University of California

Los Angeles

Improving the BitTorrent Protocol Using Different Incentive Techniques

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Rafit Izhak-Ratzin

2010

The dissertation of Rafit Izhak-Ratzin is approved.

Deborah Estrin

Mario Gerla

Leonard Kleinrock

Mihaela Van-Der Schaar, Committee Co-chair

Rupak Majumdar, Committee Co-chair

University of California, Los Angeles

2010

To my amazing family

My husband Sagi that shares the entire way with me

My beloved children Gonni and Gur that keep my spirit up

My parents Aviva and David for their unconditional love and support

Table of Contents

List of Figures

List of Tables

Acknowledgments

I am heartily thankful to my advisor, Rupak Majumdar, one of the smartest people I know, for his continuous support and guidance. Rupak was always there to listen and to give technical and editorial advice. I also thank him for giving me the freedom to choose my research topics and to explore on my own.

A very special thanks to my co-advisor, Mihaela Van der Schaar, who is a role model for me being lively, enthusiastic, and energetic. I would like to thank her for all the advice, collaboration, opportunities, and her willingness to share her bright insights with me, which were very fruitful for shaping my ideas and research.

Two of the works that are described in this thesis are joint work with my advisor Rupak Majumdar, my co-advisor Mihaela Van Der Schaar, Hyugon Park, and Nikitas Liogkas. Rupak and Mihaela helped me clarify my thoughts and provided valuable feedback. Hyuggon Park contributed to the formalization of the foresighted Model (Section 5.2). Nikitas contributed insights into the operation of BitTorrent systems. It is a pleasure to pay tribute to Nikitas for our numerous fruitful discussions. My understanding of BitTorrent would not be the same without him.

To Deborah Estrin, who first met me in a decisive moment and persuaded me to continue and not to give up. I would like to thank her for the advice she gave me, which directly effected this thesis subject. You are an inspiration to me.

I gratefully thank Mario Gerla and Leonard Kleinrock that in the midst of all their activity, they accepted to be members of my committee.

I was fortunate in having Yuval Shavitt as my first advisor in Tel Aviv University. I never could have embarked all of this without him introducing me to research.

I would like to show my gratitude to Verra Morgan for all her help with finding funds, administration, and other bureaucratic matters.

I would like to thank my very special friends and colleagues for being there for me in all the dry runs, reviewing my papers, and discussing my work: Cesar Marcondes, Luiz Vieira, Uri Schonfeld, Elias Bareinboim, and Rafael Laupher.

Finally nothing would be complete without the love and support of my amazing family who experienced with me all my successes and failures these past few years. My amazing husband that shared all this journey with me. My boy, Gonni, that was born just few months before starting the program, and my little boy, Gur, that was born during my work on the PhD. Thank you for keeping me happy, and hopefully now you will have more mommy time. To my amazing parents that education for them is the most important investment, who always support and love my family and me unconditionally. To my extended family my brothers and sister, my in-laws, my aunts, my cousins and all my friends in Israel and all over the world that are just like a family for me, thanks for all the support, the love, the encouragement, everyone of you has a special place in my heart.

Vita

1992 - 1995	Lieutenant, Israel Defense Forces, Israel.
1998 - 1999	Lecturer, Ben Gurion University, Beer-Sheva, Israel.
1999 - 2000	Research and Development Engineer, Cisco Systems, Herzliya, Israel.
2000	B.Sc with Magna Cum Laude in Communication Systems Engineering
2000 - 2003	Research and Development Engineer, Avaya Communication, Tel-Aviv, Israel.
2003	MSc in Electrical Engineering Tel Aviv University, Tel Aviv, Israel.
2003–2005	Research Assistant, IRL Lab Computer Science Department, UCLA.
2005	MSc in Computer Science, UCLA, Los Angeles, California.
2006	Teacher Assistant, UCLA, Computer Science Department Los-Angeles, California.
2006	Research and Development Engineer, Google, Mountain View, California.
2006–2009	Research Assistant, Computer Science Department, UCLA.

Publications and Presentations

Rafit Izhak Ratzin, August 2009, Team Incent ives in BitTorrent Systems, Presented in ICCCN Conference, San Francisco California

Rafit Izhak Ratzin, July 2009, Team Incentives in BitTorrent Systems, Presented in DB-UCLA seminar, Los Angeles California

Rafit Izhak Ratin, May 2009, Collaboration in BitTorrent Systems , Presented in Networking 2009 Conference- IFIP, Aachen, Germany

Gunes Ercal and Rafit Izhak-Razin and Rupak Majumdar and Adam Meyerson, *Frugal Routing on Wireless Ad-Hoc Networks.*, SAGT, 2008

Rafit Izhak-Razin, *Collaboration in BitTorrent Systems - Best Paper Award.*, Networking, 2009

Rafit Izhak-Razin and Nikitas Liogkas and Rupak Majumdar, *Team Incentives in BitTorrent Systems.*, ICCCN, 2009

Ricardo Olivera, Beichuan Zhang, Dan Pei, Rafit IzhakRatzin, and Lixia Zhang, *Quantifying Path Exploration in the Internet.*, IMC06

Ricardo V. Oliveira, Rafit Izhak-Ratzin, Beichuan Zhang, and Lixia Zhang, *Measurement of Highly Active Prefixes in BGP.*, Globcom06

Hyunggon Park, Mihaela, Rafit Izhak-Ratzin, and van der Schaar, *Book Chapter: Peer-to-Peer Networks – Protocols, Cooperation and Competition.*, Streaming Media Architectures, Techniques, and Applications: Recent Advances 2010.

Abstract of the Dissertation

Improving the BitTorrent Protocol Using Different Incentive Techniques

by

Rafit Izhak-Ratzin

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2010

Professor Rupak Majumdar, Co-chair

Professor Mihaela Van-Der Schaar, Co-chair

Peer-to-peer (P2P) content sharing protocols dominate the traffic on the Internet [IPO09], and thus have become an important piece in building scalable Internet application. Peers in P2P network are typically independent entities that together form a self-organizing, self-maintaining network with no central authority. As a result, P2P network performance is highly dependent on the amount of voluntary resources individual peers contribute to the system. However, peers in P2P systems have self-interest to control their degree of collaboration and contribution [SP03, FS02, Pap01], as cooperation may incur significant communication and computation costs. Thus, rational peers may refuse to contribute their fair share of resources [AH00, SPD02, HCW05]. In some scenarios this may lead to the "tragedy of commons" [G68], when maximizing peers' own utilities may effectively decrease the overall utility of the system. Hence, mechanisms that incentivize peers to actively cooperate and contribute their resources are fundamental and crucial for these systems' continued success. In such mechanisms, fairness among the peers participating in content distribution is an important factor, as it incentivizes peers to contribute their resources.

Recent research efforts have shown that the popular BitTorrent P2P protocol [Coh03] does not strictly enforce fairness. Moreover BitTorrent allows free-riding, in spite its tit-for-tat mechanism. In an effort to address the incentive-related problems in the BitTorrent protocol, we propose and examine three different protocols that encourage cooperation and contribution of resources, and improve fairness and resistance to free-riding in BitTorrent-like systems.

We propose the *team-enhanced BitTorrent protocol* that dynamically organizes peers of similar upload bandwidth in *teams*—groups of peers that are formed by a central entity. Based on an analytical model, we prove that the dominant strategy of a rational peer in a system with teams is to be a team member . In our experiments, we observed download time improvement of $10\% - 26\%$ for all team members, while free-riders were delayed by more than 100%.

Additionally, we propose the *buddy-enhanced BitTorrent protocol*, which distributively and dynamically creates *buddies*— pairs of peers having similar upload bandwidth whom collaborate for mutual benefit. We prove the existence of Nash Equilibrium when all of the contributing peers adopt the buddy protocol. In our experiments, we observed improving fairness in the buddy-enhanced network where high capacity peers improved their download time by $8\% - 36\%$. Moreover, in a network with free-riders, all buddy-enhanced leechers improved their download time by $2\% - 36\%$, while free-riders were delayed by $21\% - 48\%$.

Finally, we propose the *foresighted resource reciprocation protocol*. Here we use a reinforcement-learning process to capture the associated peers' statistical behaviors and their corresponding expected utilities. In our experiments, the protocol improves fairness, where high-capacity peers improved their download time by up to 33%, while delaying free-riders by $8\% - 20\%$.

CHAPTER 1

Introduction

This thesis proposes and examines three different protocols, which encourage collaboration, and improve fairness and resistance to free-riding in the popular Peer-to-peer (P2P) BitTorrent networks.

P2P networks connect many end-hosts (also referred to as peers) in an ad-hoc manner. P2P networks are typically used as file sharing applications, which share digitized content such as general documents, audio, video, electronic books, on-line gaming, real-time conferences, etc. Unlike traditional client-server networks, where servers only provide content, and clients consume the content, in the P2P networks, each peer is both a client as well as a server.

It has been observed that P2P file sharing applications dominate the traffic usage on the Internet. Wide measurements, performed on 8 different geographic regions during the years of 2008-2009 show that P2P networks generate most of the traffic in all monitored regions, ranging from 43% in Northern Africa to 70% in Eastern Europe [IPO09]. Moreover, this study also identified P2P protocols such as BitTorrent [Coh03] and eDonkey [eDo] in the traffic, and shows that BitTorrent is the most popular protocol on the Internet, generates most of the traffic in 7 out of 8 regions ranging from 32% in South Africa to 57% in Eastern Europe. This thesis is focused on how to improve this popular protocol.

1.1 Overview of P2P System

The P2P systems can be classified into two different classes: structured P2P systems and unstructured P2P systems. In structured P2P systems, peers maintain the information about resources that their neighboring peers have. Hence, data queries can be efficiently directed to the neighbor peers who have the desired data, even if the data is extremely rare. The most common indexing that is used to structure P2P systems is the Distributed Hash Tables (DHTs) indexing. Similar to a hash table, a DHT provides a lookup service with (key, value) pairs that are stored in the DHT. Any participating peer can efficiently retrieve the value associated with a given unique key. However, this may result in higher overhead in comparison to unstructured P2P networks. Different DHT-based systems such as Chord [SMK01], Pastry [RD01], Tapestry [ZHS04], CAN [SPM01] are different in their routing strategies and their organization schemes for the data objects and keys.

Unlike the structured P2P systems, the unstructured P2P systems are formed arbitrarily in a flat or hierarchical manner. In order to find as many peers that have the desired content as possible, peers in unstructured P2P systems query data based on several techniques such as flooding (e.g., among the super-peers in KaZaA [Kaz01]), random walking [CMS04], and expanding-ring (e.g. Time-To-Live counter in Gnutella [Gnu]). Three different designs of unstructured P2P systems are proposed: the centralized unstructured P2P systems, the decentralized (or pure) unstructured P2P systems, and the hybrid unstructured P2P systems.

In a centralized unstructured P2P system, a central entity is used for indexing and bootstrapping the entire system. In contrast to the structured approach, the connection between peers in the centralized unstructured approach is not determined by the central entity. The BitTorrent protocol that is discussed in Chapter 2 is an example for a centralized unstructured P2P protocol. Napster [Nap], the network that pioneered the idea of P2P file sharing, is another example for a centralized design. In Napster design, a server (or server farm) is used to provide a central directory. A peer in the network informs the directory server about its IP address and the names of the contents it makes available for sharing. Thus, the directory server knows which object each peer in the network has, and then creates a centralized and dynamic database that maps content name to a list of IPs. The main drawback of Napster design is that the directory server is a single point of failure. Hence, if the directory server crashes, then the entire network also crashes. Moreover, increasing size of the network may cause a bottleneck in the directory server, due to the need to respond to many queries and maintain a large database.

The decentralized (or pure) unstructured P2P network is an overlay network. An overlay network is a logical network. An edge in this network exists between any pair of peers who maintain a TCP connection. The decentralized unstructured overlay network is flat. All peers act as equals. There is no central server that manages the network, nor preferred peers with a special infrastructure function. The network has a single routing layer. Gnutella [Gnu] is an example of a decentralized unstructured P2P network. In order to join the Gnutella network, a user initially connects to one of several known-bootstrapping peers. The bootstrapped peer then responds with the information about one or more existing peers in the overlay network. This information includes the IP address and the listening port of each peer. The peers in Gnutella are aware only of their neighboring peers. Neighboring peers are peers that are connected with each other in the overlay network, thus, have a common virtual edge in the overlay network. In Gnutella, queries are distributed among the peers using a variation of a flooding mechanism. A peer that is interested in a specific content sends a query to its neighbors in the overlay network. Every neighbor then forwards the query to all of its neighbor peers. The procedure continues until the query reaches a specific depth search limit (counted by Time-To-Live counter). Upon receiving a flood query, a peer that has a copy of the desired content sends a query hit response to the peer that originated the query, which is an indication of having the content. The response is sent back on the reverse path of the query, using pre-existing TCP connections. The peer that originated the query then selects one peer from the responded peers, and downloads the desired content, through a direct TCP connection, from the selected peer. Although Gnutella design is simple, highly decentralized, and does not require peers to maintain information related to location of contents, it is often criticized for being non-scalable. This is because query traffic can grow linearly with the total number of queries, which in turn grows with the system size. In addition, another drawback of the protocol is that a peer that originated the query may not find the desired content especially if the content is rare.

A hybrid unstructured P2P network allows the existence of infrastructure nodes, often referred to as super-peers (or super-nodes or overlay nodes). This creates a hierarchical overlay network that addresses the scaling problems on pure unstructured P2P networks such as Gnutella. A peer in such network typically acts accordingly to the momentary role, and can become a super-peer that takes part in coordinating the P2P network structure. KaZaA [Kaz01] that is based on FastTrack [Fas] protocol is an example of a hybrid unstructured P2P network. The KaZaA network uses specially designated super-peers with high bandwidth, disk space, and processing power. When a peer joins the network, it is assigned to a super-peer. The peer then informs its super-peer concerning the contents that it will share. The super-peer facilitates the search by maintaining a database that maps content to peers, and tracking the contents specifically within its assigned peers. Similar to the centralized design, the super-peer plays a role of a directory server, although only to its assigned peers. The super-peers together create a structured overlay of super-peers, which makes the content search more efficient. A query in this network is routed to a super-peer. Then, as in the decentralized design, the query is flooded in the overlay super-peer network. The super-peer then responds to the peer that originated the query with a list of peers having the content. The hybrid networks are no longer dedicated to a single server, since the database is distributed among the super-peers. Moreover, the size of the database is relatively small, since each super-peer tracks only the contents of its assigned peers. However, the drawback of this approach is that this approach is considerably complex, and requires a non-trivial maintenance of the overlay network. Moreover, super-peers that have more responsibilities than ordinary peers may become bottlenecks.

1.2 Incentives in P2P Networks

In P2P networks peers are allowed to share their computational, storage, and networking resources to the benefit of every participant. This cooperative sharing provides the peers access to an abundance of resources they could not afford individually. These peers are typically independent entities that form a self-organizing, self-maintaining network with no central authority. Thus the system enables organic scaling as the system evolves, while requiring no dedicated infrastructure beyond network connectivity. As a result, P2P system performance is highly dependent on the amount of voluntary resource contribution from the individual peers. Most of existing P2P system are designed to address issues such as scalability, load-balancing and fault-tolerance, but assume that the peers are obedient peers that follow the protocols without consideration of their own utility and observe the system's fair use policies. However, this obedience assumption appears unrealistic in P2P systems where participants have self-interest to control their degree of collaboration and competition [SP03, FS02, Pap01], while cooperation may incur significant communication and computation costs. Thus, rational users may refuse to contribute their fair share of resources [AH00, SPD02, HCW05]. Moreover, in some scenarios, this may lead to the "tragedy of commons" [G68], since maximizing peers' own utility may effectively decrease the overall utility of the system.

Therefore, researchers have turned in recent years to take participant incentives and rationalities into consideration and design appropriate peer selection mechanisms to ensure participants are fairly sharing their resources. In such a design, fairness among peers participating in content distribution is an important factor, as it incentivizes peers to actively cooperate and contribute for dessimination of content, which again leads to improved system performance.

In this thesis we specifically focus on the popular BitTorrent (BT) protocol [Coh03] for peer-to-peer content distribution. The BitTorrent protocol strives to provide incentives to cooperate and to ensure fairness among peers: peers download similar amount to their total upload contribution to the system. Moreover clients who do not contribute data to the system should not be able to achieve high download throughput. To achieve this BitTorrent applies built-in contribution incentives through the *tit-for-tat unchoking* mechanism. Despite this, it has recently been demonstrated (e.g., [PIA07, BHP06, GCX05, LLK07]) that BitTorrent does not strictly enforce fairness and does not provide fair resource reciprocation, particularly for high capacity leechers, and especially in node populations with heterogeneous upload bandwidths. In other studies [LNK06, LMS06, SPC07], it has been shown that BitTorrent systems do not effectively cope with selfish peer behaviors, such as free-riding (i.e., a free-rider downloads data without contributing its data to other peers) and that free-riding opportunities indeed exist. Such free-riders might significantly hurt the performance of compliant peers, a hypothesis validated by our extended experimental results.

One of the main contributors to this lack of fairness in BitTorrent is the *optimistic unchokes* mechanism, a peer selection mechanism that is currently used in BitTorrent. As this mechanism facilitates continuous discovery of better peers to interact with, it also facilitates upload imbalance, dynamically creates a major opportunity for peers to obtain data without uploading in exchange. Moreover, it may induce unfairness in the system, as it forces high-capacity peers to interact with low-capacity ones. Despite this shortcoming, optimistic unchokes have been shown to greatly contribute to the robustness of the protocol [LNK06], making them more difficult to be replaced.

Another contributor to the lack of fairness is the tit-for-tat peer selection mechanism. This mechanism is based on short-term history, i.e., upload decisions are made based on most recent resource reciprocation observation. Moreover the decision is based on backward looking and not forward looking. Thus, a peer can follow the tit-for-tat policy only if it continuously uploads pieces of a particular file and as long as it receives pieces of interest in return. However, this is not always possible, as peers can have no pieces in which the other peers are interested, regardless of their willingness to cooperate [PIK08]; yet, this behavior is still perceived as a lack of cooperation.

1.3 The Goals of the Thesis

The main goal of this thesis is to improve contribution incentives in BitTorrent, an existing popular hybrid P2P system. This can be broken down into few subgoals. First, we seek to improve the protocol by proposing three different enhanced-BitTorrent protocols that use different novel techniques such as incentivizing collaboration, and foresighted resource reciprocation.

1. The *team-enhanced BitTorrent protocol* [ILM09], that dynamically organizes peers of similar upload bandwidth in *teams*— groups of peers whom are formed and coordinated by a central entity. Team members collaborate for mutual benefit, mostly satisfy their data download needs within their team, and only perform optimistic unchokes when absolutely necessary.

2. The *buddy-enhanced BitTorrent protocol* [Izh09], that distributively and dynamically creates *buddies*— pairs of peers having similar upload bandwidth that collaborate for mutual benefit. Peers in buddy mode aim to satisfy their download needs through their buddies and perform optimistic unchokes only if absolutely necessary. Moreover, a buddy relation is based on the upload history during the entire buddy relation existence, in different to the regular BitTorrent that its chocking decision is based on the usage of short-term history.

3. The *foresighted resource reciprocation protocol* [IPS10]. Here we propose to replace the peer selection mechanism with a reinforcement learning mechanism that adopts a foresighted resource reciprocation policy. We model the peer interactions in the BitTorrent-like network as a *stochastic-game*[FL99], where we explicitly consider the strategic behavior peers. The peers can observe partial historic information of associated peers' statistical recipro-

cal behaviors, through which the peers can estimate the impact on their expected utility and then adopt their best response. Thus, peers can maximize their long-term utility based on the foresighted resource reciprocation policy. The foresighted resource reciprocation policy replaces both the tit-for-tat and the optimistic unchoke mechanisms in the regular BitTorrent protocol.

The second subgoal of the thesis seeks to show that each of our implemented protocols, indeed improve contribution incentive. This we achieve by the following. We implemented all three of the proposed protocols, by modifying an existing BitTorrent implementation to incorporate our protocols. We ran experiments with various configurations on a controlled PlanetLab testbed in order to evaluate each of the proposed protocols. These include setups with peer upload capacities derived from measurements of real BitTorrent swarms [PIA07]. We evaluated the impact of the protocols on different level of contributing leechers and free-riders in comparison to the regular BitTorrent. The results demonstrated that all of the three protocols promote fairness as high capacity leechers who suffer from the unfairness in regular BitTorrent improve their download rate, and low capacity leechers who used to benefit from this are delayed. In addition, in all the three protocols free-riders are able to attain significantly lower download rates, as compared to the original protocol.

Finally, our last subgoal is to achieve a deeper understanding about incentive users in BitTorrent systems, which is achieved trough study our extended experiment results. The rest of this thesis substantiates these claims.

1.4 The Contributions of the Thesis

In this thesis we make the following contributions.

1. We propose the team-enhanced BitTorrent protocol that incentivizes collaboration within teams in BitTorrent systems.

2. We justify our protocol design and parameter choices by developing an analytical model that describes peer interactions in the presence of teams. Based on this model and game theory, we prove that the dominant strategy of a rational peer in a team-enhanced BitTorrent system is to be a contributing team member, thus providing a clear incentive to join teams.

3. We have modified an existing BitTorrent implementation to implement the team-enhanced BitTorrent protocol.

4. We have evaluated the team-enhanced BitTorrent protocol impact by running experiments on a controlled PlanetLab testbed. Our results show that the protocol enables slightly improved performance for compliant peers, while free-riders are unable to sustain high download rates, as compared to regular BitTorrent. In addition, we observe a higher degree of robustness: increasing numbers of free-riders in the system have a significantly lower negative impact on the performance of contributing peers.

5. We propose the buddy-enhanced BitTorrent protocol, which also incentivizes collaboration but in a distributed way. Similar to the team-enhanced BitTorrent protocol this protocol encourages peers of similar upload bandwidth to be buddies, and collaborate for mutual benefit.

6. We develop an analytical model that explains leechers interactions in the presence of the buddy protocol and justifies our design choices. Based on

this model and game theory, we prove the existence of a Nash Equilibrium when all the contributing leechers adopt the buddy protocol.

7. We have implemented the buddy-enhanced BitTorrent protocol on top of an existing BitTorrent implementation.

8. We have evaluated the impact of buddy-enhanced BitTorrent system by running experiments on a controlled PlanetLab testbed. Our results show that the buddy protocol promotes fairness, discourages free-riding, and improves the robustness of the system as compared to regular BitTorrent. It also provides incentives to be adopted by all the peers in the system.

9. We propose a foresighted resource reciprocation mechanism that replaces the peer selection mechanisms deployed in BitTorrent. The mechanism enables a peer to estimate its associated peers' statistical behaviors, based on interactive learning of the history of resource reciprocation. The estimated reciprocal behaviors enable a peer to find its foresighted policy. The policy aims to determine the peer's optimal resource reciprocations, and enables the peer to maximize the long-term performance.

10. We show that the mechanism improves fairness as it relies on long-term history. Moreover, we show that it hurts the free-riders directly since foresighted peers are discouraged to upload to free-riders.

11. We have implemented the proposed mechanism on top of an existing BitTorrent client.

12. We have performed extensive experiments on a controlled PlanetLab testbed to evaluate the mechanism effectiveness. Our results confirm that the foresighted resource reciprocation mechanism promotes fairness, improves the

system robustness, and discourages free-riding in comparison to the regular BitTorrent.

1.5 Organization

The rest of the thesis is organized as follows. Chapter 2 provides background on BitTorrent's operation and algorithms. Chapter 3 describes the team-enhanced BitTorrent protocol. Chapter 4 then describes the buddy-enhanced BitTorrent protocol. Chapter 5 then proposes the foresighted resource reciprocation protocol. Chapter 6 describes related work, with respect to other efforts to understand and improve the BitTorrent systems. Lastly, Chapter 7 summarizes our main results and discusses directions for future work.

CHAPTER 2

Background – BitTorrent Systems

BitTorrent is a popular peer-to-peer file sharing protocol that was created by Bram Cohen [Coh03, Coh]. BitTorrent has been shown to scale well with large number of participating end-hosts. Ipoque [IPO09] measurements for years 2008-2009 show that BitTorrent is the dominant protocol on the Internet, and that it accounted for approximately 20-57 of all Internet traffic depending on the geographical location.

2.1 System Description

The BitTorrent centralized unstructured system design is composed of two interacting units:

1. A "control level" that describes control methods such as the required file sharing preparation, which takes place prior to sharing the actual content, and the coordination among end-hosts, which is done by a central entity during the downloading process.

2. A "reciprocation level" that describes the actual data reciprocation among the end-hosts.

These two design levels are described in more detail next.

2.1.1 The BitTorrent System Design - Control Level

The BitTorrent content distribution system consists of the next entities:

1. A data content
2. An original content provider
3. The metainfo file
4. The tracker
5. The end-hosts or peers or clients

A *torrent* or *swarm* is a collection of peers participating in the download of a particular *content* which can be one (e.g. the Linux operating system) or multiple (e.g. several video or audio) files. *The tracker* is a server that coordinates and assists the peers in the swarm. It maintains the list of peers that are currently in the swarm, as well as statistics about the peers. The tracker listens on a BitTorrent TCP port for incoming client requests. While the default BitTorrent port is port 6969, some trackers may use different ports.

Prior to the content distribution, a *content provider* divides the content into multiple *pieces*, typically 256KB a piece. Each piece is further divided into multiple *pieces* typically 16KB a subpiece. The content provider then creates *a metainfo file.* The metainfo file contains information that is necessary for initiating and maintaining the download process. This information may include the URL of the tracker, the name of the data file (files), the length of the data file (files), the length of a piece, information related to multiple data files, some optional information such as creation date, author's comments, name and version of the .torrent creator. The metainfo file also contains a special string. This special

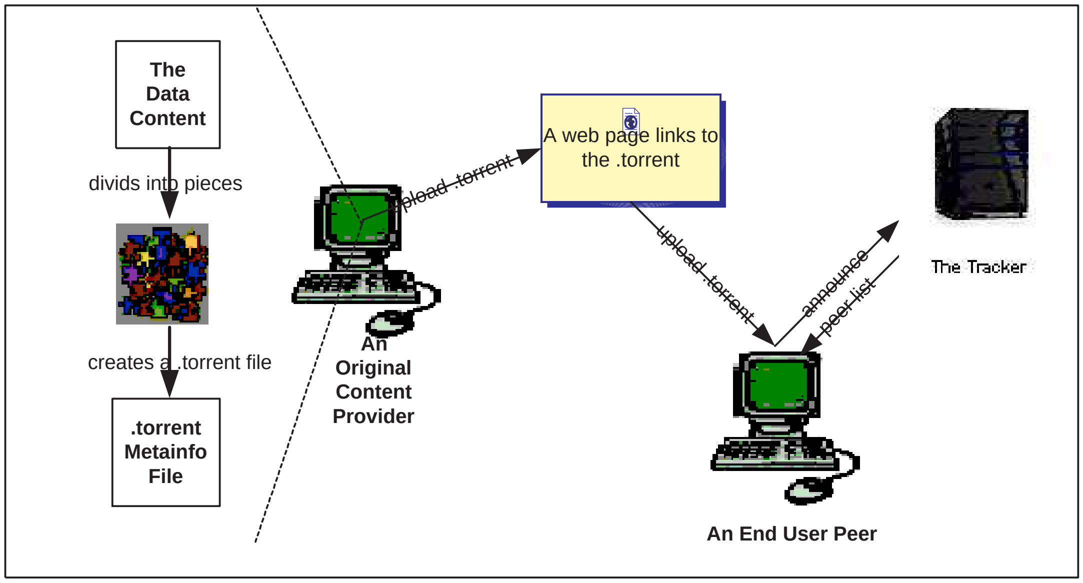


Figure 2.1: BitTorrent system, prior to sharing the file

string is a concatenation of 20-byte encoded hash values. Each value is a SHA-1 hash of a piece at the corresponding data content. These hash values are later used by the downloading peers to verify the received data pieces. The metainfo file is usually uploaded to a website, thereby making it accessible to peers that are willing to download a content by joining the swarm.

A peer that is willing to join the swarm first retrieves the metainfo file out of band. Then, this peer contacts the tracker by sending an "announce" HTTP GET request. The request in general may include necessary information such as the total amount uploaded, the total amount downloaded, the number of bytes the peer still has to download, an event description such as *started*- if it is the first request to the tracker, *completed*- if the peer shut down gracefully, *stopped*- if the download completed. This information helps the tracker to keep overall statistics about the torrent (e.g. number of *seeds*, number of *leechers*, life time of a seed, etc). The tracker responds back with a "text/plain" document, which includes a randomly selected set of peers whom are currently online. A typical size of a peer set is 50. The random peer set may include both seeds and leechers. Seeds are those peers who already have the entire content and are sharing it with others. Leechers are those peers who are still in the process of downloading (i.e. they do not possess the entire file). The new peer can then initiate new connections with the peers in the swarm and start to exchange data content pieces. The maximum number of connections, which a peer can open, is limited in BitTorrent to 80 in order to avoid performance degradation due to competition among concurrent TCP flows. In addition, the new peer is also limited to establish a fixed number of outgoing connections, typically 40, in order to ensure that some connection slots are kept available for new peers, which will join at a later time. Figure 2.1 portrays the preliminary steps, which need to be performed before starting to distribute the data content.

A Peer that has already begun downloading may contact the tracker and ask for more peers, if its peer set falls below a given threshold, which is typically set to 20 peers. Moreover, usually there is a minimum interval between two consecutive peer requests to avoid overwhelming the tracker. In addition, peers contact the tracker periodically, typically once every 30 minutes, to indicate that they are still present in the network. If a peer does not contact the tracker for more than 45 minutes, the tracker assumes that the peer has left the system and will remove the peer from the torrent list.

2.1.2 The BitTorrent System Design - Reciprocation Level

The connection between two peers starts with a handshake message followed by control and data message exchanges. The control messages between peers in the swarm as well as data messages are transferred over the TCP protocol. The range of TCP ports, which is used by BitTorrent clients is 6881-6999. The connection between peers is symmetric, the control messages in both directions have the same format, and the data can flow in either direction. The messages that are used in a connection between two peers are:

1. Handshake: the “handshake”- message is a required message and must be the first message that is transmitted by the peer.

2. Bitfield: the “bitfield”- message is an optional message and may only be sent after the handshaking sequence is completed, and before any other messages are sent. A peer can choose not to send this message if it has no pieces. Using a single bit for every piece, the message describes the pieces that were downloaded successfully, those pieces that are valid and available. Bits that are set indicate valid and available pieces, bits that are cleared indicate missing pieces.

3. Interested: an “interested” message is a notification that the sender is interested in a portion of the receiver’s data.

4. Not-interested: a “not-interested” message is a notification that the sender is not interested in any of the receiver’s data pieces.

5. Choke: the term choke is commonly used in BitTorrent as a verb that describes a temporary refusal to upload. A “choke” message is a notification that the sender will not upload data to the receiver until unchoking happens.

6. Unchoke: An “unchoke” message is a notification that the sender peer will upload data to the receiver if the receiver is interested in some portions of the sender’s data pieces.

7. Request: a “request” message is used to request a piece.

8. Piece: a “piece” message is correlated with a request message implicitly and contains piece.

9. Have: a “have” message describes the index of a piece that has been downloaded and verified via the SHA-1 hash function.

10. Keep-alive: Peers may close the TCP connection if they have not received any messages for a given amount of time, generally 2 minutes. Thus, the “keep-alive” message is sent to keep the connection between two peers alive if no message has been sent in a given amount of time.

11. Cancel: a “cancel” message is used to cancel piece requests. It is mostly sent towards the end of the download process (see more details in section 2.2.3).

A peer A in the swarm maintains a 2-bits connection state for every associated peer B that it is connected to. The first bit is the choking/unchoking bit. The second bit is the interested/not-interested bit. A connection state is initialized to choked and not interested.

Peer B transfers data to peer A only if the state of the connection with peer A is unchoked and interested. Peer B responses to peer A's “request” messages with encapsulated pieces in “piece” messages. After peer A finishes downloading a piece it verifies that the piece is uncorrupted. It calculates the SHA-1 value of the downloaded piece and compares this value with the encrypted reference value of the piece that is given in the metainfo file. Since the SHA-1 value is assumed to be unique, a corrupted piece would not match the reference hash value. After verifying that the piece is uncorrupted, peer A announces that it has the piece to all of its associated peers using “have” massage. Figure 2.1 shows an example of a possible message flow among peers that have an active connection in a BitTorrent overlay network. In the example the connection is established after peer A sends

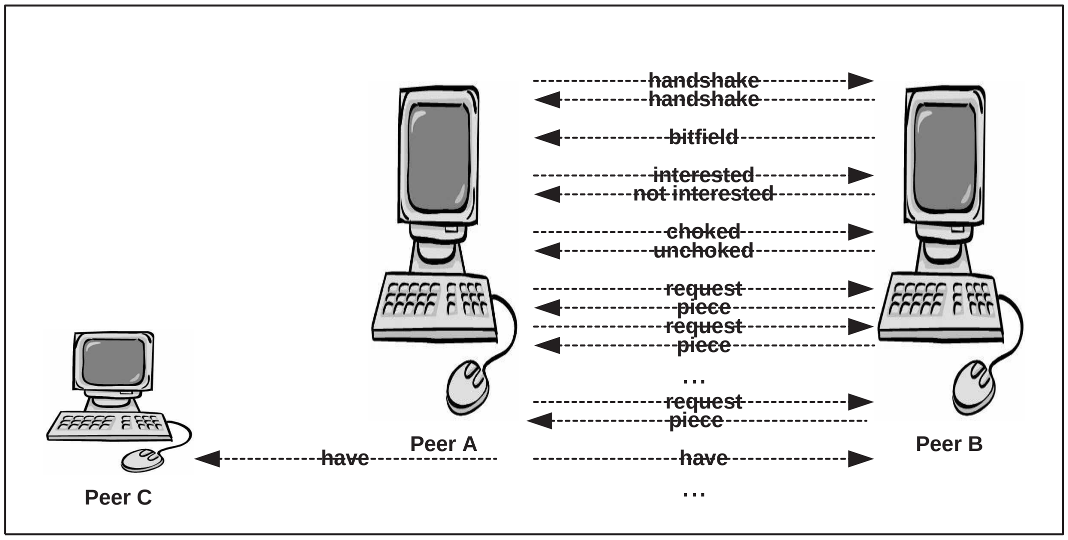


Figure 2.2: An illustrative example for a message flow among peers

a “handshake” message, and B responses with one as well. Then peer B sends a “bitfield” message but peer A does not. Such a scenario might happen if peer A has no piece ready to be shared. Peer B sends a “not interested” message to peer A, and peer A sends a “choke” message to peer B, thus data will not flow from peer A to peer B untill both messages will be replaced. On the other hand, data does flow from peer B to peer A since peer A sends an “interested” message to peer B and peer B sends an “unchoked” message to peer A. Then, peer A requests pieces of a particular piece and peer B responses with “piece” messages, by uploading the requested pieces. Once peer A gets an entire piece and confirms the validity of the piece, it sends “have” messages regarding the received piece to all the peers it is connected to in the BitTorrent overlay network.

2.2 Piece Selection Mechanisms

In the BitTorrent system, peers download the data content in random order, unlike other protocols as http or ftp, where an end-host downloads a file from start to finish. In order to facilitate such a downloading process, when a BitTorrent application is activated in a peer, the peer first allocates space for the entire content. Then, the peer tracks the pieces that each of its associated peers possess. A peer is able to identify what pieces its associated peers have by exchanging the "bitfield" message upon establishing new connections as well as by tracking the "have" messages that its associated peers send after downloading and verifying pieces. In this way, a peer is able to select a particular piece to download from a particular associated peer.

The piece selection mechanism is fundamental in achieving efficient P2P networks. A poor selection strategy can lead to inability to download, e.g., when peer is not interested in any of the pieces its associated peers have to offer, and vice versa, it can lead to inability to upload, e.g., when all associated peers are not interested in the pieces that a peer has to offer. More generally, it can prevent the peer selection mechanism to reach an optimal system capacity [LUM06].

BitTorrent applies the strict priority policy for piece selection. The strict priority policy prioritizes pieces that are part of the same piece once the first subpiece of a piece is being requested. This policy helps downloading a complete piece as quickly as possible.

The piece selection mechanism in BitTorrent is composed of three different algorithms that are applied in different stages of the downloading process. The three algorithms are: Random Piece First, Rarest Piece First, and End Game.

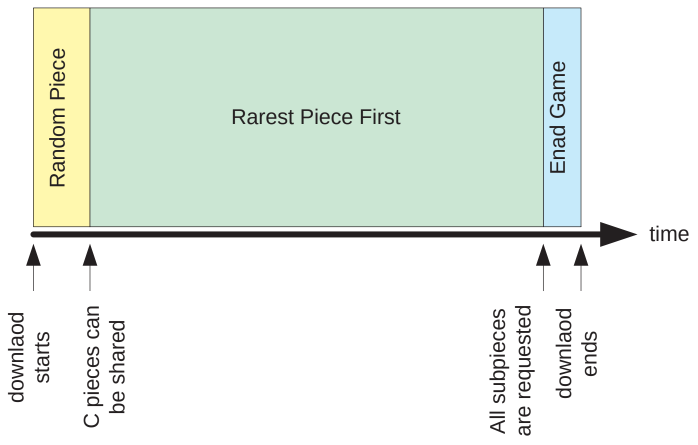


Figure 2.3: BitTorrent's piece selection algorithms

2.2.1 Rarest Piece First Selection

The rarest piece is the piece that has the least amount of copies in the peer set. For every piece, the peer maintains a counter of the number of copies that exists in its peer set. A peer, which runs the rarest piece first selection algorithm, selects the rarest missing piece as the next piece to download. If there are multiple missing pieces that are rarest pieces as well, the peer selects at random one of the rarest pieces as the next piece to download. The rarest piece first algorithm aims to achieve the follow effects on a leecher:

1. Upload pieces that many of the associated peers are interested in, such that uploading can be performed when it is needed.

2. Increase the likelihood that peers will offer pieces through the entire downloading process by leaving pieces that are more common to a later download.

3. When downloading from a seed, download new pieces first, where new pieces are those pieces that no leecher has. This is crucial especially when the system has a single seed that may eventually be taken down, since this can

lead to the risk that a particular piece will no longer be available. This is also important when the seeds in the system are slower than the leechers in the system. In this case a redundant download wastes the opportunity of a seed to get more information out to faster uploaders.

In [LUM06], the authors studied the efficiency of the rarest piece first selection algorithm in BitTorrent. More specifically, they defined the entropy of the system as the repartition of pieces among peers, and evaluated the efficiency of the rarest piece first selection strategy by characterizing the entropy of the system, with peer availability. They defined peer availability as the ratio of time a peer is interested in its associated peer. They show that the rarest piece first strategy achieves a close to ideal entropy, when each leecher is almost always interested in any other leecher.

2.2.2 Random Piece First Selection

At the beginning of the downloading process, when a leecher has nothing to share, the leecher should download pieces faster than in the rarest-first piece selection strategy, as it is important that a new peer that has just joined the swarm will get some complete pieces, and start reciprocate pieces as soon as possible. A piece that is chosen at random is likely to be more replicated than the rarest piece, thus its download time on average will be shorter by downloading simultaneously from more peers. Hence, at the beginning of the downloading process the peer selects a piece to download at random applying the random piece first selection algorithm. Once the peer downloads C pieces that are ready to be shared (C is a constant that may vary in different BitTorrent client implementations), the leecher switches to the rarest piece first selection algorithm.

2.2.3 End Game Piece Selection

The end game piece selection algorithm is applied once a peer has requested all the pieces of the content. In this phase a peer requests all the subpieces that have not yet been received, from all of its associated peers that have the corresponding pieces. This is done in order to avoid a potential delay in the end of the content download since a request is sent to a peer with a very slow upload rate instead of a faster one. Once a subpiece is downloaded in the end game phase the peer sends a "cancel" messages to its associated peers that have the corresponding appending requests of the particular subpiece, this avoids bandwidth from being wasted due to redundant sends. The endgame is applied at the very end of the process and thus, may have a small impact on the downloading process.

2.3 Peer Selection Mechanisms

In BitTorrent peers download from whom they can, and upload simultaneously to a constant number of peers. The number of associated peers, which a peer uploads to, is limited in order to avoid sending data over many connections simultaneously, which may result in poor TCP congestion control behavior. Thus, peers need to make decisions about which peers to unchoke. The default number of peers to unchoke (unchoke slots) is four. However, this number may increase if a peer does not saturate its upload rate. A peer independently makes the decision regarding whom to unchoke and whom to choke every rechoke period, which is typically ten seconds. The peer uploads to unchoked peers for the duration of the rechoke period. The peer selection mechanism, which is also called the choking mechanism, can affect the performance of the system. A good peer selection mechanism should:

1. Motivate peers to contribute and upload data to the network.

2. Utilize all available resources.

3. Be resistant to free-riding behaviors where peers only download and do not upload.

In BitTorrent different mechanisms are applied in peers that are leechers and peers that are seeds.

2.3.1 Leecher's Peer Selection Mechanism

The peer selection mechanism in a leecher has two parts: The regular mechanism, which is a variant of the tit-for-tat and the optimistic unchoke mechanism.

2.3.1.1 The Tit-For-Tat Mechanism

In tit-for-tat peer selection mechanism, a leecher decides to unchoke peers that it currently downloads data from. It chooses those peers who have the highest upload rate. The idea of tit-for-tat is to have several connections that are actively transfer data in both directions at any time. In order to avoid wasting of resources due to rapidly choking and unchoking peer the designer of the protocol sets the rechoke period to 10 seconds, claiming that "Ten seconds is a long enough period of time for TCP to ramp up new transfers to their full capacity"[Coh].

2.3.1.2 The Optimistic Unchoke Mechanism

BitTorrent applies the optimistic unchoke mechanism in parallel to the tit-for-tat mechanism. The goals of the optimistic unchoke mechanism are: 1) to enable a continuous discovery of better peers to reciprocate with, 2) to bootstrap new leechers who do not have any content piece, to download some data and start reciprocate pieces with others. The optimistic unchoke mechanism chooses to unchoke a peer randomly regardless to its current upload rate. Optimistic unchoke is rotated every optimistic unchoke period, when an optimistically unchoked peer is unchoked for the entire optimistic unchoke period. The designer of the protocol chose the optimistic unchoke duration to be 30 seconds, since 30 seconds is enough time for the upload to get to full capacity, for the download to reciprocate, and finally for the download to get to full capacity. Optimistic unchoke is typically applied on a single unchoke slot while the tit-for-tat is applied on the rest of the unchoke slots. New connections are three times as likely to start at the current optimistic unchoke as any other connection in the rotation.

Note that the amount of data, which peer uploads, is not a function of its download from the particular unchoked peers but only a function of its bandwidth, which the uploading task is able to use. The upload bandwidth is divided equally among the unchoked peers.

2.3.1.3 Anti-Snubbing

If a peer has received nothing from a particular peer, for a certain amount of time, typically 60 seconds, it marks the particular peer's connection as snubbed. Peer does not upload to an associate snubbed peer except as an optimistic unchoke. This may result in more than one simultaneous optimistic unchoke, when the peer is choked by many of its associated peers. In such a case the peer may experience poor download rates until the optimistic unchoke finds better peers. Thus, increasing number of optimistic unchokes in this scenario is important.

2.3.2 Seed's Peer Selection Mechanism

Seeds, who do not need to download any pieces, follow a different peer selection mechanism than leechers. The most common mechanism is a round-robin manner striving to distribute data uniformly.

2.4 Limitations of BitTorrent Systems

In addition to the analytical models of BitTorrent [QS04, FCL06, Izh09, ILM09, NPZ07, MV05, LLS08], some measurement studies have been performed [GCX05, IUB04, PGE05, PIA07, PIK08]. Most of these studies were performed on peers that were connected to public torrents. These studies provide interesting details about the overall behavior of deployed BitTorrent systems. The studies also pinpoint some limitations of the BitTorrent protocol. In this section we discuss two of the BitTorrent protocol limitations that were shown in these studies:

1. The feasibility of free-riding.
2. The lack of fairness.

2.4.1 Free-Riding in BitTorrent Systems

Free riders are those peers who attempt to circumvent the protocol mechanism and download data without uploading any data to other peers in the network. Researchers have argued that free-riding in BitTorrent is feasible through the optimistic unchoke mechanism and through seeds [SPM04, JA05, LNK06, LMS06, SPC07]. We discuss these works in details in Chapter 6. Widely adoption of free-riding strategy can result in a tragedy of the commons where overall performance in the system will decrease.

2.4.2 Lack of Fairness in BitTorrent Systems

Fairness in BitTorrent is commonly defined as peers receive as much as they give. Fairness among peers participating in content distribution such as in BitTorrent encourages peers to actively collaborate in disseminating content. Thus, fairness is an important factor, which can lead to improved system performance. However, research studies, such as [GCX05, PIA07, BHP06, LLK07], show that BitTorrent does not provide fair resource reciprocation, especially in node populations with heterogeneous upload bandwidths. Two of the parameters that contribute to this luck of fairness are the tit-for-tat mechanism and the optimistic unchokes mechanism, which together compose the peer selection mechanism in BitTorrent.

The peer selection mechanism, can be modeled as a distributed algorithm for the stable *b*-matching problem [**?**], as noted in [**?**, LLK07],

The algorithm converges to a (weakly) stable state in which peers are matched based on their upload capacity and no peer has an incentive to deviate from its matches. The algorithm converges to a stable states where peers that have the same upload rates are matched, and do not have insentive to deviate. The algorithm converges to a stable state through a series of random exploration rounds (i.e. optimistic unchokes) in which unstable matchings are formed: in these intermediate cases, a peer may end up being matched to peers that cannot sustain a fair reciprocation. This implies that some peers might offer more upload bandwidth than they receive. The time it takes for the algorithm to converge may be up to few hours [CNM08].

In addition, the tit-for-tat mechanism is based on short-term history, i.e., upload decisions that are made based on the most recent resource reciprocation observation. Thus, a peer can follow the tit-for-tat policy only if it continuously uploads pieces of a particular file and as long as it receives pieces of interest

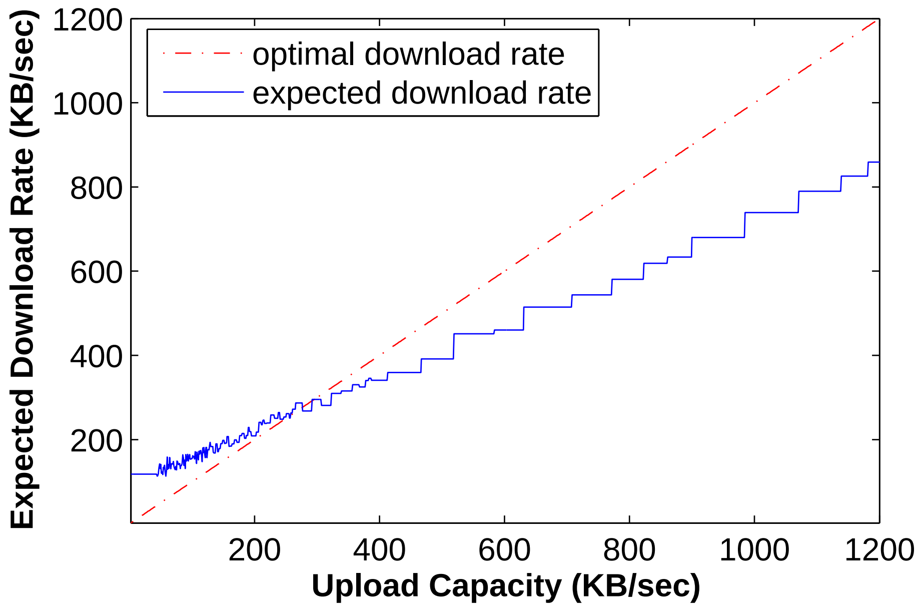


Figure 2.4: Upload vs. Download in BitTorrent network

in return. However, this is not always possible, as peers can have no pieces in which the other peers are interested in, regardless of their willingness to cooperate [PIK08]; yet, this behavior is still perceived as a lack of cooperation and lead to loosing a good peer match while entering to an unstable period where the peer searchs again for a good peer match.

The impact of optimistic unchokes on fairness in the BitTorrent system is shown in Figure 2.4. More specifically, the Figure shows the impact of optimistic unchokes on the expected download rate as a function of the peers upload rate. The peer upload rate distribution is based on observed bandwidth distribution given in [PIA07]. It is assumed that one unchoke slot is used by the optimistic unchokes mechanism, and that the regular unchoke mechanism works perfectly in term of fairness, i.e. the download rate and the upload rate through tit-for-tat unchokes are equal. Clearly, we can see that the sub-linear behavior of the expected download rate leads to unfairness for high capacity leechers that are forced to interact with low-capacity ones, and low capacity leechers who benefit from this unfairness. Note that in the observed bandwidth distribution the majority of the leechers are low capacity leechers with 84% of the leechers having less than 250KB/s upload capacity.

Finally, the number of unchoke slots may also contribute to the luck of fairness in BitTorrent. The number of unchoke slots that a leecher uses for regular unchoke is a function of the leecher's ability to saturate its upload capacity. This again might lead to unfairness, since typically, in real torrents, the download capacity of a leecher might be greater than the upload capacity. Thus, high capacity leechers might upload in full capacity, but not be able to download as much due to upload constraints of the downloading leechers and limited number of unchoke slots.

CHAPTER 3

Team Incentives in BitTorrent Systems

3.1 Overview

The popular BitTorrent (BT) protocol for peer-to-peer content distribution, which was described in detail in the previous section, strives to ensure fairness: clients who do not contribute data to the system should not be able to achieve high download throughput. However, it has been recently demonstrated (e.g., [LNK06, LMS06, SPC07]) that BitTorrent does not strictly enforce fairness, and moreover that free-riding opportunities indeed exist. Such free-riders might significantly hurt the performance of compliant peers, a hypothesis validated by our experimental results.

One major way for a peer to obtain data without uploading to others is via so-called *optimistic unchokes*, a BitTorrent mechanism that facilitates the continuous discovery of better peers to interact with. Despite this shortcoming, optimistic unchokes have been shown to greatly contribute to BitTorrent's robustness [LNK06], making them difficult to be replaced.

In this chapter we propose exactly such an alternative mechanism that mostly obviates the need for optimistic unchokes in BitTorrent, while improving fairness and system robustness. Our approach advocates dynamically organizing peers of similar upload bandwidth in *teams*— groups of peers collaborating for mutual benefit. Team members mostly satisfy their data download needs within their team and only perform optimistic unchokes when it is absolutely necessary.

We modified an existing BitTorrent client and tracker implementations to incorporate our team protocol. We evaluated the modified protocol impact on contributing and free-riding peers, by running experiments with different configurations on a controlled PlanetLab testbed. These include setups with peer upload capacities derived from measurements of real BitTorrent swarms [PIA07]. Our results demonstrate that free-riders are able to attain significantly lower download rates in the presence of teams, as compared to the regular BitTorrent protocol, while at the same time the performance of contributing peers improves slightly. In particular, we show that the team protocol exhibits the following properties.

1. It provides incentives for contributing peers to join a team by rewarding them with improved download rates compared to being independent.
2. It discourages free-riding by limiting the number of optimistic unchokes, thereby directly hurting free-rider performance.
3. It results in a more robust system: increasing numbers of free-riders have a significantly lower negative impact on the performance of contributing peers, as compared to regular BitTorrent.

We also justify our protocol design and parameter choices by developing an analytical model that describes peer interactions in the presence of teams. Based on this model and game theory, we prove that the dominant strategy of a rational peer in a $team-enhanced$ BitTorrent system is to be a contributing team member, thus providing a clear incentive to join teams.

The rest of this chapter is organized as follows. Section 3.2 describes the design of the $team-enhanced$ protocol, while Section 3.3 presents the analytical model that has guided this design. We discuss some implementation details in Section 3.4, and then present the results of our PlanetLab experiments in Section 3.5.

3.2 Design

We now present the design of the team protocol, which augments BitTorrent with the notion of *teams*, groups of peers with similar upload bandwidth collaborating for mutual benefit. The main goal of the protocol is to improve peer performance via explicit cooperation, in addition to the tit-for-tat choking algorithm already employed by BitTorrent. At the same time, it aims to limit bandwidth spent on random optimistic unchokes, a major opportunity for free-riding [SPC07]. To achieve these goals, we design the team protocol to exhibit the following properties.

- It enables a peer to find a team of other peers with similar upload bandwidth, and provides that peer with a clear incentive to join the team, by rewarding it with improved download rates compared to being independent.
- It discourages free-riding by significantly limiting the number of optimistic unchokes team members perform, thereby directly hurting free-riders' performance.
- It enables team members to collectively detect and punish free-riders inside their team.

The protocol comprises three main parts: (1) the process by which teams are formed, (2) the algorithm peers use to decide whom to unchoke, and (3) the detection and punishment of free-riders. We describe these in turn.

3.2.1 Team Formation

Team formation is centrally coordinated by the tracker, who decides how to allocate peers into teams based on their available upload bandwidth. The tracker makes this decision based on the following procedure: Every team has to satisfy the condition $\frac{U_{max}}{U_{min}} \leq (1+x)$, where the *range factor* parameter $x \in [0,1]$ defines the allowable range of upload bandwidth for peers belonging to the given team; U_{max} and U_{min} are the upload bandwidths of the fastest and slowest team member respectively. Thus, to assign a peer P with upload bandwidth U_P to a team, the tracker makes sure that the following two *range factor conditions* are satisfied:

$$U_P \leq U_{min} \cdot (1+x),\ \textit{and}\ U_P \geq \frac{U_{max}}{(1+x)} \tag{3.1}$$

As a result of satisfying these conditions, the slowest team member can be slower than the fastest team member by at most a factor of $1+x$. Given an appropriate selection of x, this ensures that all team members have similar upload bandwidth. There is of course a trade-off involved in the value of x. It should not be too large, as fast team members will then suffer from interacting with much slower team members. It should not be too small either, as that could cause the creation of many small-sized teams. We discuss a concrete method of calculating a good value for x in Section 3.4.2.

We define a *full team* to be a team with T members, where T is a fixed constant that depends on the number of available unchoke slots at the peers. For our PlanetLab experiments, for instance, we set this to 4, which is the default number of unchoke slots for the BitTorrent implementation we use. A *matching team* for a peer i is simply a team that is not full whose bandwidth range includes i's upload bandwidth.

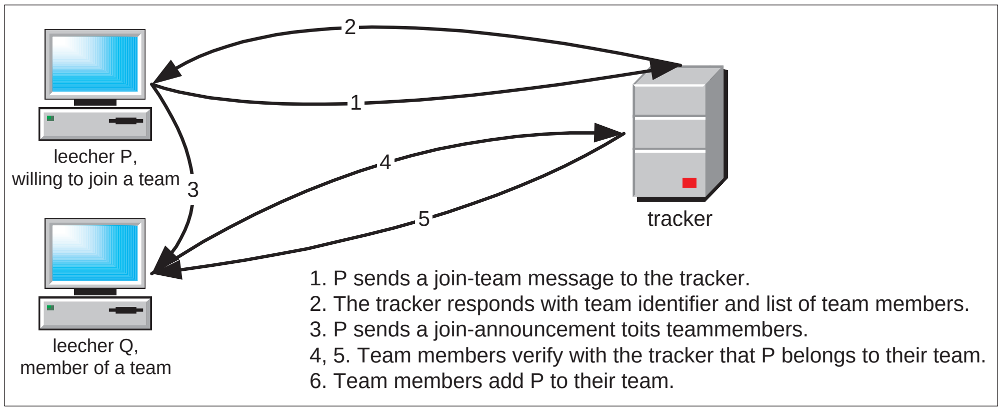

2
1
3
4
5
leecher P,
willing to join a team
leecher Q,
member of a team
tracker
1. P sends a join-team message to the tracker.
2. The tracker responds with team identifier and list of team members.
3. P sends a join-announcement toits teammembers.
4, 5. Team members verify with the tracker that P belongs to their team.
6. Team members add P to their team.

Figure 3.1: Message flow for the team formation process

In order to thwart free-riding attempts targeted at the new mechanisms the team protocol introduces, the tracker enforces certain rules for a peer joining a team. First, it imposes a limit on the frequency of switching from one team to another, by keeping the history of team joins for each peer in the system for a short time period. In this manner, a peer can only join a fixed number of different teams within a given time interval. In addition, a peer is not allowed to join a team if it has been found non-compliant with the team protocol in the recent past, e.g., if it lied about its upload bandwidth. In such a case, i is blacklisted by the tracker and is not allowed to participate in teams. We describe in Section 3.2.3 how such non-compliant peers can be detected.

The team formation process is shown in Figure 3.1.

Step 1. A leecher P who wishes to join a team needs to estimate its own available upload bandwidth first. This can be done either by simply consulting the upload limit set by the user, or by monitoring its own uploads during a short initial measurement period. The leecher then sends a *join-team* message to the tracker (separately from the regular initial BitTorrent messages), which includes its estimated bandwidth.

Step 2. Upon receiving such a message, the tracker checks whether the given leecher is blacklisted due to misbehavior or has exceeded the limit on the frequency of switching teams. If none of these is true, the tracker searchs for a matching team for the leecher. If more than one matching teams are found, the tracker selects the team that minimizes the bandwidth differences between team members, i.e., the one with $min_T d(P,T)$, where $d(P,T) = \frac{1}{|T|}\sum_{P' \in T} |U_P - U'_P|$. If the tracker cannot find a matching team, it creates a new team for the leecher. It then sends back a *join-team response* that includes a unique team identifier and a list of the other team members belong to this team.

Step 3. Upon receiving the response from the tracker, the leecher sends a *join-announcement* to all the team members that belong to its team, notifying them of its wish to join their team.

Step 4. A team member that receives a join-announcement message verifies its validity by sending a *join-verification* message to the tracker with the announcing leecher's identifier (peers in BitTorrent are uniquely identified by strings chosen by peers themselves). This step prevents malicious peers from circumventing the previous protocol steps in an attempt to join an arbitrary team.

Step 5. Upon getting a join-verification message, the tracker checks if its sender and the announcing leecher contained in the message should indeed belong to the same team, and if so, sends an affirmative response to the sender.

Step 6. Upon receiving such a response, the team member adds the announcing leecher to its team. Otherwise, it ignores the join-announcement message.

This process ensures that leechers can only join teams after the approval by the tracker. It is also robust to malicious team members, who can in no way cause the acceptance of a new member by the rest of the team.

3.2.2 Peer Selection Decisions

Once in a team, a leecher has to periodically decide which peers to unchoke. In regular BitTorrent, this decision is made based on two separate strategies, the tit-for-tat strategy, which called also the regular peer selection algorithm, and the optimistic unchoke strategy. The former guides a leecher to unchoke those that upload data at the highest rates, while the latter selects peers at random, independently of their contributions. Our team protocol extends these with a third, *team unchoke* strategy, which dictates that a team member should always unchoke all members of its team. The rationale is that team members should upload to other team members whenever possible, in order to maximize team performance. At the same time, at least one regular unchoke is performed to an external peer to avoid isolating the team from the rest of the swarm; external interactions are required to introduce new content pieces into the team. Uploading via optimistic unchokes is performed only as a last resort, when a leecher has no data other team members are missing and thus cannot upload to them.

Assuming that there are n_s available unchoke slots in total, the combined unchoke decision algorithm is as follows.

Step 1 (team unchokes). A leecher A who belongs to a team strives to satisfy all its team members that are "interested" in some of its data pieces that are ready to be shared. We denote the number of unchoke slots reserved by leecher A for its team members as a function of time with $n_T(t)$, where $0 \leq n_T(t) \leq n_s - 1$, having at least one slot reserved for regular unchoke of a peer that is not part of leecher A's team. Note that the team size T is bounded by n_s. If a peer is *independent*, i.e., not a member of a team, then $n_T(t) = 0$.

Step 2 (optimistic unchokes). Leecher A performs optimistic unchokes to external peers with probability

$$P_T(t) = P_0 \frac{n_s - n_T(t) - 1}{n_s - 1} \tag{3.2}$$

where P_0 is the probability for an optimistic unchoke in regular BitTorrent, which depends on the implementation. We denote with $n_O(t)$ the number of slots used for optimistic unchokes as a function of time.

Step 3 (Regular unchokes). For the rest of the unchoke slots (that are not used for team unchokes or optimistic unchokes), $(n_s - n_T(t) - n_O(t))$, leecher A observes its associated peers uploading rates, and selects the peers with the highest uploading rates that were not selected already.

Note that the probability P_T of independent leechers to perform optimistic unchokes is the same in the team protocol as in the regular BitTorrent (for $n_T = 0$, $P_T = P_0$). However, for team members, this probability decreases as more team members are served (or as n_T increases). Those who belong to a full team and can serve all their team members at the same time perform no optimistic unchokes at all (for $n_T = n_s - 1$, $P_T = 0$).

Based on the design of this peer selection algorithm, we expect it to have two effects.

1. The total number of optimistic unchokes in the system should be reduced, thereby directly hurting free-riders who critically depend on such unchokes to obtain data.

2. Team member who suffer from asymmetric download rate through optimistic unchokes should improve their download rate, as there are now symmetric peers (peers who have similar upload rate) who make it their priority to serve them as a replacement to the optimistic unchokes.

Our experimental evaluation results bear out both these hypotheses. In a sense, *teams mostly replace BitTorrent's optimistic unchoke mechanism* as the means of discovering better partners. Having the guarantee of other leechers with similar upload bandwidth willing to upload to them, team members no longer need to explore the network as much.

3.2.3 Detection and Punishment

So far we have assumed that all peers follow the team protocol in a compliant manner. We now describe mechanisms that can be used to detect and punish team members who attempt to circumvent the protocol for their benefit. A compliant team member can periodically execute the following steps to verify compliant behavior by other team members.

Step 1 (self upload examination). Every team member keeps track of the total amount of data they have downloaded from and uploaded to all other team members. We denote these with $D_i(t)$ and $U_i(t)$ respectively for a team member i at time t. Every team member then checks if it is providing satisfactory service to others by verifying that the condition $\frac{U_i(t)}{D_i(t)} \leq (1+x)$ is true for the majority of its team members (this condition follows from Equation 3.1). If the condition is false, a compliant leecher sends a *leave-team* message to its team, and a *switch-team* message to the tracker to switch to another team that is more appropriate for its upload bandwidth. This message includes the leecher's upload bandwidth, like the initial join-team message.

Step 2 (team upload examination). Moreover, every team member examines other members' upload rates to verify they are not lower than expected (every team member i has to satisfy the condition $\frac{D_i(t)}{U_i(t)} \leq (1+x)$). If this is not the case, the team member executes the punishment process, described below, to attempt expelling peer i from its team.

Step 3 (team satisfaction examination). Even without free-riders in a team, a leecher might still not be satisfied with the team's performance. This can happen either because its available upload bandwidth has increased since joining the team, or because other team members did not concur to punish a peer that did not satisfy step 2. Thus, the peer can also determine its satisfaction with the team as a whole by checking whether $\frac{\sum_{i=1}^{T-1} D_i(t)}{\sum_{i=1}^{T-1} U_i(t)} \leq (1+x)$. If this is false, the peer can send a switch-team message to the tracker to switch teams.

Note that the execution intervals for steps 2 and 3 should be longer than the execution interval for step 1, to allow team members to leave their team if they cannot upload fast enough before being punished. However, it should not be too long, so as to allow free-riders to abuse team unchokes before being detected. Determining the optimal values for these parameters is the subject of ongoing work.

Punishment process. The punishment process described here can be invoked by any team member j for another team member i if j is not satisfied with the upload rate it is getting from i, and the team is not already in the process of deciding whether to expel i. To initiate the process, j sends *unsatisfied* messages about i to its team members (excluding i). Upon receiving these messages, the team members execute the team upload examination on i, and, if they also find it to be unsatisfactory, they too send an unsatisfied message to their other team members (excluding i).

Once a team member receives unsatisfied messages from the majority of its team, it sends an unsatisfied message to the tracker, who keeps count of how many such messages it receives. If this count exceeds half of the size of the team, the tracker proceeds to remove peer i from the team and adds it to a blacklist. The blacklisting prevents peer i from joining another team for a given period of time. Note that a single malicious team member can in no way cause another team member to be blacklisted. Only a collusion of at least half the team members can achieve that, in which case compliant leechers would arguably leave the team on their own due to the team satisfaction examination. Moreover, a collusion of group of peers to join the same team is not obvious to perform, since not the peers themselves but the tracker is in charge of setting peers into teams.

When blacklisting a team member, the tracker also replies back to the unsatisfied team members with a *punish* message. The team members then proceed to forward the punish message to the rest of their team, including peer i, effectively notifying it of its expulsion. This process of forwarding messages is necessary, because, unlike team members, the tracker does not maintain open connections with all leechers in the team. The compliant team members remove peer i from their local team list upon receiving a punish message from at least half the team or from the tracker. Since teams are typically small, the communication overhead of the punishment procedure should not be prohibitive, but we are still in the process of evaluating this.

3.3 Analytical Model

We have validated the protocol design presented in the previous section both via analysis and by running experiments on a controlled testbed. The analytical model we have developed describes peer interactions in the presence of teams as a repeated game with rational players. This section presents this model and demonstrates that the dominant strategy of a leecher in a team-aware BitTorrent system is to be a contributing team member. The next section presents our experimental results.

3.3.1 Optimistic Unchoke Probability

We assume that all teams in the system have the same size T (enforced by the tracker), and that all contributing leechers have the same total number of unchoke slots n_s (which is a good approximation for most implementations). In addition, a leecher belonging to a team with at least two members is considered a team member, otherwise it is considered an independent leecher or free-rider, depending on its strategy choice. We define $n_{T_{max}}$ to be the maximum number of unchoke slots that can be taken up by a leecher's team members, and, as we explained, $n_{T_{max}} = n_s - 1 = T - 1$. Given the peer selection algorithm (Section 4.2.3), if all contributing leechers belong to teams and all of them can serve all their team members at time t, then the probability for an optimistic unchoke by any peer in the system at that time will be 0. As a result, free-riders will not be able to receive any data via optimistic unchokes and will only be able to download data from seeds or individuals, thereby achieving significantly lower download rates. We will now prove that the probability for this to occur in a team-aware BitTorrent system is very high.

Similarly to Qiu *et al.* [QS04], we assume that:

A1. If K is the total number of content pieces, the number of pieces each leecher has will be uniformly distributed in $[0, ..., K-1]$, due to the random arrival of peers to the network.

A2. Let k_i denote a random variable representing the number of pieces a given leecher i has. These k_i pieces would then be chosen randomly from the set of all content pieces. As shown by Dale *et al.* [DL07], this is a reasonable assumption, as BitTorrent peers typically follow a rarest-first piece selection policy, which is effective in replicating pieces uniformly.

Theorem 3.3.1. *Under assumptions A1 and A2, the probability for a team mem-*

x	bandwidth range factor
T_i	size of the team that leecher i belongs to
n_s	total number of unchoke slots
$n_T(t)$	number of unchoke slots taken up by team members at time t ($n_T \leq n_s - 1$)
$n_O(t)$	number of unchoke slots used for optimistic unchokes
P_0	probability to perform optimistic unchoke in regular BitTorrent
P_T	probability to perform optimistic unchoke by a team member
K	total number of pieces of the content
$P_{hasall}(i, j)$	probability of leecher i to have all the content pieces that leecher j has
U_i	utility function of peer i
u_i	upload capacity of peer i
R	punishment duration (in rechoke periods)
δ	future discount factor for punishment
d_T	the download rate of a leecher due to its participation in a team

Table 3.1: Team Model Notation

ber to perform an optimistic unchoke $P_T \rightarrow 0$ *as* $K \rightarrow \infty$*, with high probability.*

Proof. The probability in question is given by Equation 4.1, with P_0 and n_s given constants. In order then to show that $P_T \rightarrow 0$ with high probability, we need to show that $n_T \rightarrow n_s - 1 = n_{T_{max}}$, with high probability. Let us define $P_{hasall}(i, j)$ the probability that team member i has all the content pieces that team member j has, so that j cannot upload anything of interest to i, and thus might have to perform an optimistic unchoke instead. Qiu *et al.* [QS04] have already shown that $P_{hasall}(i, j) \approx \frac{logK}{K}$, under the assumptions stated above. Thus, the probability that a team member is able to upload to all other team members at time t is $P(n_T = n_{T_{max}}) \approx (1 - \frac{logK}{K})^{n_{T_{max}}}$. A team member performs an optimistic unchoke when it cannot serve all other team members simultaneously. This occurs with probability

$$\begin{aligned} P_T &= P(n_T \neq n_{T_{max}}) \cdot P_0 \frac{n_s - \overline{n}_T - 1}{n_s - 1} \\ &\leq P(n_T \neq n_{T_{max}}) \cdot P_0 \\ &\approx (1 - (1 - \frac{logK}{K})^{n_{t_{max}}}) \cdot P_0 \end{aligned} \tag{3.3}$$

where $\overline{n}_T$ is the average value of n_T. Therefore, as the number of content pieces increases, $lim_{K \to \infty} P_T \to 0$. □

Note that, even for smaller values of K, the probability P_T for a team member to perform an optimistic unchoke is still close to 0. For instance, a typical piece size in BitTorrent is 256 kB and a typical value for n_s is 4, which would make $n_{T_{max}} = 3$. In addition, P_0 is typically 0.25. Given these parameters, a 100 MB file will consist of 400 pieces, giving us $P_T \leq 0.015$. Similarly, for a 1 GB file, $P_T \leq 0.002$.

3.3.2 The Game

Having proven that the probability to perform optimistic unchokes by team members is very low, we now turn to incentives.

Definition. We describe the process of distributing a given content as an infinitely repeated game with rational participants and punishment [OR94]. The players (leechers) repeatedly engage in rounds and choose their actions simultaneously. A player can follow one of the following four strategies.

1. **An independent peer:** a contributing leecher who does not belong to a

team.

2. **A free-rider:** an independent peer who does not upload data to others.
3. **An honest team member:** a leecher who joins a team and follows the protocol rules, e.g., reports its correct upload bandwidth to the tracker.
4. **A dishonest team member:** a player who joins a team, but does not follow the protocol rules, e.g., lies about its upload capacity, does not upload to its team members, etc.

Leecher Utility Function. We define the utility function of leecher i to be the average download rate from all other leechers, $U_i = \overline{d}_i$. Note here that there should also be a term expressing downloads from the seeds. For instance, for seeds follow a round-robin unchoking strategy, this term would be a constant. As this term does not depend on the existence of teams, we can safely ignore it for the rest of the analysis.

Fan *et al.* [FCL06] have already expressed the leecher utility function for the regular BitTorrent protocol as a combination of the regular (selective, as they call it) and the optimistic (non-discriminative) unchoke policy. Our protocol adds yet another term due to the team unchoke policy. Thus, the utility function of a leecher i will be

$$\overline{U}_i = \overline{d}_i(regular) + \overline{d}_i(optimistic) + \overline{d}_i(team) \tag{3.4}$$

As in regular BitTorrent, peers distribute their uploading capacity uniformly over all their unchoke slots. Due to the way team formation works, and given a credible punishment mechanism inside a team, we can assume that all team members have a similar uploading capacity u_i. In addition, we assume that a leecher's downloading capacity is greater than or equal to its uploading capacity, which is typically the case in real torrents. Each team member will then be served with approximate probability $(1-\frac{logK}{K})$ by each of the other team members ($T-1$ in total) and will enjoy a download rate of $\frac{u_i}{n_s}$ from each team member. As a result, the team policy term will be

$$\overline{d}_i(team) \approx (1-\frac{logK}{K}) \cdot (T-1) \cdot \frac{u_i}{n_s} \tag{3.5}$$

Let us define

$$P_{dt} = (1-\frac{logK}{K}) \cdot (T-1) \cdot \frac{1}{n_s} \tag{3.6}$$

P_{dt} not only describes the download rate a leecher enjoys from its team members, but also represents the percentage of upload bandwidth it spends on those team members.

Fan *et al.* [FCL06] have shown that, for regular unchokes, the download bandwidth of a leecher is approximately equal to its upload bandwidth, i.e., $d_i \approx u_i$. The percentage of the upload bandwidth dedicated to regular unchokes is $1 - P_{dt} - P_T$. Therefore, the regular unchoke term will be

$$\overline{d}_i(regular) \approx (1 - P_{dt} - P_T) \cdot u_i \tag{3.7}$$

From Equations 3.4, 3.5, and 3.7, the utility function of team member i will therefore be

$$U_i \approx (1 - P_T) \cdot u_i + \overline{d}_i(optimistic). \tag{3.8}$$

On the other hand, for a free-rider who does not upload anything and does not belong to a team,

$$U_{FR} = \overline{d}_{FR}(optimistic). \tag{3.9}$$

Also, the utility function of an independent leecher will be

$$U_{IL} \approx (1 - P_0) \cdot u_i + \overline{d}_{IL}(optimistic). \tag{3.10}$$

Note that the optimistic unchoke yield $\overline{d}$(optimistic) is not sensitive to the strategy of the peer, but is based on random selection among its associated peers. However, as the number of associated peers joining teams increases, this value decreases since less optimistic unchokes are performed by the associated peers.

The detection and punishment mechanism presented in the previous section can be used to prevent free-riding inside a team. More specifically, punishment was defined as expelling a leecher from a team and preventing it from joining any other team for a fixed number of R rounds (rechoke periods). R should be long enough to decrease a free-rider's utility and therefore make the punishment credible as a threat. In particular, the utility of a dishonest team member should always be lower than that of a contributor. To examine whether this is true in our model, we define the following variables. d_T is the download rate of a leecher due to its participation in a team during a single round, $U_i[0..R]$ is the cumulative utility from time 0 to time R, and $\delta \in [0, 1]$ is a *future discount factor*, which expresses how important future contributions from others are to the utility of a leecher. (e.g., the closer δ is to 0 the less future contributions are valued). In our case, δ should be close to 1, as every piece of the content carries the same importance. Therefore, based on the folk theorem [OR94], in order for the punishment to be credible, the following equation should be satisfied.

$$\begin{aligned}
&U_i(\textit{dishonest team member})[0..R] \\
&= \frac{1}{R+1}(d_T + \sum_{j=1}^{R} \delta^j((1-P_0)\cdot u_i + \overline{d}_i(\textit{optimistic})) \\
&< \frac{1}{R+1}\sum_{j=0}^{R} \delta^j((1-P_T)\cdot u_i + \overline{d}_i(\textit{optimistic})) \\
&= U_i(\textit{honest team member})[0..R] \qquad (3.11)
\end{aligned}$$

Thus, a credible punishment will always lead to a lower utility for a dishonest team member compared to an honest contributing peer.

Nash Equilibrium.

Theorem 3.3.2. *The dominant strategy of a rational leecher in a team-aware BitTorrent system is to be an honest team member. Therefore, there is a unique Nash Equilibrium (NE) when all team members follow the honest strategy.*

Proof. In order to prove this, we need to show that the utility of an honest team member is always higher than with any other strategy. From Equation 3.3 we can see that for values of $K \geq 3$, $P_T < P_O$ with high probability. Therefore, the utility of an independent leecher i will always be lower than the utility of an honest team member $U_i(IL) = (1-P_0)\cdot u_i+\overline{d}_i(\text{optimistic}) < (1-P_T)\cdot u_i+\overline{d}_i(\text{optimistic}) = U_i(\text{honest})$.

Moreover, we assume that the punishment mechanism constitutes a credible threat and must therefore satisfy Equation 3.11. Therefore, the utility of a dishonest team member will also be lower than the utility of an honest one.

Finally, the utility of a free-rider is always lower than the utility of an honest team member since $U_i(FR) = d_i(\text{optimistic}) < (1 - P_T) \cdot u_i + \overline{d}_i(\text{optimistic}) = U_i(\text{honest})$.

We can thus conclude that the dominant strategy in the game is to be an honest team member, and that there is a unique NE when all leechers follow the honest team member strategy.

□

3.4 Implementation

The analytical model gives us confidence that teams will indeed limit free-riding in a real BitTorrent system. To examine this claim, we have modified existing open-source BitTorrent client and tracker implementations to realize the team protocol. This section discusses interesting aspects of our prototypes and justifies our protocol parameter choices.

3.4.1 Client

Our team-aware BitTorrent client is based on Enhanced CTorrent, version 3.2 [Enh]. We added code to enable the client to run in *team mode*, in which it follows the honest team member strategy. The client can also run in *regular mode*, in which it simply behaves as an independent peer, or in *free-riding mode*, where it remains independent and does not upload any data at all. We also added functionality for a team member to maintain state about its team, including the team identifier and other members' uploading history. We have also extended the BitTorrent peer protocol with the following messages.

- *join-team request*, sent by a leecher to the tracker in order to join a team (contains the leecher's upload bandwidth)
- *join-team response*, sent back by the tracker (contains the team identifier and the list of team members)
- *join-team announcement*, sent by a prospective leecher to its team members
- *join-verification request*, sent by a team member to the tracker to verify the joining of a prospective leecher (contains the leecher's identifier and the team identifier)

- *join-verification response*, sent by the tracker back to a team member verifying the prospective leecher is allowed to join the team (contains the leecher's identifier)

All in all, we added 988 non-comment lines of code to Enhanced CTorrent, while we did not need to remove or modify any of the existing code.

For our experiments presented in the next section, we use the default client parameters, unless otherwise specified. Enhanced CTorrent uses a 10-second rechoke period, and optimistic unchokes are rotated every three rechoke periods. The number of unchoke slots n_s has a minimum value of 4, from which one is always devoted to optimistic unchokes in regular mode. When running in team mode, the number of optimistic unchokes is determined by our peer selection algorithm. In particular, the optimistic unchoke is performed with probability given by Equation 4.1, and is decided every three rechoke periods.

Although fully functional, our prototype currently has certain limitations. Most notably, the detection and punishment mechanism described in Section 3.2.3 is still under development. This mechanism is designed to limit the benefit free-riders can get from joining a team by selecting appropriate parameter values, such as the execution intervals for each step and the period a peer is blacklisted. Determining the optimal values for these parameters is the subject of ongoing work.

3.4.2 Tracker

Our tracker is based on BNBT EasyTracker, version 7.7r3 [BNB]. We augmented the message classifier inside the tracker to support the new protocol messages. We also added functionality to manage the information for all teams in the system, which includes the team identifier, list of team members, and additional data, such as members' upload bandwidth. All in all, we added 522 non-comment lines of code and did not remove or modify any existing code.

Since Enhanced CTorrent uses a minimum of 4 unchoke slots, the tracker sets the maximum team size T to 4 as well, for all teams in the system. To select the value of the range factor x, we use an empirical peer upload capacity distribution derived from measurements of real BitTorrent swarms [PIA07]. Given the size of the network N, the maximum team size, and the upload capacity distribution, we can calculate x based on the desired p_f, the percentage of peers that will be members of full teams (for multiple-tracker swarms, the trackers could periodically communicate with each other to collaboratively calculate the size of the network).

For the team protocol to work well, the tracker should aim to keep p_f as high as possible. For instance, for $T = 4$, and $N = 500$, by setting $x = 0.05$, we get $p_f = 93\%$. for N=1000, we get $p_f = 94\%$ by setting $x = 0.03$, while for N=200, we get $p_f = 93\%$ by setting $x = 0.09$. In our experiments with 51 peers, we set $x = 0.1$, which results in $p_f = 90\%$. Therefore, within a given team, the slowest member was at most slower by 10% than the fastest member.

3.5 Experimental Evaluation

We now turn to real experiments to validate the properties of the team protocol. We first describe our experimental methodology and then discuss the results.

3.5.1 Methodology

We ran all our experiments on the PlanetLab experimental platform [BBC], utilizing nodes spread across the globe. These nodes are typically not behind NATs or firewalls, so these effects are not represented in our results, although we do not believe the conclusions we draw are predicated on that fact. We always execute all runs of an experiment consecutively in time on the same set of machines.

We ran extensive sets of experiments with content of different sizes, different numbers of leechers (24, 48), different numbers of free-riders (4, 8, 16, and 24), different leecher upload rates (20kB/s, 50kB/s, and 100kB/s), and different percentages of peers belonging to teams. We report here representative results for a 71 MB file divided into 284 pieces, each of size 256 kB (making $K = 284$), which demonstrate our main conclusions. Unless otherwise specified, we use the default CTorrent client parameters.

All leechers start the download process at the same time, emulating a flash crowd scenario, and leave the network once they have downloaded the entire content. The initial seeds stay connected for the duration of the experiment. The available bandwidth of PlanetLab nodes is relatively high for typical BitTorrent peers. We define upload limits on the leechers and seeds to model more realistic scenarios, but do not define any download limits to observe the effect of our protocol on peers' download rates.

3.5.2 Results

We look at comparative results for contributing leechers' and free-riders' performance, in order to validate the hypotheses stated in Section 4.1. We also examine system robustness when increasing the number of free-riders. Finally,

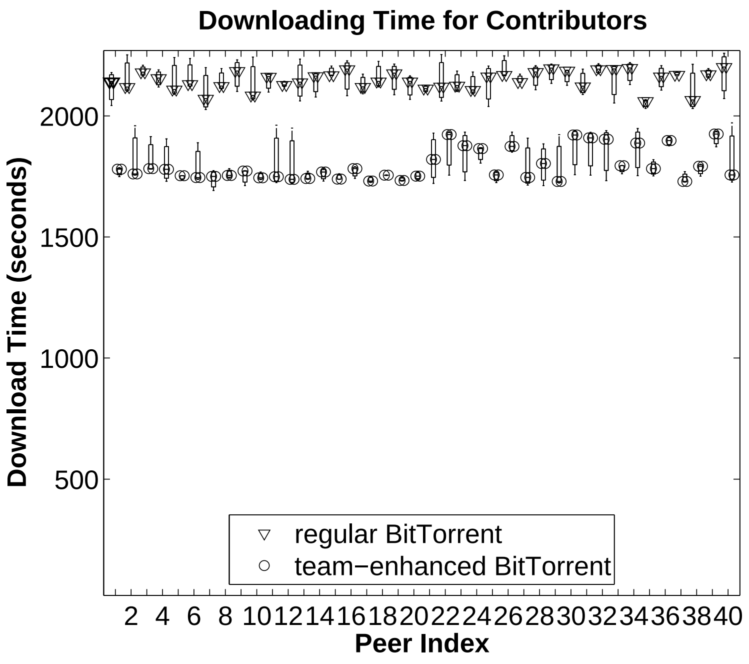


Figure 3.2: Download completion time for contributors

we additionally test our protocol using an empirical BitTorrent upload capacity distribution [PIA07].

3.5.2.1 Contributor Performance.

Figure 3.2 shows leechers' download completion time both in regular BitTorrent and in a $team - enhanced$ system where all leechers join teams of size 4. This experiment involves 50 PlanetLab nodes, 2 seeds and 48 leechers, 8 of them are free-riders. All contributing leechers have 50 kB/s upload bandwidth. For each leecher, we draw two boxplots [RL78] corresponding to the two scenarios. The top and bottom of each box represent the 75th and 25th percentile download rates over all 5 runs of the experiment. The marker inside the box is the median, while the vertical lines extending above and below represent the maximum and minimum values respectively (within the high and low ranges extending 1.5

times the box height from the box boundaries). Potential outliers are marked individually.

We observe that the *download time for all team members improves by* 10% – 26%, as measured by the median. Downloading from team members consistently enables a leecher to maintain high download rates.

Looking at the amount of uploaded data for each contributor leecher in the two scenarios, we observe that 35 of the contributors upload less when they are in teams. In particular, 32 of them upload at least 10% less data. This is because team members' download time decreased in comparison to regular BitTorrent, thus they uploaded data to the system for a shorter period of time.

We also performed experiments where only a fraction of contributing leechers belong to teams and the rest are independent. As expected, team members in these experiments did better when the percentage of independent peers increased, while independent peers did slightly worse. This is because team members spend their upload bandwidth mainly on other team members, while independent peers receive less data via optimistic unchokes.

In summary, we see that *the team protocol provides a clear incentive for contributing leechers to join teams*, since a team member achieves improved performance compared to an independent one, while at the same time limiting the amount of data it needs to upload to others. In addition, this improvement would be even more pronounced with larger content, which is typical for BitTorrent swarms. Larger content would entail a higher number of pieces, which, according to Equation 3.3, will lead to an even lower probability of optimistic unchokes.

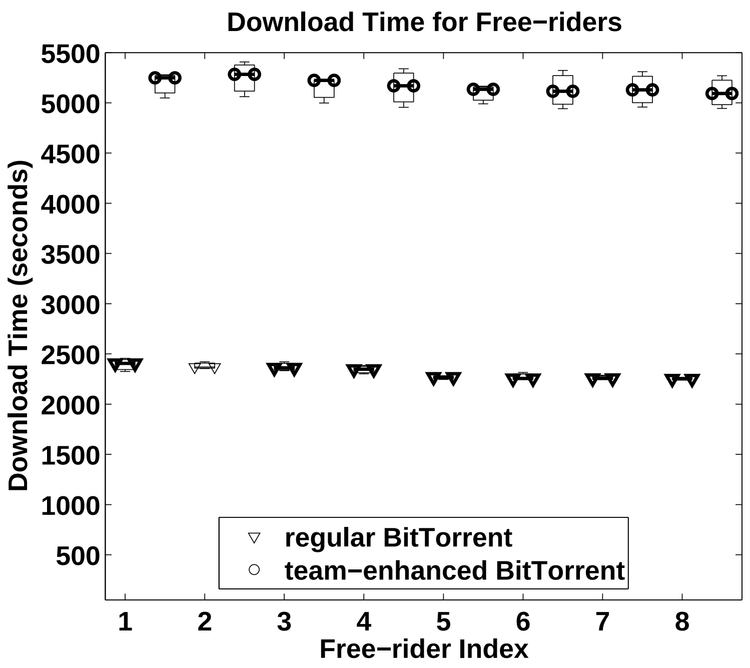


Figure 3.3: Download completion time for free-riders

3.5.2.2 Free-rider Performance.

Figure 3.3 shows the download completion times for free-riders in the same experiment, for the two scenarios. We observe that *the existence of teams slows down free-riders by more than 100%.*

This is because free-riders can no longer take advantage of optimistic unchokes and thus have to depend mainly on the two seeds for downloading data. Indeed, Figure 3.4, which plots the percentage of content free-riders download from seeds, validates this hypothesis. This percentage increases from $\sim 10\%$ in regular BitTorrent to $\sim 60\%$ in a $team - enhanced$ system.

Interestingly, our analytical model predicts an even higher percentage of download from seeds, closer to 100%. This is not the case here since the model only considers a system in its steady state, while, in reality, there are warm-up and

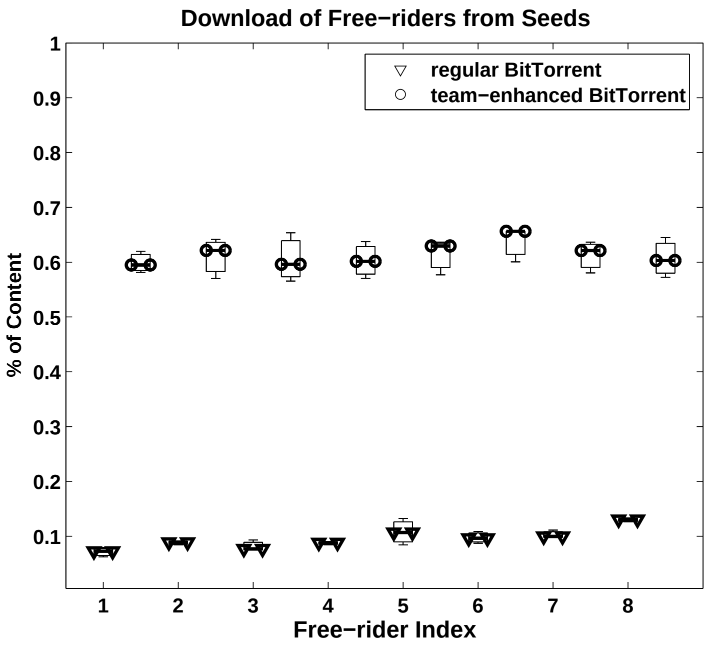


Figure 3.4: Content free-riders download from seeds

endgame periods, during which contributing leechers may perform some optimistic unchokes, since they are not able to upload to all their team members at the same time.

Figure 3.5 explains this more accurately by plotting the ratio of the probability of optimistic unchokes by team members in a $team - enhanced$ system over the same probability in regular BitTorrent ($\frac{P_T}{P_0}$), as given by Equation 4.1, for all 10-second intervals (the rechoke period) during the download. This value is averaged over all 5 runs for all 40 contributor leechers. In addition, the dotted line represents the percentage of contributors still present in the system at that point in time. We observe that the ratio is lower than 0.1 (an average of 6.7%) between the 40th and 130th time unit, indicating that very few optimistic unchokes are performed by team members in that period. On the other hand, there is clearly a higher probability for optimistic unchokes by team members during the warm-up

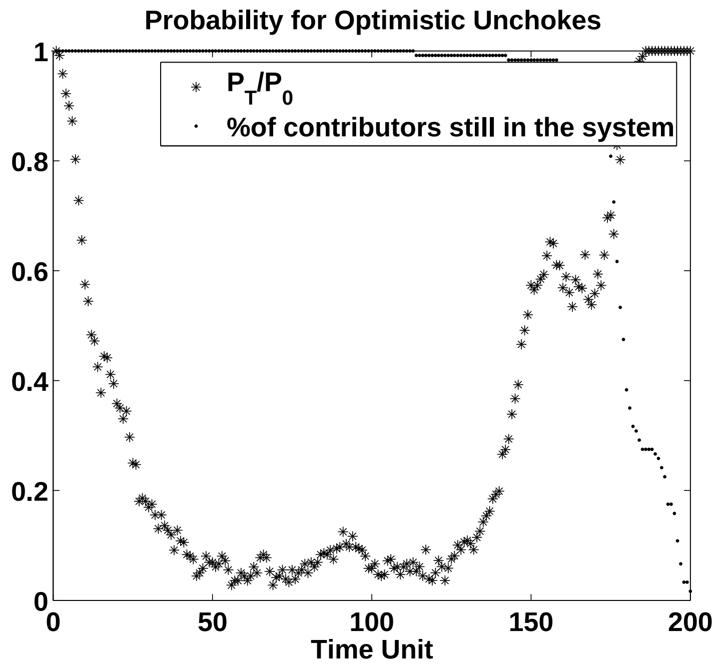


Figure 3.5: $\frac{P_T}{P_0}$ probability ratio

and endgame periods.

3.5.2.3 System Robustness.

Robustness is the degree to which a system is able to deliver good service despite the presence of free-riders. In a robust system, increasing numbers of free-riders should not be able to significantly affect the performance of contributing leechers.

Figure 3.6 plots the performance of contributing leechers with different numbers of free-riders, when all the contributors belong to teams, and in regular BitTorrent, when all are independent. All other experiment parameters remain the same, including the total number of leechers (48). We observe that contributors perform noticeably better ($\sim$ 15% improvement) when they run as team members. Interestingly, the difference in performance is more pronounced for

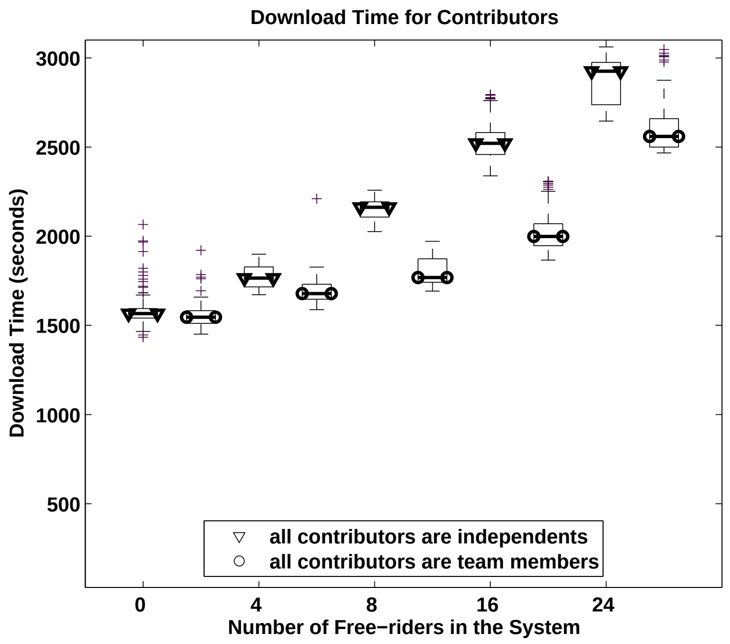


Figure 3.6: Download completion time for contributors in the presence of different numbers of FRs

higher numbers of free-riders. Therefore, *a team − enhanced system appears to be more robust than regular BitTorrent*, especially in the presence of a high percentage of free-riders in the network.

Figure 3.7 examines free-rider performance. Although we expect their performance to deteriorate as their percentage increases even in regular BitTorrent, due to the decreased total upload capacity in the system, we observe that free-riders suffer considerably more in the presence of teams. For instance, when 16 of the leechers are free-riders, their download time slowed down by 29% in comparison to regular BitTorrent.

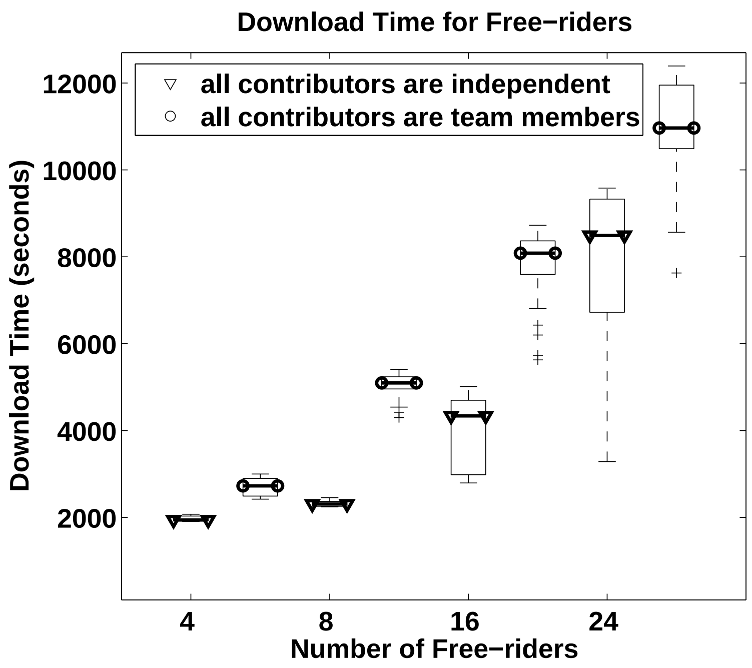


Figure 3.7: Download completion time for free-riders as their number increases

3.5.2.4 Empirical Upload Capacity Distribution.

In another set of experiments, we used a scaled upload capacity distribution observed in real BitTorrent systems, as presented in [PIA07]. This distribution was based on empirical measurements of BitTorrent swarms including more than 300,000 unique IP addresses. Our experiments included 51 PlanetLab nodes: 40 contributors with upload capacities drawn from the empirical distribution, 8 free-riders, and 3 seeds with combined capacity of 128 kB/s serving a 15 MB file. The other experiment settings remained the same. We compared the results of regular BitTorrent to a $team - enhanced$ system. In the latter, not all teams were full due to differences in peer upload rates.

Our results demonstrate that download times for 76% of the team members improved by up to 33%, as measured by the median of 5 experiment runs. The majority of the team members who did not improve their download time were the slowest leechers. In regular BitTorrent these peers would significantly benefit from optimistic unchokes, since their upload rate is very slow compared to that of fast contributor leechers. Regarding free-riders, we observed that the download time for all of them increased by up to 25%.

CHAPTER 4

Buddy Incentives in BitTorrent Systems

4.1 Overview

The BitTorrent protocol aims to provide fairness among peers by applying the tit-for-tat (TFT) unchoking mechanism to incentivize contribution and discourage free-riding. Despite this, previous studies [PIA07, BHP06, GCX05] showed that BitTorrent suffers from unfairness, particularly for high capacity leechers. Other studies [LNK06, LMS06, SPC07] showed that free-riding opportunity exists in BitTorrent.

In the previous chapter we proposed a BitTorrent-like protocol that discourages free-riding and thus improves the robustness of BitTorrent-like system. The proposed protocol is controlled by the tracker, which manages additional tasks such as forming teams, controlling punishment of team members, etc.

In this chapter we propose another alternative protocol. However, the difference is that in this chapter the protocol is distributed, thus, requiring changes only in the peers while requiring no additional tasks from the tracker.

The mechanism dynamically creates *buddies*—pairs of peers having similar upload bandwidth that collaborate for mutual benefit. The buddy relation is based on the upload history during the existence of the relation, and not only on the last round of contribution as in regular BitTorrent's TFT mechanism.

Similar to the team-enhanced protocol, peers in buddy mode aim to satisfy their download needs through their buddies and perform optimistic unchokes only if absolutely necessary. Thus, the buddy mechanism mostly obviates the need for optimistic unchokes, while at the same time improves fairness and system robustness.

We implemented our buddy protocol on top of an existing BitTorrent implementation. We ran experiments with different configurations on controlled PlanetLab testbeds in order to evaluate the impact of the buddy protocol on different level of contributing leechers and free-riders in comparison to the regular BitTorrent. In the experiments, our buddy protocol exhibited the following properties:

1. It promotes fairness as high capacity leechers who suffer from the unfairness in regular BitTorrent improve their download rate, and low capacity leechers who used to benefit from this unfairness are slowed down.

2. It provides a clear incentive for all levels of contributing leechers to adopt it, even to those low capacity leechers who are slowed down due to it.

3. It discourages free-riding by limiting the optimistic unchokes, in comparison to the regular BitTorrent, thus directly hurting free-riders' performance.

4. It improves the system robustness as increasing free-riding has less impact on the contributors' performance, in comparison to the regular BitTorrent.

5. It converges faster to the steady state unchokes peer selection, and minimizes the peer selection fluctuations, in comparison to the regular BitTorrent.

We have developed an analytical model that explains leechers' interactions in the presence of the buddy protocol and justifies our design choices. Based on this model and game theory, we have proven the existence of Nash Equilibrium when all the contributing leechers adopt the buddy protocol.

The rest of this paper is structured as follows. In section 4.2 we present the design of our $buddy - enhanced$ protocol, while in section 4.3 we describe the analytical model that has guided this design. Implementation details are discussed in Section 4.4. Finally, the results of our PlanetLab experiments are presented in Section 4.5.

4.2 Design

We now turn to describe the design of our buddy protocol, which augments BitTorrent with the notion of *buddies*—pairs of peers having similar upload capacity, collaborating for mutual benefit. The main goals of the protocol are reducing the unfairness from which BitTorrent suffers, particularly for high capacity peers, and discouraging free-riding. The protocol achieves these goals via explicit cooperation, as well as the tit-for-tat choking mechanism already employed by BitTorrent. In addition, it significantly limits the optimistic unchokes, where high capacity clients are forced to peer with those with low capacity while searching for better peers, an acknowledged opportunity for free-riding in BitTorrent [SPC07].

The protocol comprises three main parts:

1. The formation process, which forms buddy relations;
2. The monitoring process, which monitors existing buddy relations;
3. The peer selection algorithm that peers use to decide who to unchoke.

We describe these in turn.

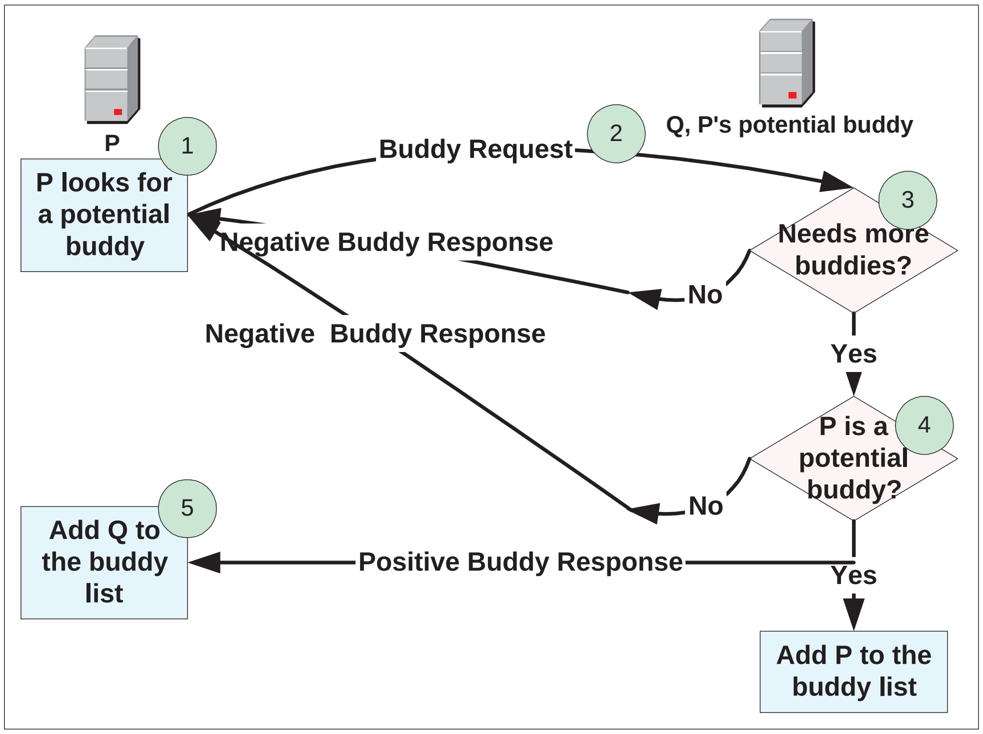


Figure 4.1: Buddy formation mechanism

4.2.1 Buddy Formation

A leecher P that runs in buddy mode is willing to maximize the number of buddies that have a similar upload rate as itself. Leecher P reserves an unchoked slot for each buddy to which it can upload data to in order to minimize buddy chokes, which can lead to termination of the buddy relation. At the same time, leecher P reserves one unchoke slot for a regular unchoke in order to avoid isolating a group of buddies from the rest of the swarm. Therefore, leecher P can have maximum $n_s - 1$ buddies, given that n_s is the number of unchoke slots in leecher P.

Figure 4.1 shows the buddy formation process. Leecher P runs this process once every rechoke period only if it has not reached the maximum number of buddies.

Step 1. Aiming to find a potential buddy with a similar upload as its own, leecher P estimates the upload bandwidth of peers with which it has interacted in the past, and that are currently not in its buddy group, using its own history about past download interactions with these peers. Notice that here leecher P looks at continuous history and not only at the last round as in the regular BitTorrent peer selection mechanism. If leecher P is not able to find a potential buddy Q due to its upload capacity being too high(low) as compared to the leechers with which it has interacted with, it may gradually increase(decrease) the number of unchoke slots n_s, while keeping maximum(minimum) n_s size constraints satisfied.
Step 2. Once leecher P has found a potential buddy Q, leecher P sends a buddy request to leecher Q.
Step 3. Upon receiving a buddy request from leecher P, leecher Q checks that it has not reached the maximum number of buddies, otherwise it sends a negative buddy response.
Step 4. If leecher Q can have more buddies, it estimates leecher P's upload capacity by using history about past interactions with leecher P. If it finds that leecher P has a similar upload capacity as its own, leecher Q sends a positive buddy response to leecher P and tags leecher P as its buddy. Otherwise it sends a negative buddy response to leecher P.
Step 5. Upon receiving a positive buddy response from leecher Q, leecher P tags leecher Q as its buddy.

The negotiation between leecher Q and leecher P (Steps 2-4) is important for stability and fast convergence of the peer selection mechanism. However, it requires both sides to support the $buddy - enhanced$ protocol, which cannot be assumed in practice. Hence, in case leecher P does not get any response from leecher Q, leecher P assumes that leecher Q does not support the buddy protocol. Even though leecher Q does not support the protocol, leecher Q can still match the requirements to be a buddy, thus in such case leecher P can add leecher Q to its buddy list.

In order to keep the design simple, the look-up for buddies does not require a view of all the network peers (as required in the design that was suggested in the previous chapter); thus, we expect the improvement to be sub-optimal.

4.2.2 Monitoring Buddies

During every rechoke period, a peer in buddy mode checks the upload rate of its buddies. It looks separately at the history of each buddy since the buddy connection was established (more specifically, it compares the total upload rate and the total download rate since the buddy connection was established), and checks that the estimated upload rate of the buddy remains similar to its own. If this is not the case, the peer removes the buddy that uploads slower than expected from its buddy list. Thus, the longer the connection, the more reliable it is in comparison to the regular unchoke mechanism. A buddy cannot gain much from cheating by uploading less than agreed, since it will be caught and disconnected quickly. Thus, it will not be able to download much before being caught, and will lose an established buddy connection. Notice that a stronger punishment can be considered by keeping the cheating buddies' history and banning them from download or have a buddy relation for a period of time.

4.2.3 Peer Selection Decisions

During the uploading process, a leecher decides periodically which leechers to unchoke. In regular BitTorrent protocol, this peer selection decision is made using two separate mechanisms (1) the regular or tit-for-tat unchoke mechanism, which guides a leecher to unchoke those peers that upload data to the given peer at the highest rates; and (2) the optimistic unchoke mechanism, which selects the unchoked peers at random, independent of their contributions.

In addition to these two mechanisms, our buddy protocol extends the peer selection algorithm design with a third - the *buddy unchoke* mechanism - which, like the team protocol, dictates that a leecher always unchokes all its buddies. The rationale is that a leecher that has buddies should opt for uploading to its buddies whenever possible, expecting its buddies to do the same. Since buddies have similar upload rates, this then guarantees similar upload as download rates. At the same time, it performs at least one regular unchoke to a peer that is not its buddy to avoid isolating the group of buddies from the rest of the swarm. Uploading via optimistic unchokes may happen only when a leecher has no data, other buddies are missing, or it does not reach the maximum number of buddies. The unchoke decision algorithm is very similar to the one that was presented in the team protocol, only that now instead of considering behaviors of team members, here buddies' behaviors are considered. The unchoke decision algorithm that combines these three mechanisms is as follows.

Step 1 (buddy unchokes). A leecher in buddy mode strives to satisfy as many of its buddies as possible. We denote the number of unchoke slots reserved for buddies as a function of time with $n_B(t)$, where $0 \leq n_B(t) \leq n_s - 1$, $n_B(t)$ is bounded by $n_s - 1$ devoting at least one slot for regular unchokes. If a peer is *independent*, thus, does not have buddies, $n_B(t) = 0$.

Step 2 (optimistic unchokes). Given that P_0 is the probability to perform optimistic unchokes in regular BitTorrent, a leecher in buddy mode performs optimistic unchokes with probability:

$$P_B(t) = P_0 \frac{n_s - n_B(t) - 1}{n_s - 1} \tag{4.1}$$

Depending on the implementation, P_0 can be fixed or changes dynamically as being calculated at runtime using heuristics.

Step 3 (Regular unchokes). $n_O(t)$ denotes the number of optimistic unchoke slots. Thus, for the remaining $(n_s - n_B(t) - n_O(t))$ unchoke slots, a leecher selects those peers not selected already who have the highest uploading rates.

Note that for independent leechers that run the regular BitTorrent $P_B = P_0$, because $n_B = 0$. However, P_B decreases for buddies as more buddies are served. Those peers who serve $n_s - 1$ buddies perform no optimistic unchokes ($n_B = n_s - 1 \rightarrow P_T = 0$). They do not need to continue peer discovery.

For this peer selection algorithm design, we expect to have the following effects:

1. High capacity buddy peers' performance should be improved, as there are now other high capacity peers who make it their priority to serve them.

2. Reduction in the total number of optimistic unchokes, thus directly hurting the free-riders, who critically depend on these unchokes to obtain data.

3. Less fluctuations in peer selection due to the buddy mechanism.

These hypotheses have been validated by our experimental evaluation results. In a sense, *BitTorrent peer selection mechanisms are partially replaced by buddies* as the means of discovering better partners. Hence, buddies no longer need to explore the network as much, having the guarantee of other leechers with similar upload capacity willing to upload to them.

4.3 Analytical Model

This section presents an analytical model that guides the protocol design presented in the previous section. It describes peer interactions in the presence of buddies as a game with rational players, and shows that there is a Nash-Equilibrium when all the peers in the system are contributing buddies.

4.3.1 The Game

We model the process of distributing given content between incentive leechers in the P2P system as an infinitely repeated *game* [OR94] with rational participants. To simplify this file sharing process, we do not consider a long-term punishment, although this is a design issue that is part of future work. In the model the players repeatedly engage in rounds. In each round they choose their actions simultaneously, where the actions are determined by the players' strategy choices. To keep the model simple, we define only three strategies. A player can choose one of the three following strategies.

1. **An independent peer:** a contributor who runs the regular BitTorrent
2. **A free-rider:** a player who does not upload data to others
3. **A buddy-aware peer:** a player who follows the $buddy-enhanced$ protocol

4.3.2 Leecher Utility Function.

The leecher utility function is defined as the average download rate from all other leechers, thus leecher i's utility function is $U_i = \overline{d}_i$. We will ignore the download from the seeds in the analysis since a seed upload process does not depend on the existence of buddies.

As was mentioned before, Fan *et al.* [FCL06] have already expressed the leecher utility function for the regular BitTorrent protocol as a combination of the average download rate that is obtained through the regular unchoke and the optimistic unchoke mechanisms. Similar to the team-enhanced protocol, the $buddy - enhanced$ protocol also adds yet another term for the average download rate that is obtained through the buddy unchoke policy. Thus, the utility function of a leecher in the $buddy - enhanced$ system will be

$$\overline{U}_i = \overline{d}_i(regular) + \overline{d}_i(buddies) + \overline{d}_i(optimistic) \tag{4.2}$$

We assume that all peer i's buddies have a similar upload rate to peer i's $\frac{u_i}{n_s}$, given that u_i is peet i's upload rate. That is due to the way the buddy connections are created and maintained. In addition, we assume that the download bandwidth of a leecher is at least as much as its own upload bandwidth; this is typically the case in real torrents.

Each peer that has n_B buddies enjoys a download rate of $\frac{u_i}{n_s}$ from each of its buddies. As a result, the buddy policy yield will be

$$\overline{d}_i(buddy) \approx n_B \cdot \frac{u_i}{n_s} \tag{4.3}$$

$P_{db} = \frac{n_B}{n_s}$ describes the download rate a peer enjoys from its buddies. It also represents the percentage of upload bandwidth it spends on those buddies.

As has been mathematically shown in [FCL06], the download bandwidth of a leecher is approximately equal to its upload bandwidth for regular unchokes, thus, $d_i \approx u_i$. The percentage of the upload bandwidth dedicated to regular unchokes is $1 - P_{db} - P_B$. Thus, the regular unchoke yield will be

$$\overline{d}_i(regular) \approx (1 - P_{db} - P_B) \cdot u_i \tag{4.4}$$

Therefore, from Equations (4.2), (4.3), and (4.4), the utility function of buddy i is:

$$U_i \approx (1 - P_B) \cdot u_i + \overline{d}_i(optimistic). \tag{4.5}$$

The utility function of a free-rider who uploads nothing is:

$$U_{FR} = \overline{d}_{FR}(optimistic) \tag{4.6}$$

The utility function of an independent peer who runs the regular BitTorrent is:

$$U_{IL} \approx (1 - P_0) \cdot u_i + \overline{d}_{IL}(optimistic). \tag{4.7}$$

Note that since the download through optimistic unchokes can be received from any peer in peer i's peer set as it is based on random selection among all peers, the optimistic unchokes yield is not sensitive to the strategy choice of peer i. Hence, the unfairness that is due to optimistic unchokes in the regular BitTorrent is a function of the upload capacity and P_0. This is evident when the amount of upload that is given to optimistic unchokes is not the same as the amount of download that is received through optimistic unchokes, i.e., $P_0 \cdot u_i \neq d_i(optimistic)$.

In the $buddy - enhanced$ system, the unfairness is a function of the upload capacity, and P_B. P_B gets closer to 0 as the number of buddy connections increases, which leads to a reduction in optimistic unchokes upload. Thus, peers will benefit more by maximizing the number of buddy connections. Moreover, increasing the number of buddy connections in the system will reduce the average download that is received through optimistic unchokes. As for the system, if every peer will choose the buddy-aware strategy, thus minimizing P_B and devoting less upload to the unfairness inequality, the unfairness in the system will decrease.

4.3.3 Nash Equilibrium.

Theorem 4.3.1. *The game in a $buddy - enhanced$ system model has Nash Equilibrium when all the rational leechers in the game are buddy-aware peers.*

Proof. In order to prove this, we need to show that the utility of a buddy-aware peer is not lower than any other strategy's utility. Thus, if all the peers choose the buddy-aware strategy, none of the peers has the incentive to change their strategy. Equation 4.1 shows that $P_B \leq P_O$. Therefore, $U_i(IL) \leq U_i$(buddy-aware). Both utilities are equal when the buddy-aware peer is not able to find any matching buddy through the whole downloading process, and when it has not increased its slot numbers due to buddy conditions. Otherwise, the utility of the buddy-aware peer will be always greater than the utility of an independent peer. A free-rider's utility will be $U_i(FR) \leq U_i$(buddy-aware). The utility of a free-rider will be equal to the utility of a buddy-aware i only in the case when all the peers in the network do not upload to i through the regular unchokes or the buddy unchokes policies during the whole downloading process.

We can thus conclude that the $buddy - enhanced$ system has a Nash Equilibrium when all the rational peers are buddy-aware peers.

□

4.4 Implementation

The analytical model shows that the $buddy-enhanced$ protocol, has the potential to reduce unfairness and free-riding in a BitTorrent system. In order to examine this claim, we modified an existing open-source BitTorrent client to implement our buddy protocol. This section discusses interesting aspects of our prototype and justifies our protocol parameter choices.

In order to apply the new $buddy - enhanced$ protocol we have implemented a new BitTorrent client. However, we were not required to make any changes in the existing BitTorrent tracker. Our $buddy - enhanced$ BitTorrent client was implemented on top of Enhanced CTorrent, version 3.2 [Enh]. The modified client can run in *buddy mode*, in *regular mode*, in which it simply runs the regular BitTorrent and behaves as an independent peer, or in *free-riding mode*, where it remains independent but does not upload any data at all. We have added functionality for a buddy peer to maintain state about its buddies such as buddies' uploading history. We have also extended the BitTorrent protocol with the following messages.

- *buddy request* is sent by a leecher to a potential buddy, asking the potential buddy to be its buddy.
- *buddy response* is sent back by the potential buddy. The response can be positive or negative.

We added in total 679 non-comment lines of code to the Enhanced CTorrent. We did not need to remove or modify any of the existing code.

The rechoke period in Enhanced CTorrent client takes 10 seconds, and optimistic unchokes are rotated every 3 rechoke periods. The number of unchoke slots is dynamic with a minimum of 4 slots. This number may increase if a client does not saturate its capacity. In buddy mode, this number might change also if the upload capacity of a peer is too low/high compared to the peers it interacts with. We did not limit the maximum number of slots.

In regular mode, one unchoke slot is always devoted to optimistic unchokes, while in buddy mode, our peer selection mechanism determines the number of optimistic unchokes. In particular, optimistic unchokes are performed with the probability given by Equation 4.1. The decision whether to perform an optimistic unchoke and, if so, which peer to optimistically unchoke, is taken every three rechoke periods. The chosen peer remains optimistically unchoked for the following three rechoke periods.

Before running in buddy-mode, a peer needs to collect information about the upload rate of itself and other peers in its peer set. The first three minutes (six optimistic unchoke periods) after a client starts uploading are used for that; only after this period the peer is able to start running in buddy mode.

4.5 Experimental Evaluation

We ran extensive experiments on a controlled testbed, validating our protocol properties. Here we discuss our experimental methodology and results.

4.5.1 Methodology

All our experiments were run on the PlanetLab experimental platform [BBC], utilizing nodes that are located across the globe. We executed all experiment runs consecutively in time on the same set of machines. Unless otherwise specified, we used the default BitTorrent client and seeding behavior.

All peers started the downloading process simultaneously, emulating a flash crowd scenario. The initial seeds stayed connected through the whole experiment. To provide synthetic churn with constant capacity, leechers disconnected immediately upon completion of downloading the entire file, and reconnected to the network as new leechers, starting the downloading process from the beginning. This ensures keeping the same network uploading distribution during the entire experiment.

We artificially constrained the upload capacity of the nodes according to the bandwidth distribution for typical BitTorrent leechers as presented in [PIA07]. This distribution is based on empirical measurements of BitTorrent swarms including more than 300,000 unique BitTorrent IPs. Since PlanetLab is often oversubscribed and shares bandwidth equally among competing experiments, not all nodes were capable of matching the required upload capacity drawn from the empirical distribution. Thus, we scaled by 1/20th the upload capacity and other relevant experimental parameters such as file size. We did not define any download limits.

Unless otherwise specified, the experiments hosted 104 PlanetLab nodes, 100 leechers and 4 seeds with combined capacity of 128 KB/s serving a 50 MB file.

4.5.2 Results

We looked at comparative results for leechers with different contribution levels in order to validate our design.

4.5.2.1 Leecher Performance - System without free-riders.

In this subsection we discuss three aspects that have been raised in this study: (1) Does the $buddy - enhanced$ protocol improve performance? (2) Does the $buddy - enhanced$ protocol indeed promote fairness? (3) Do all strategic leechers (having different upload capacity) have incentives to adopt the $buddy - enhanced$ protocol? Furthermore, do all/majority of the users need to adopt it in order to reach the protocol goals?

Figure 4.2 shows leechers' download completion times, comparing a regular BitTorrent system to a $buddy - enhanced$ system where all leechers run in buddy mode. For each group of leechers having similar upload capacity, we draw separate boxplots [RL78] for the different scenarios. The top and the bottom of the box represent, respectively, the 75th and the 25th percentile download rates over all 7 runs of the experiment. The marker inside the box represents the median, while

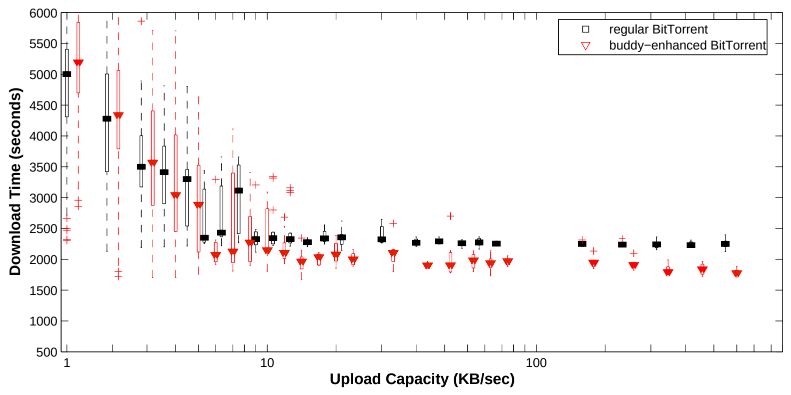


Figure 4.2: Download completion time for leechers

the vertical lines extending above and below the box represent, respectively, the maximum and minimum values within the ranges of 1.5 times the box height from the box border. Potential outliers are marked individually with a "+" sign.

We observed a clear difference between high capacity leechers, which are the fastest 42%, and low capacity leechers, which are the slowest 58%. High capacity leechers improved their download time tremendously. Leechers with upload capacity of at least 5kB/sec improved their download time by 8%-36% as measured by the median (15% improvement on average). This happened due to consistent downloading from leechers with similar upload rate, less optimistic unchoke performance, and an increased number of unchoke slots for high capacity leechers.

In regular BitTorrent, leechers provide one unchoke slot for optimistic unchokes during the entire downloading process. The leechers decide who to optimistically unchoke every optimistic unchoke period (30 seconds), and unchoke the chosen peer for this entire period. Thus, the probability of performing optimistic unchokes is always equal to 1. On the other hand, in the $buddy-enhanced$ protocol, in every optimistic unchoke period, the buddy leecher first decides if to perform an optimistic unchoke in the next optimistic unchoke period using the

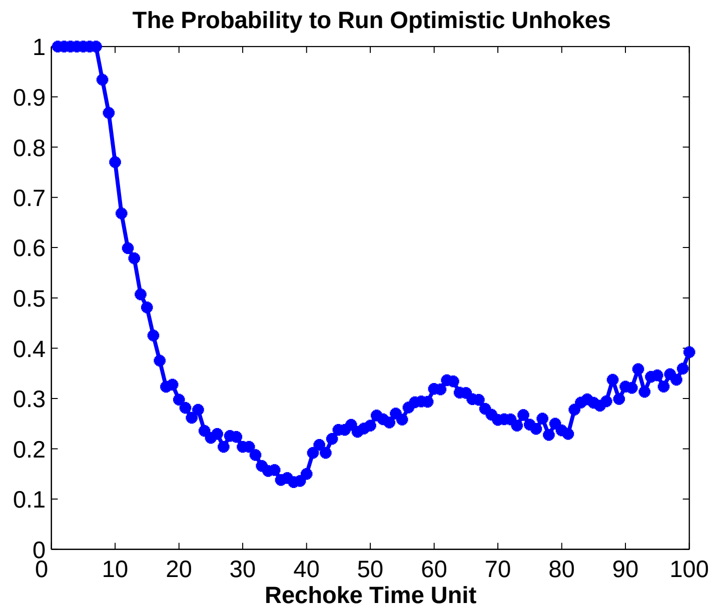


Figure 4.3: The probability for optimistic unchokes in $buddy - enhanced$ system

buddy peer selection mechanism that is described in Section 4.2.

Figure 4.3 shows the average ratio of optimistic unchokes performed in the $buddy - enhanced$ system as a function of optimistic unchoke time periods. The Figure confirms a substantial reduction of optimistic unchokes. More specifically, after ignoring the start-up period behavior (ignoring the first 510 seconds), we can see that leechers in the $buddy - enhanced$ system perform up to 86% fewer optimistic unchokes, with an average of 0.26 probability of performing optimistic unchokes.

Figure 4.4 confirms the increasing number of unchoke slots for the fastest 12% of leechers in the $buddy - enhanced$ protocol.

As a result of the aforementioned, low capacity leechers downloaded less from high capacity leechers. However, as opposed to the significant improved performance of high capacity leechers, low capacity leechers were slowed down by only

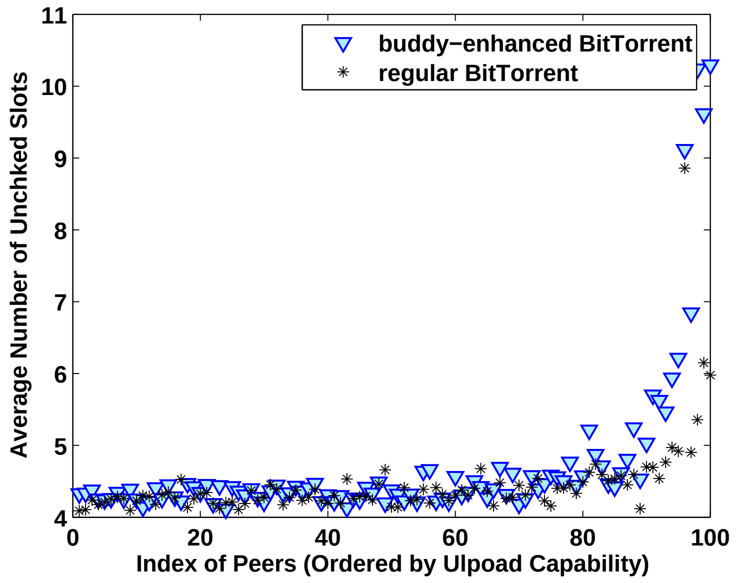


Figure 4.4: The average unchoke slots for leechers ordered by upload capacity

1.5% − 4% (as shown in Figure 4.2).

In summary, we see that *the buddy − enhanced protocol indeed promotes fairness and provides clear incentives for high capacity leechers to run in buddy mode*, as a high capacity buddy leecher achieves improved performance compared to an independent one. However, it is not clear that the low capacity leechers have the same incentives since their download time performance suffers.

Therefore, we ran another experiment where low capacity peers ran in independent mode and high capacity peers ran in *buddy−enhanced* mode. Figure 4.5, comparing our previous results with the "different strategy users" results, shows that by changing to an independent strategy, low capacity peers suffered a further performance reduction of 4% − 8%. This is because, as independents, they were also losing part of their bandwidth for optimistic unchokes without receiving anything in return. Moreover, the high capacity peers' performance (not shown here) was not sensitive to the changes, and were similar to the results presented

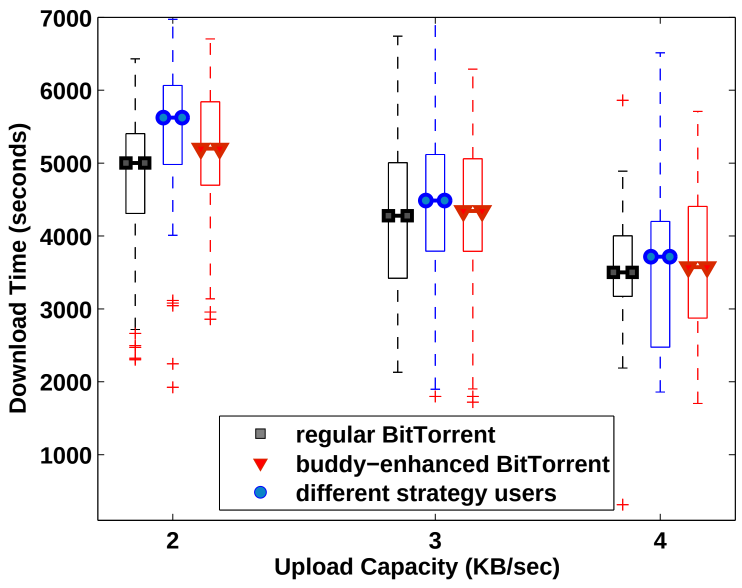


Figure 4.5: Download completion time for leechers

in Figure 4.2.

The experiments validate our hypothesis in section 4.3 showing that it is to the benefit of all the leechers to run in buddy mode. Furthermore, running in buddy mode motivates leechers to contribute and improves fairness in the network.

Single Buddy in the System

In addition, we examined the trivial case where only a single peer adopts the $buddy - enhanced$ protocol, while the rest of the peers in the network run with the regular BitTorrent (as this is a common case that is tested by other existing protocols such as [PIA07, LLS08]). Figure 4.6 compares the download time performance of a single $buddy - enhanced$ leecher with its performance when it runs the regular protocol. The Figure shows that the download time is improved for any single $buddy - enhanced$ peer. This is because a single

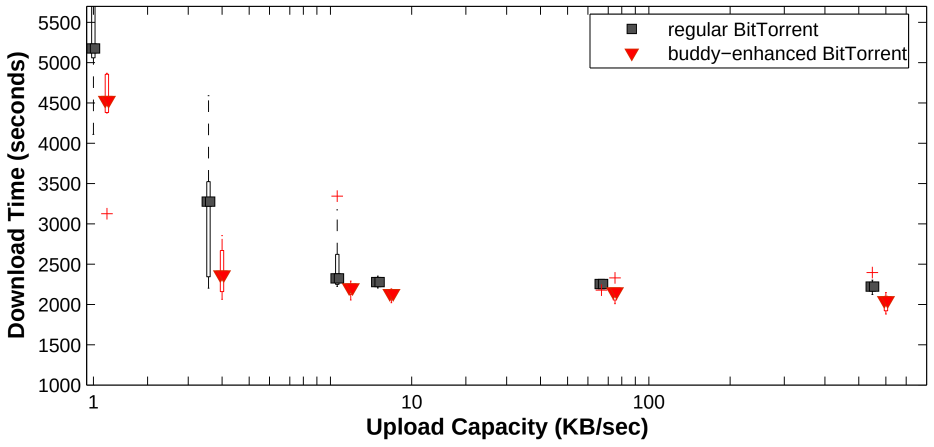


Figure 4.6: Leecher download time

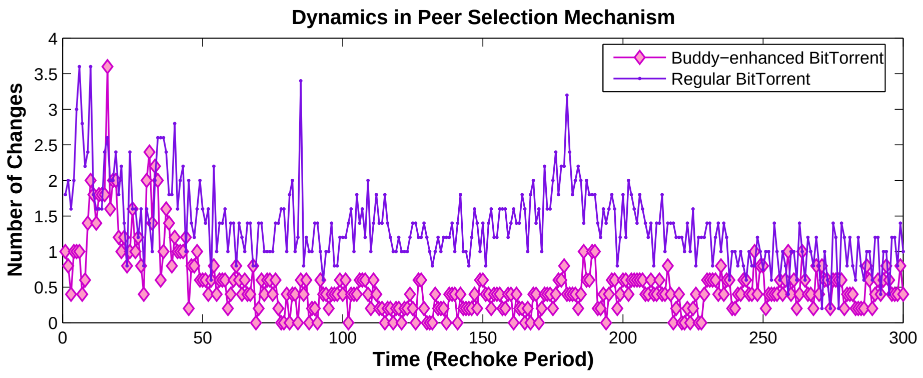


Figure 4.7: Peer selection mechanism dynamics

$buddy - enhanced$ peer continues to enjoy the download from its associate peers' optimistic unchokes, which are made independently to its peer selection strategy. This occurs while the single $buddy - enhanced$ peer limits its optimistic unchokes and instead uploads to peers that have a higher likelihood of reciprocating back at the same upload rate.

High-capacity peers in these experiments do not improve their download time as much as when the entire network is running the $buddy - enhanced$ protocol (Figure 4.2). This is because high-capacity peers perform optimistic unchokes with a probability of 0.65-0.9. As a single buddy in the network then, a buddy peer chooses its buddies without confirming collaboration (they run with the regular BitTorrent). Thus, the buddy connection here is more fragile and may hold for shorter term.

On the other hand, a low-capacity single-buddy peer significantly improves its download time when compared to its performance running as a regular BitTorrent peer. A single buddy limits its tit-for-tat response to optimistic unchoke uploads while replacing it with a response to buddy uploads. A Tit-for-tat response to high-capacity peers is a waste of upload for low-capacity peers as the high-capacity peers will not upload back. Moreover, in the given distribution, there are many slow peers with a similar upload rate. Thus, it is easier to find potential buddies, which leads to only a $0.1 - 0.29$ probability of performing optimistic unchokes.

Peer selection Mechanism Stability

We looked at the protocol peer selection stability. A stable peer selection mechanism should minimize the peer selection fluctuations. We measured peer selection fluctuations by comparing the peer selection decisions during two consecutive rechoke periods and measuring the difference between the two decisions, e.g., replacing an unchoked peer by a different peer counts as one change. Our measurements show that the $buddy-enhanced$ peer selection mechanism is more stable than the peer selection mechanism in the regular BitTorrent. Figure 4.7 indicates the average number of peer selection changes as a function of time (rechoke period units) for a peer with 5KB (one of the peers with improved download time in the experiment presented in Figure 4.2). It shows that the average number of peer selection changes is lower in the $buddy-enhanced$ network for the majority of the time, with an average of 1.35 changes in the regular BitTorrent network as compared to 0.54 average changes in the $buddy - enhanced$ network.

Notice that the optimistic unchoke mechanism contributes about 1 change every 3 rechoke periods, thus contributing an average change of about $\frac{1}{3}$ per time unit in the regular BitTorrent network. Therefore, the decrease in optimistic unchokes is not the main reason for the improvement in the stability of the peer selection mechanism. Instead,the main contributor to this stability is in However, replacing the tit-for-tat mechanisms, which relies on short-term history of associated peers with the buddy-selection mechanism, which relies on a long history and keeps matching peers unchoked as long as the peers behave as expected is the main contributor for this stability.

Quantifying System Performance

As a metric of system performance, we use the percentage of the available resources the system used throughout the downloading process. Obviously the optimal performance is achieved when the system used 100% of the available upload bandwidth resources, or otherwise the system wastes available resources instead of boosting peers that would like to download data through these upload bandwidth resources.

We calculate the percentage of resource usage in BitTorrent system using the next equation:

$$Usage = \frac{\sum_{p_i \in P} B_i}{\sum_{p_i \in P} T_i \times U_i}$$

Where N is the number of peers in the network, $P =< p_1, ..., p_i, ..., p_N >$ is the group of peers in the network, B_i is the amount of data peer p_i uploads to the system, T_i is the total time peer p_i was in the system, and U_i is the available upload rate of peer p_i. Thus, the numerator describes the amount of upload the system used in total from the N peers, and the denominator describes the total amount the system could use while the N peers were active in the downloading process.

In [LUM06] the authors studied the piece distribution mechanism and show that during most of the downloading process (except start-up and "end game" periods which last negligible time) peer has data that other peers in the network are interested in, with probability close to 1. Base on this we assume that peers can always saturate their upload rate. We also assume that a piece of data is downloaded only once by a peer as this is part of the design in the piece selection mechanism, and how it is implemented in practice for almost the entire process, i.e., peer requests a specific sub-piece from a single peer, peer requests only sub pieces that it does not have or has not started downloading from another peer. The only time the piece selection mechanism applies a different approach is in the "end game" which happens after the peer already requested all the pieces of the data content. During this short period, peer may request the last missing sub-pieces from all of its associated peers, however this part is negligible as peers may get at most 5 duplicate pieces due to the "end game" approach (1G file has 4096 pieces which are divided into 65536 sub-pieces).

The values in Table 4.1 show the $Usage$ of 7 different runs, for the regular BitTorrent system and the $buddy-enhanced$ system. Figure 4.8 shows the boxplots of these results. As we can see in the regular BitTorrent system only 64%-75% of the available resources are used, with median of 68%. Hence, the system could improve its performance by using about 47% more resources that were available during the downloading process. By replacing the regular BitTorrent peer selec-

BitTorrent	Buddy-enhanced
0.675052426	0.810991389
0.74731786	0.796512769
0.681063661	0.76888537
0.636869154	0.765860638
0.694803852	0.826679388
0.682270645	0.763435794
0.682497168	0.811222208

Table 4.1: percentage of available resources usage

tion mechanism with the $buddy - enhanced$ peer selection mechanism the system increases the usage of available resources. As we can see in Figure 4.8 and Table 4.1, the buddy-enhanced system uses between 76%-83% with median of 80%. Hence, the buddy-enhanced system increase by 18% the usage of available upload bandwidth resources during the downloading process, however this system is still far from reaching the optimal.

One reason for these results is that once peer starts to upload to another peer it takes time till the peer reaches its full capacity. In the BitTorrent protocol [Coh03] the author suggests to allow 30 seconds for a peer to reach its full capacity. Thus, system that has a high fluctuation in peer selection will have many occurrences with peers that do not reach their full capacity. In the $buddy - enhanced$ protocol we saw that this fluctuation reduced and this is one of the reasons for improving the resource usage in the $buddy - enhanced$ system.

The second reason is that both systems still have to deal with discovering the network as this information about upload rate of peers is not given in advance, but is learned during the downloading process. Moreover, in both systems the

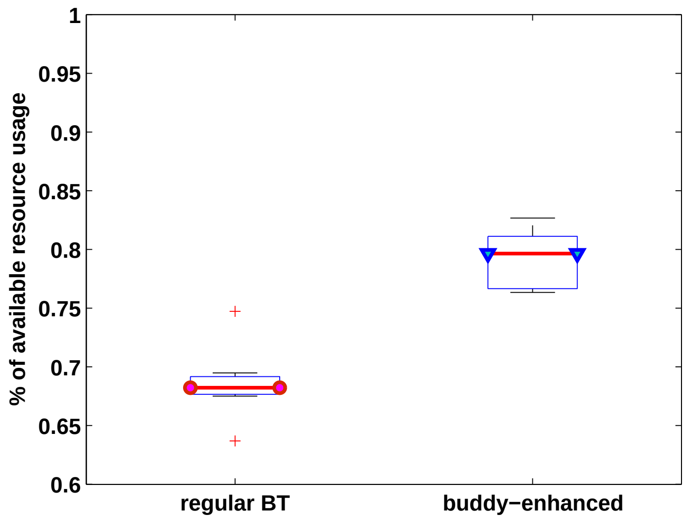


Figure 4.8: The Usage of available resources

peer selection decision is based on limited view of the network, and thus effect the ability to find peer that is interested in the available resources. In the team-enhanced protocol we further improve this by using a global view of the system.

4.5.2.2 Leecher Performance - System with Free-riders.

In this section we focus on the impact of free-riders, examining comparative results for contributing leechers' and free-riders' performance for both regular BitTorrent and $buddy - enhanced$ protocols in a network with free-riders. We added 8 free-riders, having a total of 112 peers in the experiment: 100 contributors and 4 seeds as described in previous experiments and additional 8 free-riders that download as much as possible but do not upload any data.

Figure 4.9 shows the download time for contributor leechers in the network. The Figure indicates that all the contributor leechers improve their download time performance in the $buddy - enhanced$ system in comparison to the regular

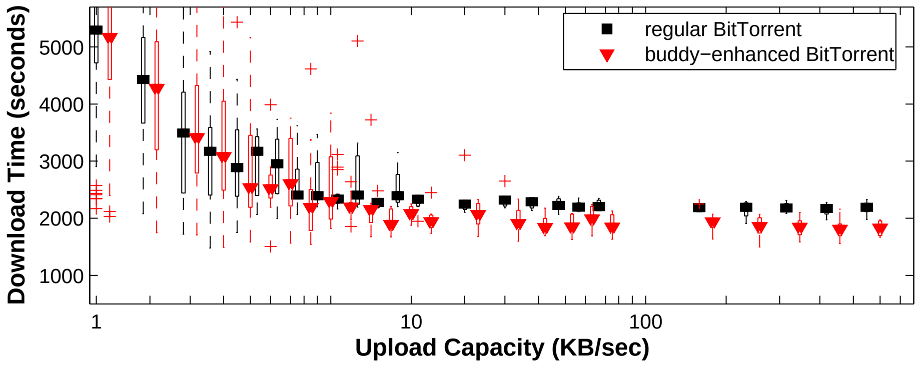


Figure 4.9: Leechers download time

BitTorrent system, an improvement of $2\% - 36\%$ as measured by the median. The entire group of contributor leechers improve their download time because this group of peers are more sensitive to additional free-riders in the regular BitTorrent network, where uploading to free-riders occurs more often due to the higher probability of performing optimistic unchokes.

Figure 4.10 validates this hypothesis. The Figure shows the percentage of the content that was downloaded from leechers (as opposed to downloaded from seeds) for each of the free-rider s in the system, which indicates a decrease of $15\% - 24\%$ in the $buddy - enhanced$ system as measured by the median.

Figure 4.11 shows the download completion times of free-riders, demonstrating that *buddies slowed down free-riders by 21-48%.* As shown in Figure 4.10, this is because free-riders cannot take advantage of optimistic unchokes as in regular BitTorrent and thus have to depend more on the seeds for downloading data. These results also attest to the increased robustness of the system to free-riding.

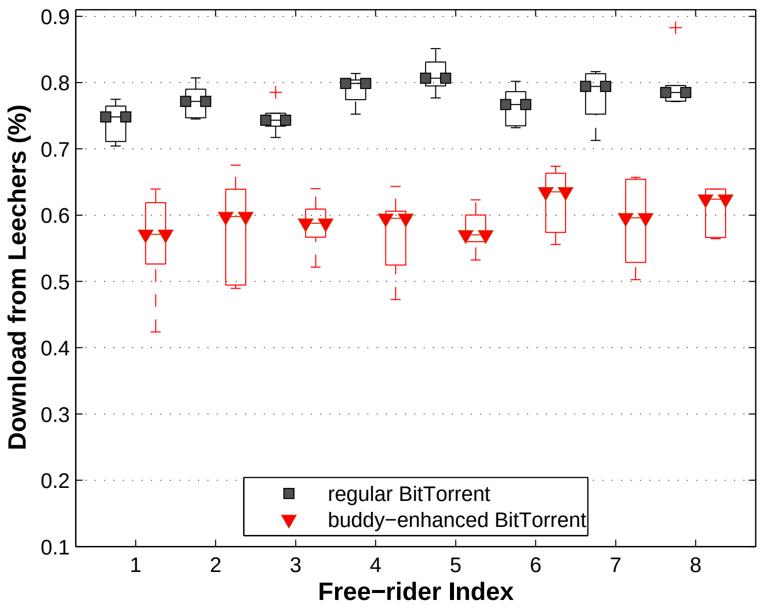


Figure 4.10: free-riders' download from leechers

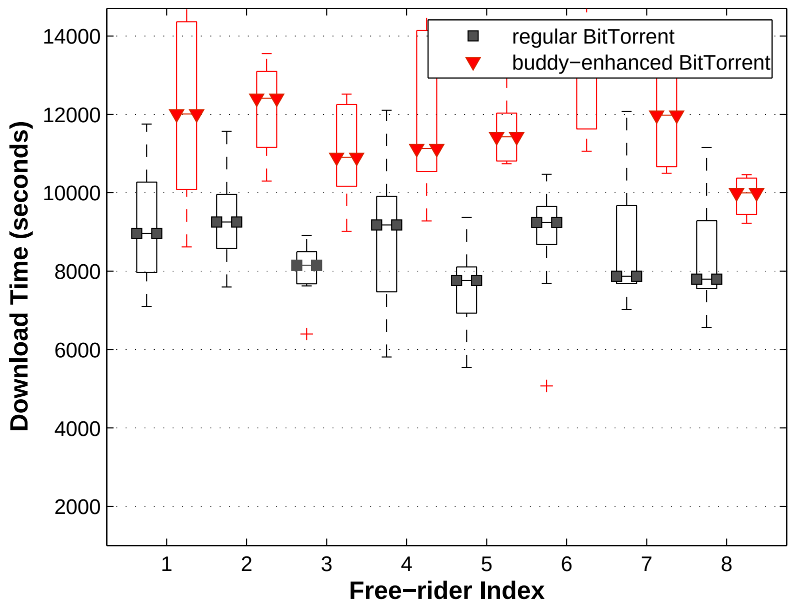


Figure 4.11: free-riders download time

CHAPTER 5

Foresighted Strategy in BitTorrent Systems

5.1 Overview

In P2P content distribution systems, fairness among peers participating in content distribution is an important factor, as it encourages peers to actively collaborate in disseminating content, which can lead to improved system performance. However, as discussed in the previous chapters, BitTorrent does not provide fair resource reciprocation, especially in node populations with heterogeneous upload bandwidths [PIA07, BHP06, GCX05, LLK07]. One of the reasons for that is the tit-for-tat strategy, that is implemented in BitTorrent, which is based on short-term history, i.e., upload decisions are made based on most recent resource reciprocation observation. Moreover the decision is based on backward looking and not forward looking. Thus, a peer can follow the tit-for-tat policy only if it continuously uploads pieces of a particular file and as long as it receives pieces of interest in return. However, this is not always possible, as peers may not have pieces that are of interest to others, regardless of their willingness to cooperate [PIK08]; yet, this behavior is still perceived as a lack of cooperation.

Additionally, it has been shown that BitTorrent systems do not effectively cope with selfish peer behaviors, such as free-riding [LNK06, LMS06, SPC07], due to the optimistic unchoke mechanism currently used in BitTorrent. This enables peers to continuously discover better leechers to reciprocate resources with. However, it creates a major opportunity for peers to obtain data, without uploading in return. Moreover, it may induce unfairness in the system, as it forces high-capacity peers to interact with low-capacity ones.

Reputation-based schemes that are based on the propagation of global history (e.g., [BB04, XL04, YZL05]) have been proposed to overcome the limitations of pure tit-for-tat and optimistic unchoke mechanisms. However, these approaches require significant communication overhead to maintain the global history across peers. Moreover, the reliability of global history is unclear as peers may exhibit different reciprocation behaviors with different peers. Alternatively, in other reputation-based approaches such as [eDo, PIK08, Izh09, ILM09], peers make peer selection decisions based on long-term local (or private) history of associate peers' upload behaviors. However, the focus of these systems is on maximizing *immediate* utility, which may be less desirable than maximizing *long-term* utility, as peers can repeatedly interact with each other in a long period of time.

In this chapter, we model the peer interactions in the BitTorrent-like network as a $stochastic - game$[FL99] – A repeated interaction (i.e., reciprocating resources) between several participants (i.e., peers) in which the underlying state of the environment changes stochastically, and is dependent on the decisions of the participants. Stochastic games extend the single participant Markov decision process (MDP) to include multiple participants whose actions all impact the resulting utility and next state. In our model, we explicitly consider the strategic behavior peers, which can observe partial historic information of associated peers statistical reciprocation behaviors, through which the peers can estimate the impact on their future rewards and then adopt their best response. The estimation of the impact on the expected future reward can be performed using different types of interactive learning [BV01]. We use reinforcement learning method [HW], as it allows the peers to improve their peer selection strategy using only knowledge of their own past received payoffs, without knowing the complete reciprocation behaviors of the peers in the network. Thus, we propose to replace the peer selection policy with a reinforcement learning foresighted resource reciprocation policy. The resource reciprocation policy is calculated by forecasting the impact of the current peer selection actions on the expected utility (i.e., future rewards) and maximizing it.

Thus, each peer can maximize its long-term utility based on the foresighted resource reciprocation policy. This can also provide an improved fairness, discourages free-riding, and enhances the system robustness. The foresighted resource reciprocation policy can replace both the tit-for-tat and the optimistic unchoke mechanisms in the regular BitTorrent protocol.

In this chapter, we propose a BitTorrent-like protocol that applies the reinforcement learning foresighted strategy. Specifically, the protocol consists of three main processes:

- A learning process, which provides an updated information about statistical behaviors of the associated peers' resource reciprocation.
- A policy finding process, which computes the foresighted policy using the reinforcement-learning algorithm.
- A decision process, which determines the associated peers that will be unchoked and choked in every rechoke period based on the foresighted policy.

We implemented our proposed protocol on top of a BitTorrent client, and performed extensive experiments on a controlled PalnetLab testbed. We evaluate and quantify the performance of the proposed protocol, and compare its performance with the regular BitTorrent protocol. Based on the experimental results, the proposed protocol provides the following advantages against the regular BitTorrent protocol:

1. It improves the fairness. The peers that contribute more resources (i.e., higher upload capacities) can achieve higher download rates. However, the peers that contribute fewer resources may achieve limited download rates.

2. It promotes cooperation among high-capacity peers.

3. It discourages free-riding by limiting the upload to non-cooperative peers.

4. It improves the system robustness by minimizing the impact of free-riding on contributing peers' performance.

The rest of this chapter is organized as follows. In Section 5.2, we briefly define stochastic games and describe the reinforcement learning foresighted resource reciprocation strategy. Section 5.3 presents the design of our foresighted resources reciprocation protocol. Details of our protocol implementation are discussed in Section 5.4. Finally, the experiment results are presented in Section 5.5.

5.2 The Stochastic Game Model

Peers in BitTorrent-like systems have to make a repeated peer selection decision given their dynamically changing environment which they experience. The evolution of the peers' interactions across the various rechoke periods can be modeled as repeated stochastic interaction. Whereas the Markov Decision Process (MDP) is a decision problem for one participant (i.e., peer) in an (unknown) environment [Gal96], when multiple participants interact with each other in such an environment, this becomes a stochastic game [FL99, Sha53] problem (i.e., n-participant MDP). The time is discrete in the stochastic game, and at each time

I	A set of peers participanting in the download process
M	Number of peers who participant in the download process
$\mathbf{S}$	A set of state profiles of all peers
$\mathbf{A}$	The joint action space
C_j	The set of peers associated with peer j
N	The number of peers associated with peer j
P_{A_j}	A state transition probability
$R_j(S_j)$	The reward of a peer j in state S_j
π_j	A policy profile for peer j

Table 5.1: Foresighted Resource Reciprocation Model Notation

slot (i.e., rechoke period), every participant has its own state and its own action space for that state. Every time slot the participants choose their own actions independently and simultaneously. After that, the participants are rewarded and transit to the next states. The reward (received by each of the participants), and the state transition also is contingent upon other participants' states and actions. Specifically to our model, during the repeated multi-peer interaction, the peers can observe partial historic information of associated peers reciprocation behaviors, through which the peers can estimate the impact on their future rewards and then adopt their best response. The estimation of the impact on the expected future reward can be performed using different types of interactive learning [BV01]. Here, we use reinforcement learning method [HW], as it allows the peers to improve their peer selection strategy using only knowledge of their own past received payoffs, without knowing the complete reciprocation behavior of the peers in the network. In this learning framework of stochastic-game, the learning peers attempt to maximize their expected rewards.

Formally, a stochastic game is a tuple, $\langle I, \mathbf{S}, \mathbf{A}, P, R \rangle$, where I is a set of participants (peers), i.e., $I = \langle 1, ..., M \rangle$, $\mathbf{S}$ is the set of state profiles of all peers, i.e., $\mathbf{S} = S_1 \times ... \times S_M$ with $\mathbf{S}_j$ being the state space of peer j, and $\mathbf{A}$ is the joint action space $\mathbf{A} = A_1 \times ... \times A_M$, with $\mathbf{A}_j$ being the action (peer selection) space for peer j. $P : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \rightarrow [0, 1]$ is a state transition probability function that maps from state profile $S(t) \in \mathbf{S}$ at time t into the next state profile $S(t+1) \in \mathbf{S}$ at time $t + 1$ given corresponding joint actions $A(t) \in \mathbf{A}$. Note that t here is discrete and measured in time slots. Finally, $R : \mathbf{S} \times \mathbf{A} \rightarrow \mathbb{R}_+$ is a reward vector function defined as a mapping from the state profile $S(t) \in \mathbf{S}$ at time t, and corresponding joint actions $A(t) \in \mathbf{A}$ to an $M-$dimensional real vector with each element being the reward to a particular participant.

5.2.1 The State Space of a Peer - $\mathbf{S}_j$

A state of peer j represents a set of resources received from the peers in $C_j \in I$, where C_j denotes the set of peers associating with peer j. Thus, it may represent the uploading behavior of its associated peers, or equivalently, it can capture peer j's download rates from its associated peers. The upload rates from peer $i \in C_j$ to peer j at time t are denoted by $UL_{i,j}(t)$. In our proposed protocol, an uploading behavior of peer i observed by peer j is denoted by s_{ij}, and defined as

$$s_{ij} = \begin{cases} 1, & \text{if } UL_{i,j} > \theta_j \ , \\ 0, & \text{otherwise,} \end{cases} \tag{5.1}$$

where θ_j is a pre-determined threshold of peer j. [1] Thus, s_{ij} can be expressed with one bit and the state space of peer j can be expressed as

$$\mathbf{S}_j = \{ (s_{1j}, \ldots, s_{Nj}) | \, s_{kj} \in \{0, 1\} \text{ for all } k \in C_j \} \,, \tag{5.2}$$

[1]In order to minimize the computational complexity, we consider $s_{ij} \in \{0, 1\}$ in this chapter. However, the granularity of state can be easily extended.

where N denotes the number of peer j's associated peers, i.e., $|C_j| = N$. Therefore, a state $S_j(t) \in \mathbf{S}_j$ can capture the uploading behavior of the associated peers at time t. A state can be described using N bits, and thus, the cardinality of the state space is 2^N.

5.2.2 The Action Space of a Peer - $\mathbf{A}_j$

An action of peer j represents a set of its peer selection decisions. The peer selection decision of peer j to peer i at time t is denoted by a_{ji}, and is defined as

$$a_{ji}(t) = \begin{cases} 0, & \text{if peer } j \text{ chokes peer } i \text{ ,} \\ 1, & \text{otherwise,} \end{cases} \tag{5.3}$$

Thus, a_{ji} can be expressed with one bit. The action space of peer j can be expressed as

$$\mathbf{A}_j = \{\, (a_{j1}, \ldots, a_{jN}) | \, a_{jk} \in \{0, 1\} \text{ for all } k \in C_j \} \,, \tag{5.4}$$

Hence, an action $A_j(t) \in \mathbf{A}_j$ is a vector that consists of peer j's peer selection decisions to its associated peers at time t. Thus, an action of peer j for its N associated peers can be described with N bits. In the proposed protocol, we assume that peer j is able to unchoke a limited number of peers, denoted by $N_u (\leq N)$, implying that the cardinality of the action space is $\binom{N}{N_u}$. Note that in order to reduce the complexity, peer j allocates the same amount of upload bandwidths to all unchoked peers. Thus, the bandwidth allocated to an unchoked peer i by peer j at time t is determined by $UL_{j,i}(t) = B_j / N_u$, where B_j is the maximum upload bandwidth available to peer j.

5.2.3 State Transition Probability in a Peer

A state transition probability represents the probability that an action $A_j(t) \in \mathbf{A}_j$ of peer j in state $S_j(t) \in \mathbf{S}_j$ at time t will lead to another state $S_j(t+1) \in \mathbf{S}_j$ at $t+1$. Thus,

$$P_{A_j(t)}(S_j(t), S_j(t+1)) = \Pr(S_j(t+1)|S_j(t), A_j(t)). \tag{5.5}$$

The state transition probability functions can be estimated at a particular peer j based on the history of $S_j(t)$, $A_j(t)$ and $S_j(t+1)$, which may be stored in a transition table. Thus, the transition table size is in order of $O(\binom{N}{N_u} \cdot 2^N \cdot 2^N)$. While we deploy an empirical frequency based algorithm to estimate the state transition probability function, which is presented in Section 5.3.1, other algorithms (e.g., [PS09]) can also be used.

5.2.4 The Reward of a Peer - R_j

The reward of a peer in a state is its total estimated download rate in the state. Thus, a reward of a peer in a state is the sum of the estimated download rates from all its associated peers. More specifically, a reward of peer j from state $S_j \in \mathbf{S}_j$ can be expressed as

$$R_j(S_j) = \left\langle S_j, [UL_{i,j}]_{i \in C_j} \right\rangle \tag{5.6}$$

where $\langle\cdot\rangle$ denotes the inner-product operation.

5.2.5 Resource Reciprocation Policy π_j

We define a history of the stochastic game as $h^t = \{S^0, A^0, R^0, ..., S^{t-1}, A^{t-1}, R^{t-1}\} \in \mathcal{H}^t$, which summarizes all states, actions and rewards of the peers in the network up to time $t-1$, where $\mathcal{H}^t$ is the set of all possible histories up to time t.

Nevertheless, during the stochastic game, each peer j cannot observe the entire history, but rather a portion of the history. The observation of peer j is denoted as $o_j^t \in \mathcal{O}_j^t$ and $o_j^t \subset h^t$. Note that the current state s_j^t can be always observed, i.e. $s_j^t \in o_j^t$. Thus a peer selection policy $\pi_j^t : \mathcal{O}_j^t \to \mathbf{A}_j$ for peer j at the time t is defined as a mapping from the observations up to the time t into the specific action, i.e. $a_j^t = \pi_j^t(o_j^t)$. Furthermore, a policy profile π_j for peer j aggregates the peer selection policies over the entire course of the stochastic game, i.e. $\pi_j = (\pi_j^0, ..., \pi_j^t, ...)$. The policy profile for all the peers at time slot t is denoted by $\pi^t = (\pi_1^t, ..., \pi_M^t) = (\pi_j^t, \pi_{-j}^t)$.

The policy of peer j is calculated using reinforcement-learning algorithm that maximizes the cumulative discounted expected reward. The expected reward is defined for a peer j in state $S_j(t)$ at time $t = t_c$ given a discount factor γ_j as

$$R_j^f(S_j(t_c)) \triangleq \sum\nolimits_{t=t_c+1}^{\infty} \gamma_j^{(t-(t_c+1))} \cdot R_j(S_j(t)). \tag{5.7}$$

The policy π_j maps each state $S_j(t) \in \mathbf{S}_j$ into an action, i.e., $\pi_j(S_j) = A_j(t)$ such that each action maximizes $R_j^f(S_j(t_c))$. The policy can be deployed as a peer selection algorithm, such that each peer can maximize its long-term utility. While the policy π_j can be obtained using well-known methods such as value iteration and policy iteration [Ber76], these algorithms may require very high computational complexity if the number of associated peers is significantly large. Hence, it is important to reduce the computational complexity of the policy calculation, such that the reinforcement learning foresighted strategy is deployed in practice.

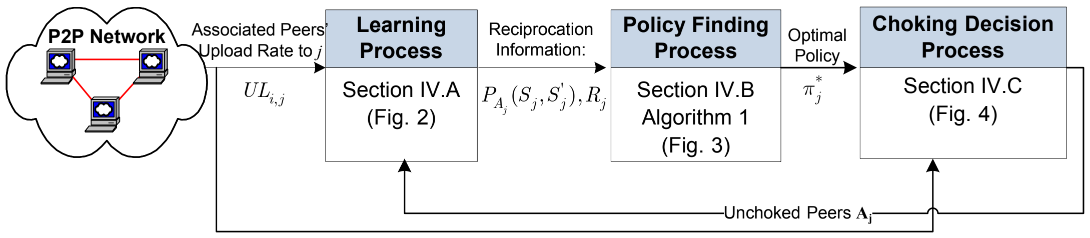

P2P Network
Associated Peers' Upload Rate to j
$UL_{i,j}$
Learning Process
Section IV.A (Fig. 2)
Reciprocation Information:
$P_{A_j}(S_j, S'_j), R_j$
Policy Finding Process
Section IV.B Algorithm 1 (Fig. 3)
Optimal Policy
π^*_j
Choking Decision Process
Section IV.C (Fig. 4)
Unchoked Peers $\mathbf{A}_j$

Figure 5.1: The main processes in the proposed protocol design

5.3 The Protocol Design

In this section, we describe the proposed protocol design that replaces the peer selection tit-for-tat and optimistic unchoke mechanisms deployed in regular BitTorrent systems with a foresighted resource reciprocation mechanism.

The protocol design is summarized in Fig. 5.1. The protocol consists of three main processes running in parallel: (1) *the learning process*, which provides an updated information about statistical behaviors of the associated peers' resource reciprocation; (2) *the policy finding process*, which computes the foresighted resource reciprocation policy; (3) and *the decision process*, which determines the peer selection decisions during the course of each rechoke period. More details about these processes are discussed next.

5.3.1 The Learning Process

As discussed in Section 5.2, in order to find the foresighted resource reciprocation policy, each peer needs to know other peers' states, their rewards, and their state transition probabilities in order to derive its own optimal policy. However, a peer cannot exactly know the other peers information, due to information that is kept private, network scalability constraints, the time-varying network dynamic, and more. Thus, to improve the peer selection policy, a peer can only predict

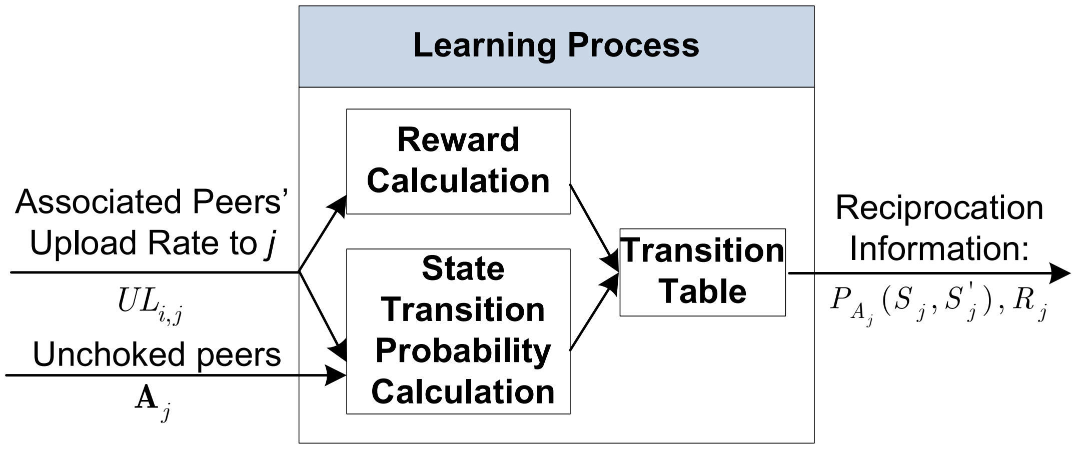

Learning Process
Reward Calculation
Associated Peers' Upload Rate to j
$UL_{i,j}$
State Transition Probability Calculation
Unchoked peers
$\mathbf{A}_j$
Transition Table
Reciprocation Information:
$P_{A_j}(S_j, S'_j), R_j$

Figure 5.2: The Learning Process

the impacts of dynamics(uncertainties) caused by the competing peers based on its observations from the past. Thus each peer needs to update the above information regularly through the learning process, while downloading content from its associated peers.

The learning process consists of two main methods (see Fig. 5.2) that compute the estimated reward and state transition probability.

5.3.1.1 Reward Calculation

The reward of peer j represents its download rates from its associated peers (or equivalently, the total upload rates of its associated peers) estimated by peer j. In the rewards calculation method, the associated peers are classified into two types based on the available information about their resource reciprocation history.

For associated peers whom have reciprocated their resources with peer j, referred to as *peers with reciprocation history*, peer j estimates their upload rates based on weighted average of the past upload rate samples. This can reduce the fluctuation induced by the protocol and network dynamics in the sampled upload rates of the associated peers. Specifically, peer j estimates the upload rates $\hat{r}_{i,j}$ of peer $i \in C_j$ based on recently observed resource reciprocation $UL_{i,j}$ as

$$\hat{r}_{i,j} \leftarrow \alpha_j \cdot UL_{i,j} + (1 - \alpha_j)\hat{r}_{i,j} \tag{5.8}$$

where α_j denotes the weight for most recent resource reciprocation.

For associated peers whom have *not* reciprocated their resources with peer j, referred to as *peers without resource reciprocation history*, peer j assumes their upload rates. Peer j optimistically initiates the information about such peers by assuming that they reciprocate their resources with high probability and high upload rate. This enables peer j to efficiently discover additional peers, and bootstrap newly joining peers, which is important for the efficiency of the system. Whenever peer j uploads to a peer without resource reciprocation history and the peer does not upload to j in return, peer j reduces the peer's presumed upload rate, as this provides j with more confidence that the particular peer may not actively reciprocate its data. This also prevents the associated peers from taking advantage through optimistic initialization and possible free-riding.

5.3.1.2 State Transition Probability Calculation

The state transition probabilities are updated every rechoke period, and thus, each peer can capture the time-varying resource reciprocation behaviors of its associated peers. Every rechoke period at $t + 1$, peer j stores a 3-bit triplet $(S_j(t), A_j(t), S_j(t + 1))$. Peer j stores the triplets to the associated peers that are in a particular peer set referred to as reduce peer set, which will be discussed later in this section or peers that uploaded to peer j at time t or $t+1$. In this chapter, we assume that the state transition probability functions are computed based on the empirical frequency, and the state transition of each peer is independent.

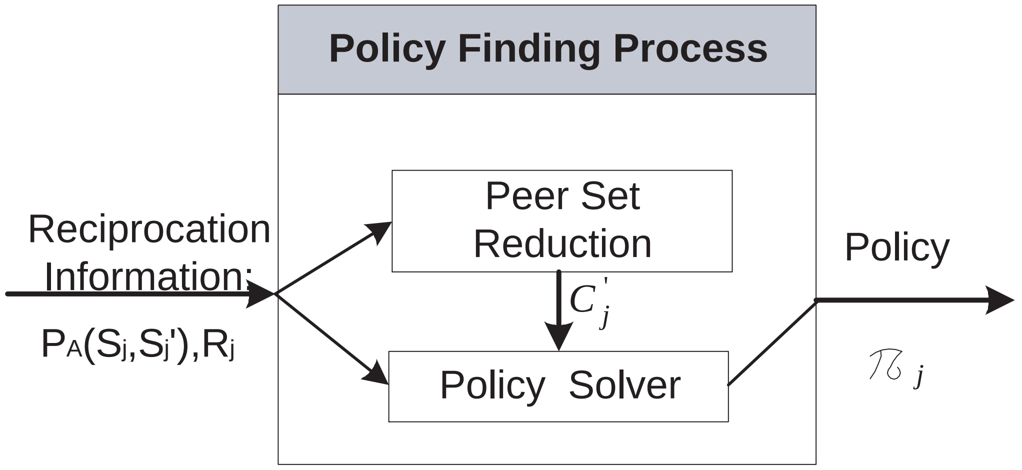


Figure 5.3: The Policy Finding Process

Thus, the probability, $\Pr(s_{ij}, a_{ji}, s'_{ij})$, where $s_{ij}, s'_{ij} \in \mathbf{S}_j$, can be expressed as

$$P_{A_j(t)}(S_j(t), S_j(t+1)) = \prod_{i=1}^{N} \Pr(s_{ij}(t+1)|s_{ij}(t), a_{ji}(t))$$

5.3.2 The Policy Finding Process

The policy finding process runs in parallel with the learning process, while computing the foresighted resource reciprocation policies based on the information obtained from the learning process. This process is depicted in Fig. 5.3.

Finding the reinforcement learning foresighted policy may require significantly high computational complexity if the number of the associated peers becomes large. Hence, for practical implementation of the foresighted resource reciprocation mechanism, it is critical to reduce the number of peers that needs to be considered (see Section 5.2). Thus, this process begins with reducing the set of associated peers, and then, finds the foresighted resource reciprocation policy π_j by maximizing the cumulative discounted expected reward (Equation 5.7) in the reduced peer set.

5.3.2.1 Reducing Associated Peer Set

As discussed, it is important for peer j to reduce the set of associated peers to find π_j, while keeping the peers that reciprocate their resources with higher probability and with higher upload rate in the reduced peer set. Specifically, peer j computes the expected rewards $K_{i,j}$ from each peer $i \in C_j$, defined as

$$K_{i,j} = R_{i,j} \times Pr(i,j), \tag{5.9}$$

where $R_{i,j}$ and $Pr(i,j)$ denote the estimated reward from peer i and the probability of resource reciprocation with peer i, respectively. Based on computed $K_{i,j}$, peer j reduces its associated peer set by iteratively consider eliminating the peers with the smallest $K_{i,j}$ in its associated peer set. The algorithm for peer set reduction is presented in Algorithm 1. The algorithm computes $K_{i,j}$ in (5.9) for $i \in C_j$ (lines 3,4). Then, the associated peers are ordered based on computed $K_{i,j}$ (line 5). The peer set reduction is performed in while loop (lines 7- 18) that reduces the peer set by c_2 peers in every iteration. In the loop, the algorithm selects c_1 peers with the smallest $K_{i,j}$ values denoted by G (line 8), from the reduced group of peers C'_j. It then obtains policy $\pi_{j,G}$ for the peers in G. (line 9). Based on $\pi_{j,G}$, it calculates the probabilities for the peers to be unchoked (lines 10-14). Given the calculated probability, it removes the c_2 peers with the lowest probability to be unchoked (18). The algorithm runs until $|C'_j| = T$ (line 7).

5.3.2.2 Scaling

Scaling of the rewards is considered in cases, when the number of reciprocation samples is small in comparison to the difference between the highest and the

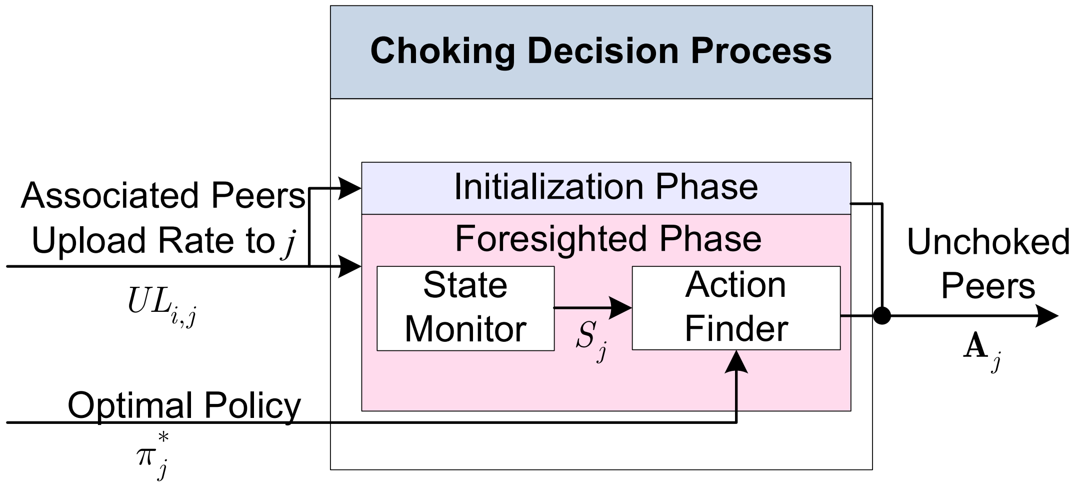


Figure 5.4: The Decision Process

lowest upload rates that are expressed in the P2P network.

5.3.3 The Decision Process

The decision process includes two phases: the initialization phase and the foresighted phase (see Fig. 5.4).

5.3.3.1 Initialization Phase

Since no information about associated peers is available for a newly joined peer j, peer j begins with adopting the regular BitTorrent mechanisms (i.e., the tit-for-tat mechanism and the optimistic unchoke mechanism) in the initialization phase. This enables the peer to collect information such as the rewards and state transition probabilities with respect to its associated peers. During this phase, j discovers new peers, i.e., downloads from peers for the first time. Once j's peer discovery is slowed down (see Section 3.4 for more details), it replaces the regular BitTorrent mechanisms with the foresighted resource reciprocation mechanism, and operates in the foresighted phase.

5.3.3.2 Foresighted Phase

Once the foresighted resource reciprocation policy is available, peer j determines the peer selection decisions based on the foresighted policy obtained from the policy finding process in every rechoke period. Peer j first determines its current state S_j and then finds an optimal action A_j mapped by the policy π_j, i.e., $A_j = \pi_j(S_j)$. A_j is a set of peers that peer j unchokes.

5.4 Implementation

In this section, we discuss the implementation of the foresighted resource reciprocation protocol prototype and study how to determine several design parameters.

Our P2P client is implemented based on Enhanced CTorrent client, version 3.2 [Enh]. We enhances the original client such that our client can operate in *foresighted mode*, where it reciprocates its resources based on the proposed foresighted resource reciprocation mechanism, or in *regular mode*, where it reciprocates its resources based on the regular BitTorrent peer selection mechanisms. We add functionality for the foresighted mode to maintain the new protocol requests. More specifically, we implemented the three different processes that are discussed in Section 3.2.

5.4.1 The Learning Process

The learning process consists of two methods, the reward calculation method and the state transition probability calculation method. We will now discuss the reward calculation method.

5.4.1.1 Reward Calculation Method

The reward calculation method can be applied differently depending on the associated peer types: peers with or without reciprocation history.

While calculating the reward of a peer with resource reciprocation history, clearly the samples of $UL_{i,j}$ will fluctuate over rechoke period time due to P2P network dynamics. Because of this fluctuation, $UL_{i,j}$ samples may be atypical. Thus, a typical upload rate of a peer with reciprocation history can be estimated based on weighted average of the samples as in (5.8). This is the estimated reward of peer j obtained from peer i. As recent resource reciprocation is considered more important than the previous reciprocations, $\alpha_j > 0.5$. Based on several trials for α such that $0.5+\epsilon \leq \alpha \leq 1-\epsilon$ for small $\epsilon > 0$, on various sets of our experiments (see more details in Section 3.5), we can verify that the smallest α achieves less fluctuation of the reward. Thus, we set α to $0.5 + \epsilon$ where $\epsilon = \frac{1}{16}$.

Fig. 5.5 shows the sampled upload rates $U_{i,j}$ of a peer i having 9KB/sec upload bandwidth (that simultaneously uploads to 4 peers) and the correspondingly estimated rewards $\hat{r}_{i,j}$ as measured by another peer in the network. Clearly, we can observe less variations of the $UL_{i,j}$ in the computation of the $\hat{r}_{i,j}$.

For a peer i without reciprocation history, a leecher j optimistically initializes the information about the rewards and the reciprocation probabilities of its associated peers. Specifically, the initial estimated upload rate is set to be the highest upload rate $R_{i,j}^{max}$ that is pre-determined in the P2P network, i.e., $R_{i,j} \leftarrow R_{i,j}^{max}$, and the probability of reciprocation with j is initiated to 1, i.e., $Pr(i,j) \leftarrow 1$. This optimistic initialization enables newly joined leechers to download almost immediately. Peer j needs to keep updating the initially assumed reward in every non-reciprocated event (i.e., peer j uploads resources to peer i while peer i does not upload resources to peer j). When peer j estimates the reward for peer i,

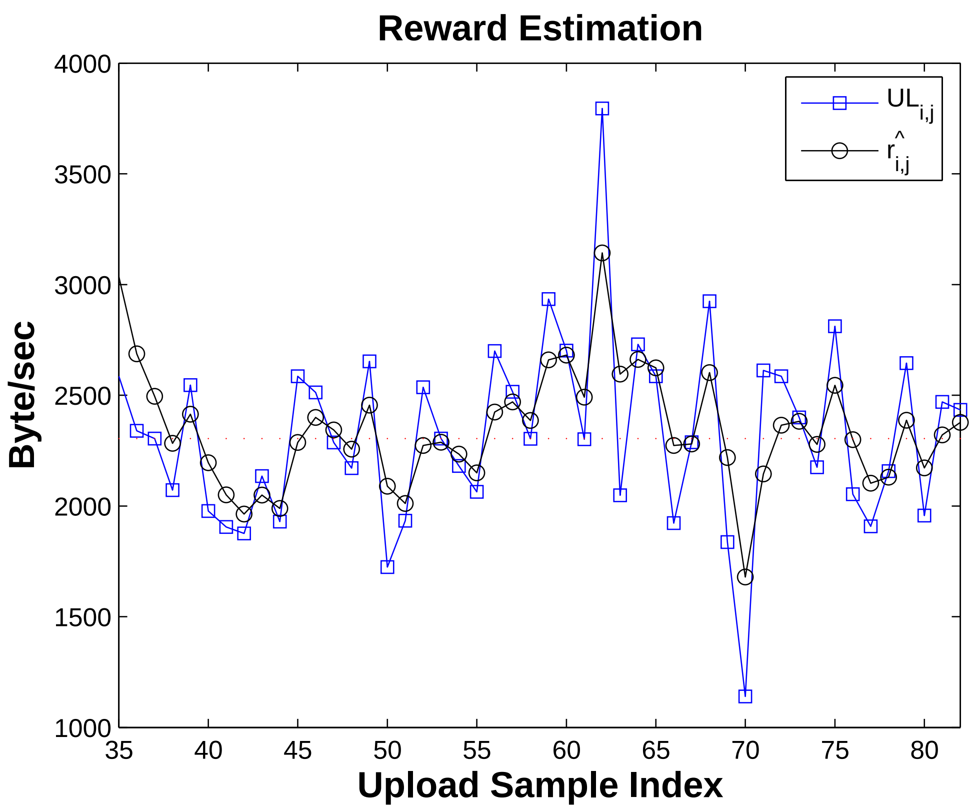


Figure 5.5: Upload samples and reward estimations

peer j can assume that (i) $\hat{r}_{i,j}$ satisfies

$$\frac{\hat{r}_{i,j}(n-1)}{\hat{r}_{i,j}(n)} < \frac{\hat{r}_{i,j}(n)}{\hat{r}_{i,j}(n+1)}, \tag{5.10}$$

where n denotes the number of non-reciprocated events. This means that the ratio of the estimated rate of two consecutive events is an increasing function of n. This implies the increasing uncertainty about peer i's reciprocation behavior. Moreover, (ii) $\hat{r}_{i,j}(n)$ decreases exponentially such that it approaches 0 after several attempts, in order to prevent the non-reciprocated behavior including free-riding. Thus, a function satisfying (i) and (ii) can have a form, such as

$$r(n) = \beta^{g(n)} \times R_{i,j}^{max}, \tag{5.11}$$

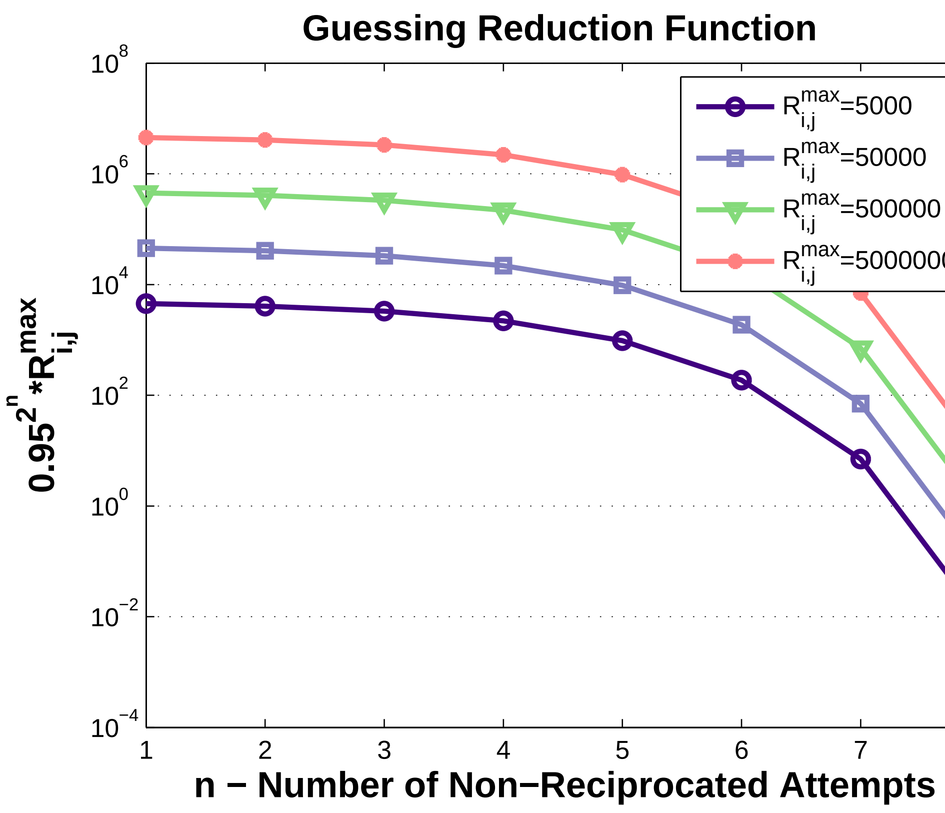


Figure 5.6: Guessing Reduction Function

where $\beta(< 1)$ is a constant and $g(n) > 1, \forall n \geq 1$ is a function that grows faster than a linear function. In our implementation, we use function $r(n) = 0.95^{2^n} \times R_{i,j}^{max}$, as the function satisfies properties (i) and (ii) as shown in Fig. 5.6.

5.4.2 The Policy Finding Process

As shown in Section 3.2, in every iteration of the policy finding process, the associated peer set is first reduced. Based on our experiments, we observe that when the reduced size of peer set is more than 7 peers, finding the foresighted strategy requires significant computational complexity. Thus, in our implementation, we set the size of the reduced peer set to 7, i.e., $T = 7$ in Algorithm 1.

The policy is calculated and holds for up to additional three rechoke periods, which is determined based on the tradeoff between the time for enough reciprocation and the time for capturing the network dynamics.

5.4.3 The Decision Process

The initialization phase and the foresighted phase in the decision process are implemented as follows.

- *Initialization Phase:* In the initialization phase, peer j makes peer selection decisions based on the regular BitTorrent mechanisms, as it does not have enough information to calculate the foresighted policy. The leechers determine the duration of the initialization phase individually. We study extensive experiment results, which include both flash crowd scenarios as well as steady state scenarios. In these experiments, the number of peers that have not uploaded to peer j from the beginning of the downloading process is counted every rechoke period. Fig. 5.7 shows the median of the counted numbers of peers collected from all the leechers in the network over time (rechoke periods) for several experiments of flash-crowd scenarios. The Figure shows that the peer counted value is exponentially decreasing and stabilized fast. Then, peer j can switch from the initialization phase to the foresighted phase. In our implementation, a peer j counts the number of peers without reciprocation history within every rechoke period. Once the count reduces by one in duration of three rechoke periods and for two consecutive durations (i.e., six rechoke periods), peer j switches to the continuous phase and starts to adopt the foresighted strategy. Based on our experiments, peers switch from initialization phase to continuous phase approximately 60 rechoke periods later in the flash-crowd scenarios and approximately 36 rechoke periods later in the steady-state scenarios. However, different network

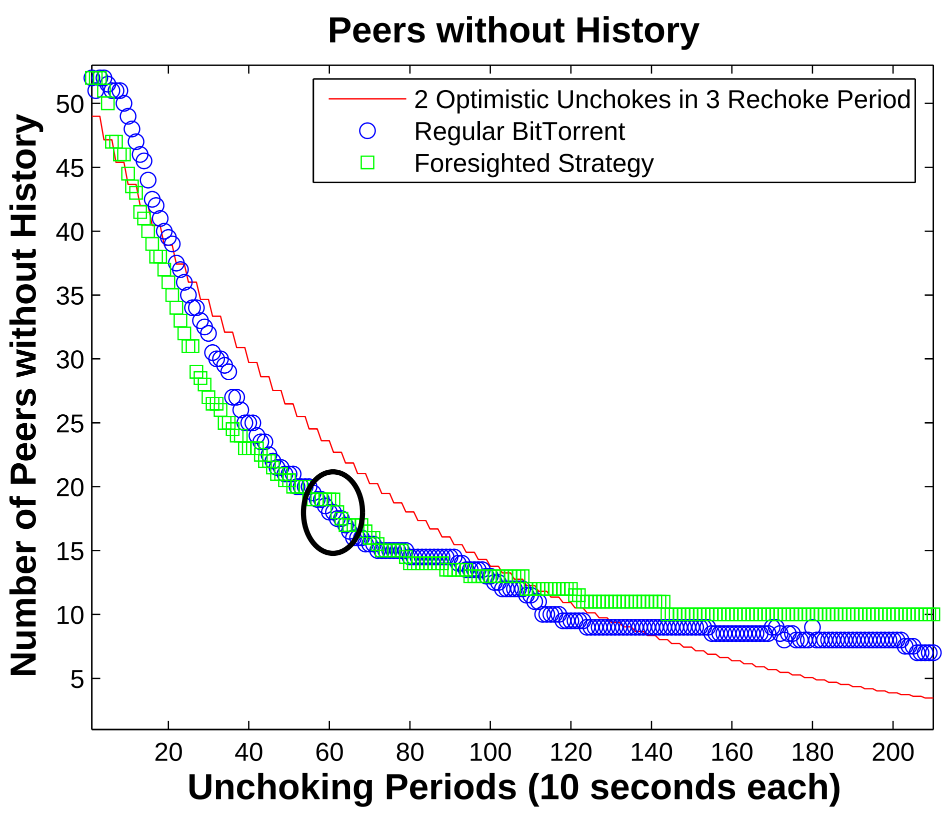


Figure 5.7: Discovering rate of peers without history

settings might lead to different durations of the initialization phase.

- *Foresighted Phase:*

In the foresighted phase, the peer selection decisions are made based on the foresighted policy, in ten second intervals (as in regular BitTorrent). The selected peers will be unchoked during the entire ten second rechoke period. The minimum number of unchoked peers is 4. The number of unchoked peers can increase if (1) the peer that makes the peer selection decision does not saturate its upload capacity, or, (2) the upload bandwidth of the peer that makes the peer selection decision is higher in comparison to most of the peers it interacts with.

We compare the performance of our protocol to the performance of the regular BitTorrent implemented with the Enhanced CTorrent code. The minimum number of unchocke slots in the regular BitTorrent implementation is 4. The number of slots can increase if a peer's upload capacity is not saturated. In this implementation, one unchoke slot is always reserved for optimistic unchokes that are rotated every three rechoke periods.

5.5 Experimental Evaluation

We perform extensive experiments on a controlled testbed, in order to evaluate the properties of the proposed protocol.

5.5.1 Methodology

All of our experiments are performed on the PlanetLab experimental platform [BBC], which utilizes the nodes (machines) located across the globe. We execute all the experiments consecutively in time on the same set of nodes. Unless otherwise specified, the default implementations of leecher and seed in regular BitTorrent systems are deployed.

The upload capacities of the nodes are artificially set according to the bandwidth distribution of typical BitTorrent leechers [PIA07]. The distribution was estimated based on empirical measurements of BitTorrent swarms including more than 300,000 unique BitTorrent IPs. Since several nodes are not capable to match the target upload capacities determined by the bandwidth distribution, we scale the upload capacity and other relevant experimental parameters such as file size by 1/20th. However, we have not set limitation on download bandwidth.

All peers start the download process simultaneously, which emulates a flash crowd scenario. The initial seeds are stayed connected through the whole exper-

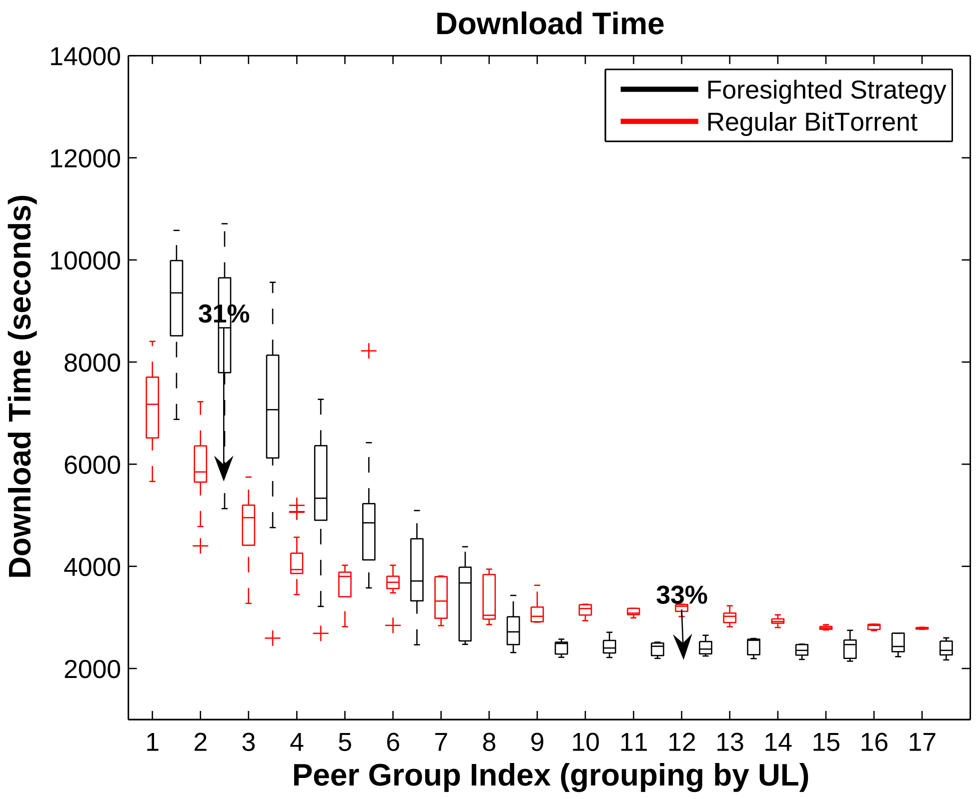


Figure 5.8: Download completion time for leechers

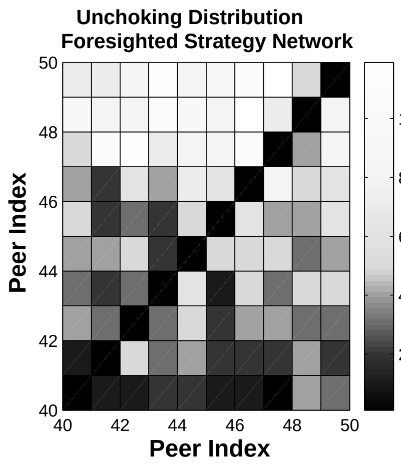
iments. To provide synthetic churn with constant capacity, leechers disconnect immediately after completion of downloading the entire file, and reconnect immediately while requesting the entire file again. This enables our experiments to have the same upload bandwidth distribution during the entire experiment time.

Unless otherwise specified, our experiments host 54 PlanetLab nodes, 50 leechers and 4 seeds with combined capacity of 128 KB/s serving a 100 MB file.

5.5.2 Experiment Results: Performance of Leechers in Network without Free-Riders

We compare a system consisting of all leechers adopting the regular BitTorrent protocol, to a system consisting of all leechers adopting the proposed protocol based on the foresighted strategy. In this section, we assume that there is no free-rider in the networks. Fig. 5.8 shows the download completion time of leechers. For each group of leechers having the same upload capacity, separate boxplots are depicted [RL78] for the different scenarios. The top and the bottom of the boxes represent the 75th and the 25th percentile sample of download time, respectively, over all 5 runs of the experiments. The markers inside the boxes represent the median, while the vertical lines extending above and below the boxes represent the maximum and minimum of samples of download time within the ranges of 1.5 time the box height from the box boarder. Outliers are marked individually with "+" mark.

The results show the clear performance difference among high-capacity leechers, which are the fastest 20% leechers, and low-capacity leechers, which are the slowest 80% leechers. High-capacity leechers can significantly improve their download completion time – leechers having the upload capacity of at least 18kB/sec improve their download completion time by up to 33% in median. Unlike in the regular BitTorrent system, where leechers determine their peer selection decisions based on the myopic tit-for-tat that uses only the last reciprocation history, the leechers adopting the foresighted strategy determine their peer selection decisions based on the long-term history. This enables the leechers to estimate the behaviors of their associated peers more accurately. Moreover, since part of the peer selection decisions is randomly determined in the regular BitTorrent, there is a high probability that high-capacity leechers need to reciprocate with



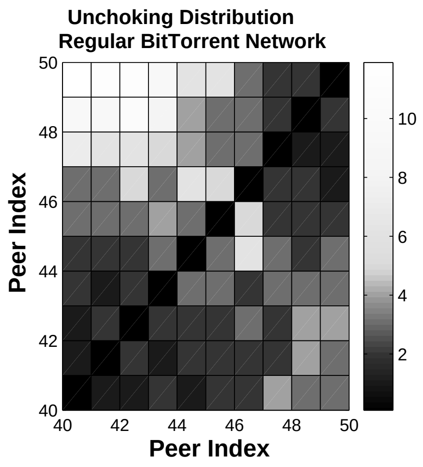


Figure 5.9: Unchokes among the 20% fastest peers

the low-capacity leechers [PIA07]. However, the randomly determined peer selection decisions are significantly reduced in the proposed approach, as the random decisions are taken only in the initialization phase or in order to collect the reciprocation history of newly joined peers. As a result, the high-capacity leechers increase the probability to reciprocate resources with the other high-capacity leechers.

This is confirmed in the results of Fig. 5.9, which shows the unchoking percentage among the 20% high-capacity leechers, comparing the two different systems. It is clearly observed that the collaboration among high-capacity leechers improves when leechers adopt the foresighted strategy. Thus, we can conclude that the foresighted strategy improves the incentive mechanisms in BitTorrent networks: as a leecher contributes more to the network, it achieves higher download rate.

Recent studies [PIA07, Izh09, BHP06, GCX05] show that the regular BitTorrent protocol suffers from unfairness particularly for high capacity leechers. In Fig. 5.10, we compare the upload rates and the average download rates of

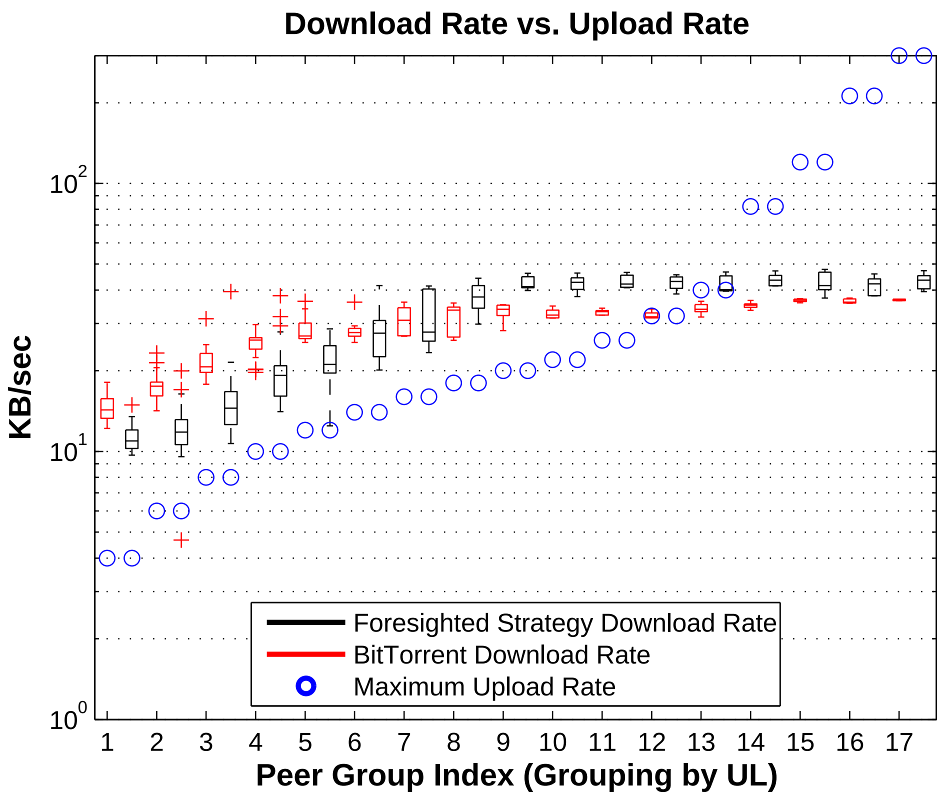


Figure 5.10: Download rate vs. upload rate

the leechers. The ratio of these values can indicate the degree of fairness in the system. The results in Fig. 5.10 show that fairness is improved as the leechers adopt the foresighted strategy, since high-capacity leechers increased their download rate getting closer to their upload rate, in spite of the restriction of limited seeds' upload rate. On the other hand, in the system where leecher adopts the foresighted strategy, the download rates of low-capacity leechers decrease, getting close by at most 36% to their upload rates, compared to the regular BitTorrent system. However, all the peers that are slowed down by the foresighted strategy still download faster than their upload rate.

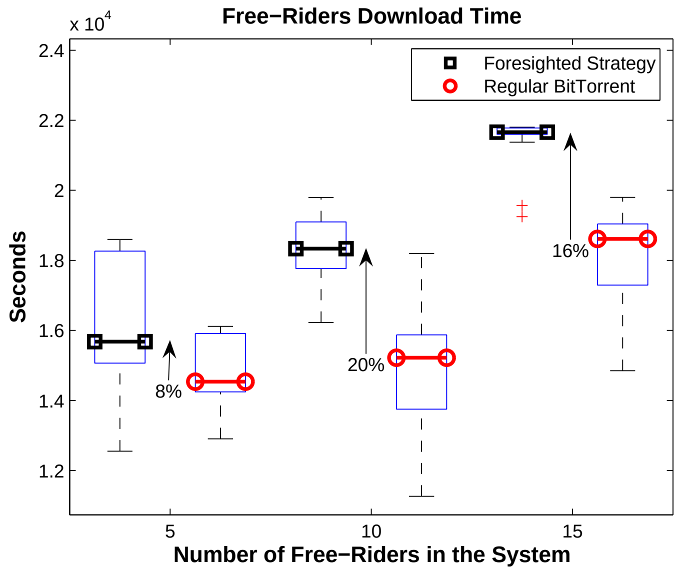


Figure 5.11: Download completion time for free-riders

5.5.3 Experiment Results: Performance of Leechers in Network with Free-Riders

In this section, we investigate how effectively the proposed protocol can prevent selfish behaviors such as free-ridings. Note that the foresighted strategy shows a similar performance for the leechers that upload their content in the network that includes free-riders (i.e., shows the improved fairness, etc.). Hence, in this section, our focus is on studying how the free-riders are punished due to their selfish behaviors. Fig. 5.11 shows the time that the free-riders complete downloading 100MB file in a network consists of 50 contributing leechers, and increasing number of free-riders (i.e., 5, 10, and 15 free-riders). It compares the results of the foresighted strategy system to the regular BitTorrent system. The Figure confirms that the foresighted strategy enables the leechers to effectively penalize the free-riders, as it takes longer time for the free-riders to complete their downloads (requires 8%-20% more time as measured by the median, in comparison to the

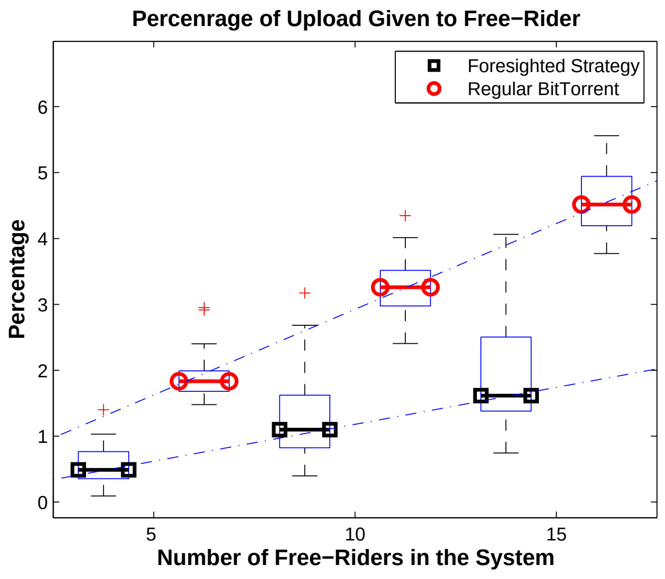


Figure 5.12: Percentage of free-riders' download from contributing leechers

regular BitTorrent protocol).

When leechers adopt the foresighted strategy, they can efficiently capture the selfish behavior of the free-riders. Thus, they unchoke the free-riders with a significantly lower probability. Hence, the free-riders can download their content mainly from seeds not from the leechers. The results shown in Fig. 5.12 also confirm that the leechers in the regular BitTorrent system upload approximately 2.8-3.7 times more data to the free-riders compared to the leechers in the system where foresighted strategy is adopted. This also shows that the P2P networks consist of the leechers adopting the foresighted strategy are more robust to the selfish behaviors of peers than the networks operating with the regular BitTorrent protocol. For example, in the network with 15 free-riders, the leechers in the regular BitTorrent systems upload 4.5% of their total upload capacity to free-riders, while they only upload 1.6% of their upload capacity in the foresighted strategy system.

Therefore, our experiment results confirm that the foresighted strategy provides more incentives for leechers to maximize their upload rate by enabling the leechers to discourage non-cooperative behaviors such as free-riding, improves fairness, and enhances the robustness of the network.

Algorithm 1 Peer-Set Reduction Algorithm

```
1: INPUT :
     · $C_j$ - set of associated peers of peer $j$
     · $T$ - target output set size (constant)
     · $R_{i,j}$ - estimated rewards of peer $i$
     · $P(S_j)$ - the probability to be in state $S_j$
     · $Pr(i,j)$ the resource reciprocation probability of peer $i$
     · $c_1, c_2$ - constants such that $T \gg c_1 > c_2$
2: OUTPUT : A reduced set of peers $C'_j \subseteq C_j$ s.t. $|C'_j| = T$
3: for all $i \in C_j$ do
4:     $K_{i,j} = R_{i,j} \times Pr(i,j)$;
5: order $C_j$ in a non-decreasing order of the $K_{i,j}$;
6: $C'_j = C_j$;
7: while $|C'_j| > T$ do
8:     G = $\left\langle C'_{j1}, ..., C'_{j_{c_1}} \right\rangle$;
9:     calculate $\pi^*_{j,G}$ the optimal policy for the set $G$;
10:    for all $i \in G$ do
11:        $Pu_i = 0$
           //Calculate $Pu_i$, the probability
           //that $j$ unchokes $i$ using $\pi^*_{j,G}$ policy;
12:        for all $s_j \in S_j$ do
13:            if $\pi^*_{j,G}(s_{(i,j)}) = 1$ then
14:                $Pu_i = Pu_i + P(s_j)$;
15:    order $G$ in a non-decreasing order of the $Pu_i$ values;
16:    if $c_2 > |C'_j| - T$ then
17:        $c_2 = |C'_j| - T$;
18:    $C'_j \leftarrow C'_j - < G_1, ..., G_{c_2} >$;
19: return $C'_j$
```

CHAPTER 6

Related Work

The main mechanisms and design rationale of BitTorrent were first described by Bram Cohen, the protocol creator, in [Coh03]. Since then, an extensive research work has focused on performance and behavior of BitTorrent systems. In this chapter we describe related works with respect to better understanding of the protocol properties, the protocol user incentives, free-riding opportunities, and lack of fairness in the protocol. In addition, we describe some game theoretic works that are related to P2P networks in general.

6.1 Analytical Models for BitTorrent Systems

Several analytical models for the BitTorrent system were developed. Qiu and Srikant [QS04] developed a fluid analytical model of BitTorrent systems and studied the peer selection mechanism and its effect on peer performance. They observed that optimistic unchokes may provide a way for leechers to free-ride. Their model assumes that peer selection is based on global knowledge of the upload rate of all the peers in the torrent, as well as uniform distribution of pieces. Moreover, they claim that the system with tit-for-tat strategy eventually converges to Nash equilibrium where fairness is achieved and all peers download at their upload capacities. Nevertheless, this claim was shown to fail in realistic BitTorrent swarms [PIA07, Izh09, GCX05, LLS08]. More recently, Fan *et al.* [FCL06] characterized the incentive design space for BitTorrent-like protocols, capturing a fundamental trade-off between performance and fairness. Our $team - enhanced$, and $buddy - enhanced$ protocol are inspired by this work. In [MV05], Massouliè *et al.* introduced a probabilistic model of coupon replication systems. They argue that performance of a file sharing system such as BitTorrent does not depend critically on altruistic behavior nor on piece selection strategy as rarest first strategy. Although all these models provide valuable insight into the behavior of the BitTorrent protocol, unrealistic assumptions limit their relevance in real scenarios [GCX05, PGE05].

Other works model the peer selection mechanism in BitTorrent as a game with strategic peers. Jun *et al.* [JA05] provide a game theoretic framework for understanding the peer selection mechanism of BitTorrent by relating the BitTorrent to the iterated Prisoner's Dilemma. Levin *et al.* [LLS08] propose an auction base model to model the peer selection mechanism, claiming that BitTorrent uses auction to decide which peers to unchoke and not the tit-for-tat as widely believed.

6.2 BitTorrent Measurement Study

In addition to the analytical models of BitTorrent, some measurement studies have attempted to characterize BitTorrent properties. These studies examined real BitTorrent traffic of public torrents. Izal *et al.* [IUB04] analyzed the BitTorrent based on measurements collected during a five months long time interval that involved thousands of peers. Their conclusions indicate that BitTorrent is a realistic and inexpensive alternative. However, while there is a correlation between upload and download rates of peers, indicating the effectiveness of the peer selection mechanism, only few leechers and the seeds contribute the majority of the content. Pouwelse *et al.* [PGE05] performed a measurement study focusing on availability, integrity, flash crowd handling and download performance. They observed that the tracker, which is a centralized entity, might potentially be a bottleneck in the system. This work encouraged us to develop the distributed $buddy - enhanced$ protocol that does not add any task to the tracker. Guo *et al.* [GCX05] performed extensive measurements of real torrents and pinpointed some of the BitTorrent limitations i.e. exponentially decreasing peer arrival rate that leads to poor service availability, unstable client performance that fluctuates widely with the peer population and the existing lack of fairness. Legout *et al.* [LUM06] studied the efficiency of the rarest piece first selection algorithm on real BitTorrent networks. Their conclusions are that the rarest first algorithm guarantees a high diversity of the pieces. In particular, it prevents the last pieces problem and the reappearance of rare pieces. More recently, Legout *et al.* [LLK07] observed that tit-for-tat incentives tend to cause leechers to cluster together with others having similar upload bandwidth. Both our $team-enhanced$ and $buddy - enhanced$ protocols facilitate this explicitly by having the tracker assigns leechers with similar upload rate to the same team in the $team-enhanced$ protocol and by encouraging collaboration among buddies having similar upload rate in the $buddy - enhanced$ protocol.

6.3 Free-Riding in BitTorrent Systems

Free-riders are those peers who attempt to circumvent the protocol mechanism and download data without uploading any data to other peers in the network. Wide adoption of free-riding strategy can result in a tragedy of the commons where over all performance in the system will decrease. Researchers have argued that free-riding in BitTorrent is feasible through the optimistic unchoke mechanism and through seeds.

The first to pinpoint that effective free-riding in BitTorrent is feasible were Shneidman *et al.* in [SPM04]. They identified forms of strategic manipulation that are based on Sybil attacks and uploading garbage data. Jun *et al.* [JA05] also argued that free-riding is feasible in BitTorrent by investigating the incentives in BitTorrent peer selection mechanism. They proposed a new algorithm to enforce fairness in peer data exchanges. They showed that removing optimistic unchokes increases the average completion time, but punishes free-riders more heavily. Liogkas *et al.* [LNK06] showed that free-riding in BitTorrent is feasible by implementing three selfish BitTorrent exploits that allow free-riders to achieve high download rates, and evaluating their effectiveness under specific circumstances. Locher *et al.* [LMS06] extended those results, presenting the BitThief, a free-riding client that combs several attacks. They demonstrated that free-riding is feasible even in the absence of seeds. More recently, Sirivianos *et al.* [SPC07] evaluated an exploit based on maintaining a view of the torrent that is larger-than-normal. This exploit affords free-riders a much higher probability of receiving data from seeds and via optimistic unchokes. They argued that large view exploit is effective and has the potential for wide adoption.

The aforementioned studies identify one of the most important vulnerabilities, namely exploiting optimistic unchokes to download data for free. The works we present in this thesis directly address this by limiting (or replacing) the optimistic unchokes. Thus, hurting the performance of free-riders while improving the performance of these who contribute to the system by uploading to other peers.

Complementary to our approach, some work has attacked the ability of free-riders to download data from seeds without uploading in return, intending to considerably hurt free-riders' performance. Locher *et al.* [LSW07] proposed a source-coding scheme to replace tit-for-tit incentives. Seeds in this scheme only upload a fixed number of content pieces to each leecher they are connected to, thus placing a hard limit on the data free-riders can obtain in this manner. Chow *et al.* [CGM07] presented another modified seed peer selection algorithm that gives preference to leechers who are either at the beginning or at the end of their download. They showed that this algorithm considerably hurts free-riders' performance.

Finally, reputation systems, such as [BB04, XL04, YZL05, eDo], which use an additional peers' reputation history to make the peer selection decision, also help to limit the ability to free-ride.

6.4 Fairness in BitTorrent Systems

Fairness in BitTorrent systems has been heavily discussed in the literature. Guo *et al.* [GCX05] and Izal *et al.* [IUB04] both observed lack of fairness in BitTorrent in their measurement studies on real torrents. Another measurement study by Piatek *et al.* [PIA07] observed the presence of significant altruism, where peers make contributions that do not directly improve their performance. Bharambe *et al.* [BHP06] utilized a discrete event simulator to evaluate the impact of BitTorrent mechanisms such as the peer selection mechanism. They observed that rate-based tit-for-tat incentives couldn't guarantee fairness across peers. They suggest a block-based tit-for-tat policy to improve fairness.

Legout *et al.* [LLK07] studied clustering of peers having similar upload bandwidth. They observed that when the seed is underprovisioned, all peers tend to complete their downloads approximately at the same time, regardless of their upload rates. Moreover, high-capacity peers assist the seed to disseminate data to low-capacity peers.

Obviously, one of the reasons for the lack of fairness in BitTorrent is that high-capacity leechers are forced to interact with low-capacity ones through optimistic unchokes [Izh09, PIA07]. Another reason is that the tit-for-tat strategy is based on short-term history. Thus, a peer can benefit from the tit-for-tat strategy only if it can continuously upload pieces of content and as long as it receives pieces of interest in return. Piatek *et al.* [PIK08] showed that this is not always possible, as peers may not have pieces that are of interest to others, especially in node populations with heterogeneous upload bandwidths. This is regardless of their willingness to cooperate; yet, this behavior is still perceived as a lack of cooperation. In Chapter 5 we use long-term history based strategy to address this problem and improve fairness. While in our $buddy-enhance$ and $team-enhance$, long term history is used to decide termination of collaboration.

In all the works we present in this thesis, we identified the potential of significant differences between leecher's upload and download rates. Our proposed protocols aim to reduce this lack of fairness.

6.5 Providing Contribution Incentives

Other researchers have also acknowledged the importance of contribution incentives in P2P systems and have proposed different alternatives. Anagnostakis *et al.* [AG04] suggested to extend the BitTorrent incentives to *n-way* exchanges among rings of peers, providing incentive to cooperate. Piatek *et al.* [PIA07] proposed the BitTyrrent client, who applies a new peer selection mechanism that reallocates upload bandwidth to maximize peers' download rates. However, whereas the appearance of a single BitTyrant client in a BitTorrent system reveals improving performance; in the case of a widespread adoption the system performs a severe loss of efficiency [CNM08]. Levin *et al.* [LLS08] proposed the propshare client that rewards other peers with proportional shares of bandwidth. They show improvement in download time of propshare client, when single or few peers run with propshare client, however when the majority of peers run with propshare client there is not much difference in performance in comparison to the regular BitTorrent protocol. As opposed to our suggested clients that were tested and discussed mainly for the case when the majority of peers adopt one of the proposed protocols.

6.5.1 Payment Systems

Payment systems (e.g.,[Wil02, VCS03, SYJ07, GPK04, P2P08, AAN07]) which enable peers to earn credits according to their uploads to other peers have been proposed using some form of virtual currency. However, although in theory payment concepts might be convincing, these concepts are not practical in real-world systems. First, a trusted third party must clear peer credit transactions to prevent cheating. This requires contacting a centralized component for every peer transaction, which arguably limits scalability. Furthermore, in order to start trading, users need startup funds. This opens up the possibility for whitewashing [FPC04, FC05a]: users could use startup funds completely, and then switch to a new identity, replenishing their available credits.

6.5.2 Reputation Systems

To overcome the weaknesses in payment systems, various design of reputation systems (e.g.,[BB04, XL04, YZL05, KSG03, LPY06, PIK08]) has been proposed.

In these systems, peers can make peer selection decisions based on private history as well as globally shared history that is measured by the past-interacted peers. For example in [LPY06] Lian *et al.* proposed multi-level tit-for-tat incentives as a hybrid between private and shared history schemes. More recently, Piatek *et al.* [PIK08] proposed a one-hop reputation system, in which reputation propagation is limited to at most one intermediary, and peers that are not interested in a current available content perform work in exchange for the assurance of future payback.

However, these reputation systems require significant communication overheads to maintain the global history. Moreover, some of them, e.g., EigenTrust [KSG03], compute the history for all participants relying on a small number of trusted peers. Furthermore, even if the shared history accurately represents peer contribution behavior, there is no guarantee that each peer expresses the same behavior to different peers with different attributes.

None of the three protocols that are described in this thesis make a piece selection decision using global history, though, the $team-enhanced$ protocol uses voting within the team to make punishment decisions.

6.5.3 Cooperative Mechanisms

Lastly, there have been some other approaches encouraging peers to cooperate in uploading content. Wong's thesis [Won04] presents a system based on volunteer *helper* peers, who heuristically download popular content pieces and upload them to others. Nevertheless, it does not provide any incentives for the helpers themselves. The 2Fast system [GIE06] enables a *collector* peer to task others to download data on its behalf. Whereas, it provides no mechanism to enforce reciprocation of this favor in the future, but rather relies simply on social (friend) pressure instead. In addition, it only helps improve the download rate of the collector.

BTSlave [BTS] is a similar BitTorrent protocol extension that also attempts to leverage a "friends" approach. Similar to 2Fast, BTSlave also suffers from the absence of a mechanism to enforce reciprocation in the future. Lastly, Wang *et al.* [WYP07] propose to utilize helpers for improved performance, by having them download a small number of random content pieces, which they then altruistically upload to those with a small fraction of the content. Their approach does not address the issue of incentives for helpers. Unfortunately, it also creates an additional opportunity for free-riders, who can misreport their download progress to the tracker to receive more data from helpers for free.

6.5.4 Foresighted Strategy

Finally, theoretical aspects of the MDP-based foresighted strategy for peers in P2P network are discussed in [PS09]. Although, this work models very simple and not realistic network setup, it motivates our design of a realistic resource reciprocation mechanism in Chapter 5.

6.6 Game Theoretic Approaches in P2P Networks

Game theoretic approach has been used extensively to study and model P2P networks in general, and to design P2P networks that provide incentives to collaborate. The first to model and analyze the utilities and costs associated with the participation in a P2P network, using tools from game theory were [GLM01] *et al.*. In [BAS03] the authors use ideas from game theory to study the interaction of strategic and rational peers in P2P network, and propose a differential service-based incentive scheme. Feldman *et al.* in [FPC04] present an economic model of user behavior. In particular they show that a mechanism that penalizes free-riders can improve system performance. Mechanism design [FS02, SP03] and double auction mechanism as PeerMart [HS05] also suggested as tools to provide incentives for node to collaborate. In [LFS03, FLS04] the authors model the free-riding problem using Evolutionary Prisoner's Dilemma on infinitely repeated games to capture tension between individual and social utility. Furthermore, an infinitely repeated game was used to model the peers' interactions in other works as [DS04, LK08]. Finally, [FC05b] surveys works at the intersection of game theory and distributed systems design.

CHAPTER 7

Conclusions

This thesis proposes and examines three different BitTorrent-like protocols:

1. The $team - enhanced$ protocol, which extends the responsibilities of the central tracker to dynamically organize peers of similar upload bandwidth in *teams*.

2. The $buddy - enhanced$ protocol, which distributively and dynamically creates *buddies*, peers with a similar upload rate that collaborate for mutual benefit.

3. The foresighted resource reciprocation protocol, which uses its private reciprocation history to capture the associated peers' statistical behaviors of resource reciprocation and the corresponding expected utility.

We performed an extensive experimental investigation of BitTorrent systems on a controlled PlanetLab testbed, while comparing the *regular* BitTorrent protocol to each of the suggested protocols. Experimental results demonstrated that our proposed protocols promote fairness. E.g., for system with free-riders, high capacity peers in $team - enhanced$, $buddy - enhanced$ and foresighted resource reciprocation systems improved their download rates by $\sim$ $15\%, 17\% - 54\%$, and $\sim$ 33%, respectively (Note that these numbers are not comparable as each protocol was tested at different times, on different sets of peers, and in some cases with different numbers of peers and free-riders). Furthermore,compared to regular BitTorrent systems, free-riders achieved lower download rates on these more robust experimental systems which are less sensitive to any increase in the number of free-riders in the network. E.g., free-riders were slowed down in $team - enhanced$, $buddy - enhanced$ and foresighted resource reciprocation systems by $100\%, 21 - 48\%$, and $8\% - 20\%$ respectively.

7.1 Comparing the Proposed Protocols

Although all three protocols target similar goals, each has its own strengths and weaknesses under particular setup and requirements of the system. The $team - enhanced$ and the $buddy - enhanced$ protocols are similar in their approaches to replacing the peer selection mechanism. However, the $team - enhanced$ protocol uses a global view of the network and thus converges faster to the peer selection steady state. This is indicated in our experiments where the usage of optimistic unchokes in the $team - enhanced$ system reduces to an average of 6.7% after 400 seconds (thus almost eliminating the need to discover the network), as compared to an average of 26% after only 510 seconds in the $buddy - enhanced$ system. Thus, the $team - enhanced$ protocol makes it more difficult for free-riders to abuse the optimistic unchokes. Our experiments confirmed this, showing that free-riders upload about 40% of the data from leechers in the $team - enhanced$ system, while this value increases to about 60% in the $buddy - enhanced$ system.

On the other hand, the $team - enhanced$ protocol applies this approach using a centralized technique. In other words, the teams are formed and managed by the tracker, a centralized authority that may potentially be a bottleneck under particular scenarios [PGE05], even without the additional team-related tasks such as formation of teams, blacklisting of peers, confirming team membership, etc.

Moreover, the $buddy - enhanced$ approach requires wide adoption of the protocol in the network, whereas the $buddy - enhanced$ protocol applies this approach in a distributed manner. Additional tasks are added only to the peers, while the tracker is not involved in the buddy-related part of the protocol at all. Moreover, it does not require a wide adoption of the protocol, i.e., even a single buddy that exists in the network can benefit from the protocol. These simplifications of the protocol make its adoption is more realistic.

Finally, while the $team - enhanced$ protocol and the $buddy - enhanced$ protocol aim to maximize the immediate utility, which can show only sub-optimal performance, the foresighted resource reciprocation protocol aims to maximize the long-term utility of participating leechers. Additionally, in this protocol the optimistic unchokes are limited and used mainly as a bootstrap method when a new peer joins the network. Moreover, just like in the $buddy - enhanced$ protocol, this protocol applies a distributed technique that adds tasks only to the peers, but not to the tracker. On the other hand, this protocol frequently requires calculating the peer selection policy using reinforcement learning methods. The usage of reinforcement learning methods adds more complexity to the peers, and increases the memory and the computational requirements. E.g., finding the reinforcement learning policy for 8 peers requires $c \times 2^7 \times 2^7 \times \binom{7}{N_u}$ CPU instructions, where c is a constant that stands for the number of CPU instructions in a single iteration. 2^7 is the number of possible states at time t, and similarly at time $t+1$. $\binom{7}{N_u}$ is the number of optional actions given that the number of unchoke slots is limited to N_u. In each iteration the reward and the probability for the given states at time t, time $t+1$ and the action are calculated.

7.2 Future Work

This work opens up many directions for future research. One major piece that is still missing in this work is an extensive experiment that compares the three protocols under the same experiment setup and confirms the strengths of each one in comparison to the other, with respect to parameters that were discussed in this thesis such as fairness and resistance to free-riding, as well as other like greediness ability of a single client, decision fluctuation, convergence to the steady state decision, upload rate utilization, system performance under churn, etc. Moreover, comparing these three protocols to other existing BitTorrent-like protocols such as BitTyrent [PIA07], and PropShare [LLS08], would be interesting as well.

In addition, most of the related works as well as this thesis study only one levels of deviation from the protocol, which is the free-riding. It would be interesting to study different level of deviation from the protocol, for example, peers that run more than one client simultaneously, thus benefiting from the sub-linear behavior of the $\frac{download\ rate}{upload\ rate}$ ratio. Another example can be peers that aim to maximize their utility by being free-riders only for fraction of the time.

During our extensive experiments, we collected a large amount of data. We believe that this data is valuable and may help us to understand other parameters and limitations of the BitTorrent network, such as the structure of the BitTorrent overlay network, peer selection convergence, etc.

In addition, in relation to the stochastic model, a possible direction is to test other existing heuristic solutions for a stochastic-game framework, as well as looking into a Nash-Equilibrium solution for the defined game.

Finally, in this work we aim to improve the peer selection mechanism of the leechers. However, free-riding exists also because of the peer selection mechanism in the seeds [CGM07, LSW07]. Currently, the common peer selection mechanism in seeds is the trivial round-robin selection [Coh]. Although a seed is an altruistic entity, it would be interesting to model its peer selection decision as a learning problem, where the incentive of the seed can be to improve the social welfare.

References

[AAN07] Ramachandran Anirudh, Das Sarma Atish, and Feamster Nick. "BitStore: An incentive compatible solution for bloked downloads in BitTorrent." In *NetEcon*, 2007.

[AG04] Kostas G. Anagnostakis and Michael B. Greenwald. "Exchange-based Incentive Mechanisms for Peer-to-Peer File Sharing." In *ICDCS*, 2004.

[AH00] E. Adar and B.A. Huberman. "Free Riding in Gnutella." In *First Monday*, 2000.

[BAS03] C. Buragohain, D. Agrawal, and S. Suri. "A game-theoretic framework for incentives in p2p systems." International Conference on Peer-to-Peer Computing, 2003.

[BB04] S. Buchegger and J.-Y. le Boudec. "A Robust Reputation System for P2P and Mobile Ad-hoc Networks." In *Economics of P2P Systems*, 2004.

[BBC] Andy Bavier, Mic Bowman, Brent Chun, David Culler, Scott Karlin, Steve Muir, Larry Peterson, Timothy Roscoe, Tammo Spalink, and Mike Wawrzoniak. "Operating System Support for Planetary-Scale Network Services." In *NSDI'04*.

[Ber76] Dimitri P. Bertsekas. *Dynamic Programming and Stochastic Control*. Academic P, 1976.

[BHP06] A. Bharambe, C. Herley, and V. Padmanabhan. "Analyzing and Improving a BitTorrent Network's Performance Mechanisms." In *INFOCOM*, 2006.

[BNB] BNBT EasyTracker. "http://bnbteasytracker.sourceforge.net.".

[BTS] BTSlave protocol page. "http://btslave.sourceforge.net.".

[BV01] M. Bowling and M. Veloso. *"Rational and convergent learning in stochastic games."*. Master's thesis, Seventeenth International Joint Conference on Artificial Intelligence (IJCAI), 2001.

[CGM07] Alix L.H. Chow, Leana Golubchik, and Vishal Misra. "Improving BitTorrent: A Simple Approach." In *IPTPS*, 2007.

[CMS04] Gkantsidis C., Mihail M., and Saberi S. "Random Walks in Peer-to-Peer Networks." In *INFOCOM*, 2004.

[CNM08] Damiano Carra, Giovanni Neglia, and Pietro Michiardi. "On the Impact of Greedy Strategies in BitTorrent Networks: the Case of BitTyrant." In *P2P*, 2008.

[Coh] Bram Cohen. "The BitTorrent Protocol Specification." In *ver.11031*.

[Coh03] Bram Cohen. "Incentives Build Robustness in BitTorrent." In *P2PEcon*, 2003.

[DL07] Cameron Dale and Jiangchuan Liu. "A Measurement Study of Piece Population in BitTorrent." In *IEEE GLOBECOM*, 2007.

[DS04] L. DaSilva and V. Srivastava. "Node Participation in Adhoc and Peer-to-peer Networks: A Game-theoretic Formulation." In *Workshop on Games and Emergent Behavior in Distributed Computing Environments*, 2004.

[eDo] eDonkey. "http://www.edonkey2000.com/.".

[Enh] Enhanced CTorrent. "http://www.rahul.net/dholmes/ctorrent.".

[Fas] Fasttrack peer-to-peer technology company. "http://www.fasttrack.nu.".

[FC05a] M. Feldman and J. Chuang. "The evolution of cooperation under cheap pseudonyms." In *Proceedings of the 7th International IEEE Conference on E-Commerce Technology*, 2005.

[FC05b] Michal Feldman and John Chuang. "Overcoming free-riding behavior in peer-to-peer systems." In *SIGecom Exch*, 2005.

[FCL06] Bin Fan, Dah-Ming Chiu, and John C.S. Lui. "The Delicate Tradeoffs in BitTorrent-like File Sharing Protocol Design." In *ICNP*, 2006.

[FL99] D. Fudenberg and D. K. Levine. *"The theory of learning in games."*. Master's thesis, Cambridge, MA: MIT Press, 1999.

[FLS04] M. Feldman, K. Lai, I. Stoica, and J. Chuang. "Robust incentive techniques for peer-to-peer networks." In *In ACM Electronic Commerce*, 2004.

[FPC04] Michal Feldman, Christos Papadimitriou, and John Chuang. "Free-Riding and Whitewashing in Peer-to-Peer Systems." In *PINS*, 2004.

[FS02] Joan Feigenbaum and Scott Shenker. "Distributed Algorithmic Mechanism Design: Recent Results and Future Directions." In *In Proceedings of the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, 2002.

[G68] Hardin G. *The Tragedy of the Commons*. 1968.

[Gal96] R. G. Gallager. "Discrete stochastic processes." 1996.

[GCX05] Lei Guo, Songqing Chen, Zhen Xiao, Enhua Tan, Xiaoning Ding, and Xiaodong Zhang. "Measurements, analysis, and modeling of BitTorrent-like systems." In *IMC*, 2005.

[GIE06] Paweł Garbacki, Alexandru Iosup, Dick Epema, and Marten van Steen. "2Fast: Collaborative downloads in P2P networks." In *P2P*, 2006.

[GLM01] Philippe Golle, Kevin Leyton-Brown, and Ilya Mironov. "Incentives for sharing in peer-to-peer networks." In *EC '01: Proceedings of the 3rd ACM conference on Electronic Commerce*, pp. 264–267, New York, NY, USA, 2001. ACM.

[Gnu] Gnutella: The Protocol Specification, v0.4. "http://www9.limewire.org/developer/gnutella_protocol_0.4.pdf.".

[GPK04] D. Ghosal, B. K. Poon, and K. Kong. "P2P contracts: a framework for resource and service exchange." In *FGCS*, 2004.

[HCW05] D. Hughes, G. Coulson, and J. Walkerdine. "Free riding on Gnutella revisited: the bell tolls?" In *IEEE In Distributed Systems Online*, 2005.

[HS05] David Hausheer and Burkhard Stiller. "PeerMart: The Technology for a Distributed Auction-based Market for Peer-to-Peer Services." In *ICC*, 2005.

[HW] J. Hu and P. Wellman. "Multiagent reinforcement learning: theorectical framework and an algorithm.".

[ILM09] Rafit Izhak-Razin, Nikitas Liogkas, and Rupak Majumdar. "Team Incentives in BitTorrent Systems." In *ICCCN*, 2009.

[IPO09] IPOQUE Internet Measurements 2008-2009. "IPOQUE, http://www.ipoque.com/." 2008-2009.

[IPS10] Rafit Izhak-Ratzin, Hyunggon Park, and Mihaela van der Schaar. "Foresighted Resource Reciprocation Strategy in BitTorrent Systems." In *under submission*, 2010.

[IUB04] M. Izal, G. Urvoy-Keller, E. W. Biersack, P.A. Felber, A. Al Hamra, and L. Garces-Erice. "Dissecting BitTorrent: Five Months in a Torrents Lifetime." In *PAM*, 2004.

[Izh09] Rafit Izhak-Razin. "Collaboration in BitTorrent Systems." In *Networking*, 2009.

[JA05] Seung Jun and Mustaque Ahamad. "Incentives in BitTorrent Induce Free Riding." In *P2PEcon*, 2005.

[Kaz01] Kazaa. "http://www.kazaa.com/." 2001.

[KSG03] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. "The EigenTrust Algorithm for Reputation Management in P2P Networks." In *WWW'03*, 2003.

[LFS03] K. Lai, M. Feldman, I. Stoica, and J. Chuang. "Incentives for cooperation in peer-to-peer networks." In *Economics of P2P Systems*, 2003.

[LK08] Jian Lian and MacKie-Mason Jeffrey K. "Why Share in Peer-to-Peer Networks?" In *International Conference on Electronic Commerce (ICEC'08)*, 2008.

[LLK07] Arnaud Legout, Nikitas Liogkas, Eddie Kohler, and Lixia Zhang. "Clustering and Sharing Incentives in BitTorrent Systems." In *SIGMETRICS*, 2007.

[LLS08] Dave Levin, Katrina LaCurts, Neil Spring, and Bobby Bhattacharjee. "Bittorrent is an auction: analyzing and improving bittorrent's incentives." *Sigcomm*, 2008.

[LMS06] Thomas Locher, Patrick Moor, Stefan Schmid, and Roger Wattenhofer. "Free Riding in BitTorrent is Cheap." In *HotNets-V*, 2006.

[LNK06] Nikitas Liogkas, Robert Nelson, Eddie Kohler, and Lixia Zhang. "Exploiting BitTorrent For Fun (But Not Profit)." In *IPTPS*, 2006.

[LPY06] Qiao Lian, Yu Peng, Mao Yang, Zheng Zhang, Yafei Dai, , and Xiaoming Li. "Robust Incentives via Multi-level Tit-for-tat." In *IPTPS*, 2006.

[LSW07] Thomas Locher, Stefan Schmid, and Roger Wattenhofer. "Rescuing Tit-for-Tat with Source Coding." In *P2P*, 2007.

[LUM06] Arnaud Legout, G. Urvoy-Keller, and P. Michiardi. "Rarest first and choke algorithms are enough." In *IMC*, 2006.

[MV05] L. Massouliè and M. Vojnovic. "Coupon Replication Systems." In *SIGMETRICS*, 2005.

[Nap] Napster. "http://www.napster.com/.".

[NPZ07] G. Neglia, G. L. Presti, H. Zhang, and D. Towsley. "A network formation game approach to study BitTorrent tit-for-tat." In *NET-COOP*, 2007.

[OR94] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory.* MIT Press, 1994.

[P2P08] Stochastic Optimization for Content Sharing in P2P Systems. "van der Schaar Mihaela and Deepak S. Turaga and Ritesh Sood." In *IEEE Transactions on*, 2008.

[Pap01] C. Papadimitriou. "Algorithms, games, and the internet." In *STOC*, 2001.

[PGE05] J.A. Pouwelse, P. Garbacki, D.H.J. Epema, and H.J. Sips. "The BitTorrent P2P file-sharing system: Measurements and Analysis." In *IPTPS*, 2005.

[PIA07] Michael Piatek, Tomas Isdal, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. "Do incentives build robustness in BitTorrent?" In *NSDI*, 2007.

[PIK08] Michael Piatek, Tomas Isdal, Arvind Krishnamurthy, and Thomas Anderson. "One hop reputations for peer to peer file sharing workloads." In *NSDI*, 2008.

[PS09] Hyunggon Park and M. van der Schaar. "A Framework for Foresighted Resource Reciprocation in P2P Networks." **11**(1):101–116, January 2009.

[QS04] Dongyu Qiu and R. Srikant. "Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks." In *SIGCOMM*, 2004.

[RD01] Antony Rowstron and Peter Druschel. "Pastry: Scalable, distributed object location and routing for largescale peertopeer systems." In *Proceedings of ACM Middleware*, 2001.

[RL78] John W. Tukey Robert McGill and Wayne A. Larsen. "Variations of box plots." *The American Statistician*, **32**:12–16, 1978.

[Sha53] L. S. Shapley. *"Stochastic games."*. vol. 39, 1095-1100, Proceedings of the National Academy of Sciences of the United States of America, 1953.

[SMK01] Ion Stoica, Robert Morris, David Karger, M. Franskaashoek, Frank Dabek, and Hari Balakrishnan. "Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications." In *SIGCOMM*, 2001.

[SP03] Jeffrey Shneidman and David C. Parkes. "Rationality and Self-Interest in Peer to Peer Networks." In *IPTPS*, 2003.

[SPC07] Michael Sirivianos, Jong Han Park, Rex Chen, and Xiaowei Yang. "Free-riding in BitTorrent Networks with the Large View Exploit." In *IPTPS*, 2007.

[SPD02] Saroiu Stefan, Gummadi Krishna P., and Gribble Steven D. "A Measurement Study of Peer-to-Peer File Sharing Systems." In *Multimedia Computing and Networking (MMCN)*, 2002.

[SPM01] Ratnasamy S., Francis P., Handley M., R. Karp, and Shenker S. A. "Scalable Content-Addressable Network." In *Sigcomm*, 2001.

[SPM04] Jeffrey Shneidman, David C. Parkes, and Laurent Massouliè. "Faithfulness in Internet Algorithms." In *PINS*, 2004.

[SYJ07] M. Sirivianos, J. H. P. X. Yang, and S. Jarecki. "Dandelion: Cooperative Content Distribution with Robust Incentives." In *USENIX*, 2007.

[VCS03] Vivek Vishnumurthy, Sangeeth Chandrakumar, and Emin Gun Sirer. "KARMA: A Secure Economic Framework for Peer-to-Peer Resource Sharing." In *P2PEcon*, 2003.

[Wil02] Bryce Wilcox-O'Hearn. "Experiences Deploying A Large-Scale Emergent Network." In *IPTPS*, 2002.

[Won04] Joseph Hao Tan Wong. *"Enhancing Collaborative Content Delivery with Helpers."*. Msc thesis, University of British Columbia, September 2004.

[WYP07] Jiajun Wang, Chuohao Yeo, Vinod Prabhakaran, and Kanna Ramchandran. "On the role of helpers in peer-to-peer file download systems: design, analysis, and simulation." In *IPTPS*, 2007.

[XL04] Li Xiong and Ling Liu. "PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities." In *IEEE Transactions on Knowledge and Data Engineering, 16(7).*, 2004.

[YZL05] Mao Yang, Zheng Zhang, Xiaoming Li, and Yafei Dai. "An Empirical Study of Free-Riding Behavior in the Maze P2P File-Sharing System." In *IPTPS*, 2005.

[ZHS04] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, , and John D. Kubiatowicz. "Tapestry: A resilient globalscale overlay for service deployment." In *IEEE Journal on Selected Areas in Communications, 22(1):41-53*, 2004.