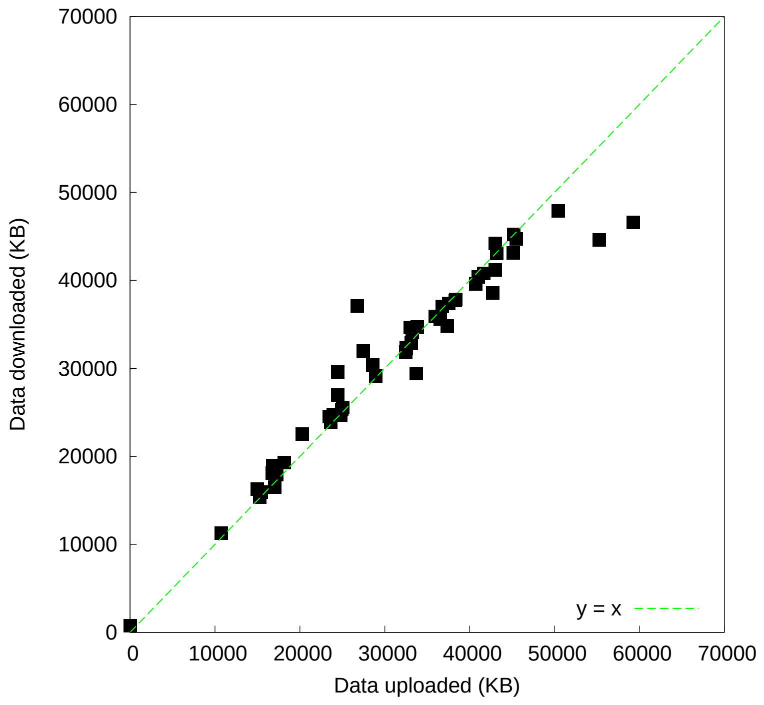
Table 3.2: regular unchoke strategy and performance result

Selection strategy	**Average download time (s)**	**Fairness index**
Download rate (Nonseeding)	1067.8	0.948
Deficit (Nonseeding)	1088.9	0.973
Download rate (Seeding)	936.7	0.938
Deficit (Seeding)	917.9	0.976

both strategies does not differ much, however, the peer selection based on deficit achieves higher fairness index. This is to be expectated because the deficit-based strategy strives to unchoke the peers with the least deficit (which the local peer owes the most), it aims to achieve better fairness. We plot in Figure 3.18 the comparison of the data downloaded and uploaded for all peers when time = 400 s. It shows that unchoking based on deficit achieves much better matching at the aggregate data level.



(a) Use of download rate for unchoking.



(b) Use of deficit for unchoking

Figure 3.18: The matching graph of upload amount vs download amount for all peers when time = 400s.

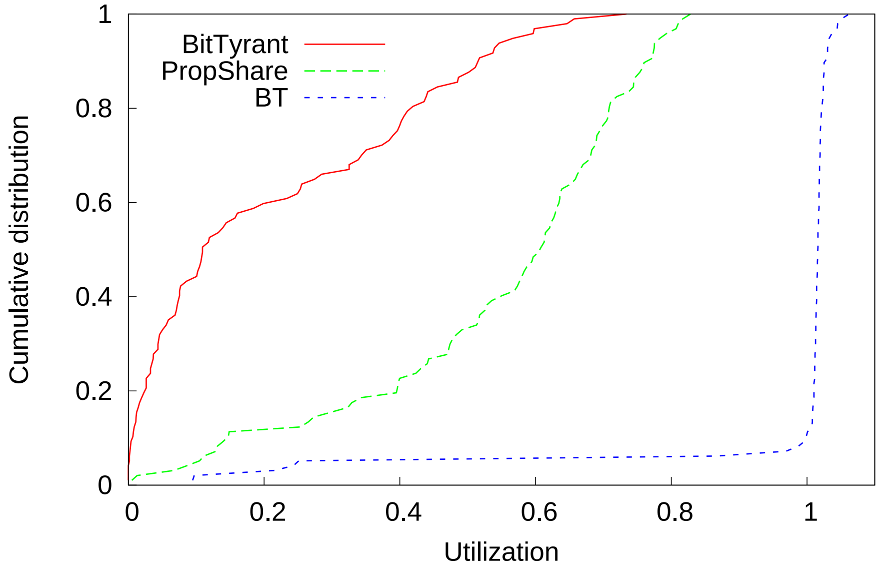


Figure 3.19: Comparison of upload bandwidth utilization among peers running BT, BitTyrant and PropShare.

3.6 Uplink Bandwidth Allocation

In the original BT protocol, no limit is imposed on data uploaded to each unchoked neighbour within each round. In fact, it is this vulnerability which BitTyrant [14] attempted to exploit. In BitTyrant, the upload contribution to each peer is adjusted to find the minimum upload contribution required for reciprocation. PropShare [11] proposes to limit the upload data of a client to different peers in order to ensure that the amount of data uploaded to each peer is proportional to the data received from it.

We ran experiments for peers running BT, BitTyrant and PropShare respectively in each round and plot the resulting distribution of the bandwidth utilization at a time after all peers have sufficient blocks to start exchanging pieces and before the first peer completely downs the file in Figure 3.19 and average utilization as compared with BT in Table 3.3. It is clear that limiting uplink allocation reduces efficiency of the utilization of the available upload bandwidth severely.

Table 3.3: Utilization of BitTyrant and PropShare

	Utilization compared to BT
BitTyrant	0.217
PropShare	0.559

3.7 Summary

In summary, the following are our key findings:

- The number of connections does not affect the performance beyond a certain threshold (around 30).
- The number of unchokes should be kept low, preferably around four. Adjusting the number based on upload capacity would weaken matching.
- The number of optimistic unchokes should be kept slightly less than half of the total unchokes. More optimistic unchokes would be more altruistic, thus reducing fairness; less unchokes may cause peers to take longer time to match with peers of similar bandwidth.
- For selection of peers for optimistic unchokes, it is useful to consider factor of reciprocation for relatively faster nodes. Slow nodes may indirectly benefit from it but do not need to adopt it themselves.
- For selection of peers for regular unchokes, using deficit may improve fairness with little impact on performance when compared with traditional method of using download rate.
- It is generally detrimental to limit upload bandwidth for each connection since that may reduce utilization since allocated bandwidths may not be used up by some connections.

Chapter 4

Design Principles

In a society, there are rules and regulations in place to ensure that all residents to live harmoniously together. In a P2P swarm, we also have similar principles that should be followed if all peers want to benefit mutually from the swarm. While studying the taxonomy along with various existing protocols, we came up with two key principles which are important for BT protocol design.

4.1 Keep Promise

In the default BT algorithm [5], Azureus [1], BitThief [13], BitTyrant [14] and PropShare [11], piece requests are serviced in a FIFO manner. When a request is received, it is expected to be served within a reasonable period of time. FairTorrent differs by ordering the requests according to the deficit (difference between upload amount and download amount) of the sender of the requests, and serving the requests from the peers with the lowest deficit first. This priority uploading scheme introduces *uncertainty* in the request serving process. Some requests will get delayed for a long time, leaving the sender of the requests with no idea of whether the requests will be serviced, and if so, when. Eventually the uploading peer may get snubbed by the requesting peer after 60 s, and the request may get time-out and cancelled by the requesting peer after 120 s if the requesting peer follows the default BT protocol specification. This priority scheduling of serving requests can often cause starvation, especially for slow peers who tend to have higher deficit from other faster peers' perspective.

We ran experiments of all nodes running FairTorrent [16] and Azureus [1] clients respectively with the same upload capacity of 128 KB/s and plot the number of CANCEL messages received for each 10 s interval for the duration of the experiments and the corresponding average upload rate of all peers in Figure 4.1. We found that all the FairTorrent peers experienced many CANCEL message as compared to that of all Azureus peers. The performance inevitably degrades when there are too many CANCEL messages as shown by a dip in average upload rate in the graph.

We assigned an ID in an increasing order of service time for the requests of FairTorrent and Azureus clients and plot the request service time for each request ID in Figure 4.2. We found that a large number of requests of FairTorrent clents experience a service time that is significantly larger than that of Azureus, which can cause uncertainty for requesting peers. From the summary of results in Table 4.1, we find that the average download time for Azureus is lower than that for FairTorrent. Therefore, it is important to keep promise by serving requests promptly.

Table 4.1: Comparison of experiment results for Azureus and FairTorrent

	Average download time (s)
Azureus	818.0
FairTorrent	976.6

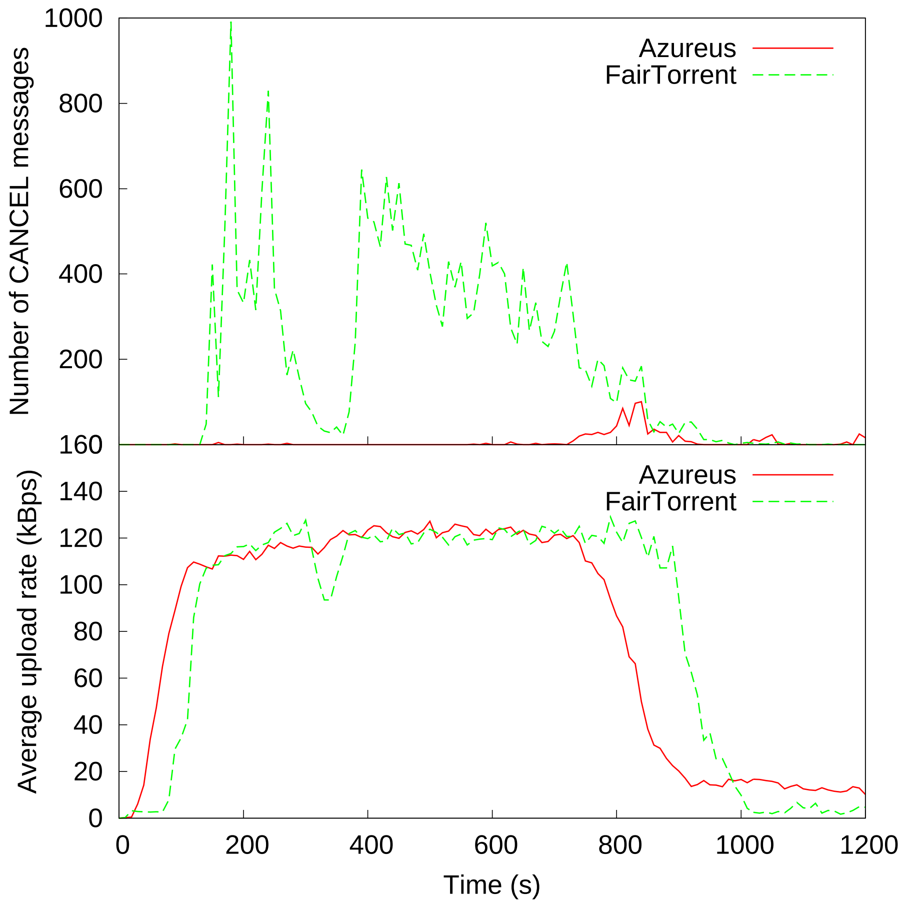


Figure 4.1: Number of CANCEL messages received for each 10 secs interval and the corresponding average upload rate of peers.

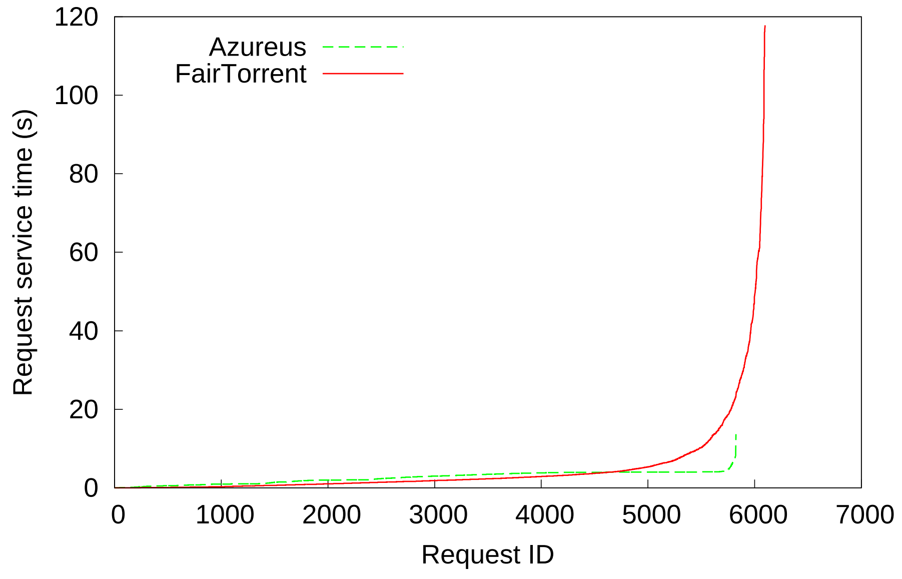


Figure 4.2: Time taken to serve each request. Requests are ordered according to service time.

4.2 Keep Neighbour Information Up-to-date

PropShare [11] clients withhold HAVE messages to prolong other peers' interest in the client. Azureus [1] clients also implement HAVE message aggregation, which accumulates and combines many HAVE messages into one message in order to save on bandwidth. We ran experiments of all Azureus nodes with HAVE message aggregation turn on and off and presented the result in Table 4.2. It shows that although average download time does not differ much, it takes a shorter time for the servers to send out all fresh pieces into the system when HAVE message aggregation is turned off. This is to be expected because when clients have more up-to-date piece information of their neighbours, they are less likely to request the pieces that their neighbours already have from the servers, so they are more likely to request fresh pieces which have not been sent out to any peer from the servers. This, if we have servers that are not well provisioned, it is important to keep neighbour information up-to-date in order to reduce the burden on the server side.

Table 4.2: Comparison of experiment results with HAVE aggregation turn on and off

	Average download time (s)	**Time taken for servers to send out all fresh pieces (s)**
On	836.5	632.5
Off	838.5	591.7

Chapter 5

Conclusion

In the dissertation, we propose a new taxonomy-based approach for analyzing the trade-offs that should be considered in the practical implementation of the BT protocol. We conducted a detailed investigation of protocol design space through PlanetLab experiments. Through the study, we come to realize that good matching is not easily achieved and maintained, careful analysis and implementation is required to achieve effective, efficient and stable matching on both the individual level and the system level.

Next, we articulate two key design principles that we gleaned from our experience working with various BT clients, namely keeping promise and keeping neighbours information up-to-date. BT-like P2P protocols are complicated systems since it involves interplay of various clients of different behaviour. However, we believe our work is helpful in guiding future BT protocol designers in implementing their clients to achieve good performance and matching while fostering a healthy P2P file sharing environment.

5.1 Future Work

For regular unchokes, we can look into whether it is better to vary the number over time or keep it fixed when upload capacity is known. We generally feel that from a system point of view, a fixed value would improve stability. But from an individual point of view, varying the number may help it to achieve better matching. As for the choice of peers, the original protocol favors peers of highest upload rate, but that may not be stable when peers have very different equal-split rates. A improved version may be to pick neighbours of similar equal-split rates.

For optimistic unchokes, we can study whether we should fix the number or allow it to vary depending on the circumstances. The original BT protocol fixes it to be one or two, but an improvement may be to use more optimistic unchokes at start-up phase, and reduce it when more matching is achieved. When reasonable matching is achieved for all regular unchokes, then optimistic unchoke could reduce to zero or switch to regular unchoke.

Bibliography

[1] Azureus p2p file sharing client. Website. http://www.vuze.com.

[2] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and improving a BitTorrent network's performance mechanisms. In *Proceedings of IEEE INFOCOM '06*, April 2006.

[3] D. Carra, G. Neglia, P. Michiardi, and F. Albanese. On the robustness of bittorrent swarms to greedy peers. *Parallel and Distributed Systems, IEEE Transactions on*, 22(12):2071–2078, 2011.

[4] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: An overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, July 2003.

[5] B. Cohen. Incentives build robustness in BitTorrent. In *Proceedings of the P2P Economics Workshop*, 2003.

[6] B. Fan, J. C. S. Lui, and D.-M. Chiu. The design trade-offs of BitTorrent-like file sharing protocols. *IEEE/ACM Transactions on Networks*, 17:365–376, April 2009.

[7] S. Jun and M. Ahamad. Incentives in bittorrent induce free riding. In *Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, pages 116–121. ACM, 2005.

[8] N. Laoutaris, D. Carra, and P. Michiardi. Uplink allocation beyond choke/unchoke: or how to divide and conquer best. In *Proceedings of the 2008 ACM CoNEXT Conference*, page 18. ACM, 2008.

[9] A. Legout, N. Liogkas, E. Kohler, and L. Zhang. Clustering and sharing incentives in BitTorrent systems. In *Proceedings of the ACM SIGMETRICS'07*, pages 301–312, New York, NY, USA, 2007. ACM.

[10] A. Legout, G. Urvoy-Keller, and P. Michiardi. Rarest first and choke algorithms are enough. In *Proceedings of IMC '06*, October 2006.

[11] D. Levin, K. LaCurts, N. Spring, and B. Bhattacharjee. BitTorrent is an auction: Analyzing and improving BitTorrent's incentives. In *Proceedings of SIGCOMM '08*, August 2008.

[12] N. Liogkas, R. Nelson, E. Kohler, and L. Zhang. Exploiting bittorrent for fun (but not profit). In *Proc. of IPTPS*, 2006.

[13] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer. Free riding in BitTorrent is cheap. In *Proceedings of HotNets '06*, November 2006.

[14] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. Do incentives build robustness in BitTorrent? In *Proceedings of NSDI '07*, April 2007.

[15] PlanetLab. Planetlab - an open platform for developing, deploying, and accessing planetary-scale services. http://www.planet-lab.org.

[16] A. Sherman, J. Nieh, and C. Stein. Fairtorrent: bringing fairness to peer-to-peer systems. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT '09, pages 133–144, New York, NY, USA, 2009. ACM.

[17] R. Thommes and M. Coates. Bittorrent fairness: analysis and improvements. In *Proc. Workshop Internet, Telecom. and Signal Proc.* Citeseer, 2005.

[18] R. Xia and J. Muppala. A survey of bittorrent performance. *Communications Surveys & Tutorials, IEEE*, 12(2):140–158, 2010.