Credits in BitTorrent: designing prospecting and investments functions

Ardhi Putra Pratama Hartono

Delft University of Technology

Credits in BitTorrent: designing prospecting and investment functions

Master's Thesis in Computer Science

Parallel and Distributed Systems group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Ardhi Putra Pratama Hartono

March 13, 2017

Author
Ardhi Putra Pratama Hartono

Title
Credits in BitTorrent: designing prospecting and investment functions

MSc presentation
TODO GRADUATION DATE

Graduation Committee

Prof. Dr. Ir. J.A. Pouwelse (supervisor)	Delft University of Technology
Prof. Dr. Ir. S. Hamdioui	Delft University of Technology
Dr. Ir. C. Hauff	Delft University of Technology

Abstract

Slow download speed in the BitTorrent community is mainly caused by the existence of freeriders. The credit system, as one of the most widely implemented incentive mechanisms, is designed to tackle this issue. However, in some cases, gaining credit efficiently is difficult. Moreover, the supply and demand misalignment in swarms can result in performance deficiency. As an answer to this issue, we introduce a credit mining system, an autonomous system to download pieces from selected swarms in order to gain a high a upload ratio.

Our main work is to build a credit mining system. Specifically, we focused on an algorithm to invest the credit in swarms. This is composed of two stages: *prospecting* and *mining*. In *prospecting*, swarm information is extensively collected and then filtered. In *mining*, swarms are sorted by their potential and then selected. We also propose a *scoring policy* as a method to quantify swarms with a numerical score. Each detail of the sub-algorithm is presented and elaborated.

Finally, we implemented and evaluated the credit mining system in both live and controlled environments. The system is fully integrated with Tribler and is able to adapt to user activity, while correctly selecting undersupplied swarms. In terms of advantages, users can gain an upload/download ratio of up to 4.91 by using 80% of their resources. The majority of the swarms in the community also get their average download speed increased by up to 34.6%. Based on the results, we showed that the implementation of the credit mining system is beneficial for both parties, especially considering the freeriding phenomenon.

Preface

I praise to The Almighty God for all of His countless blessing and protection.

This thesis is the finish line of my work as an MSc student in Delft University of Technology. Firstly, I would like to express my gratitude to my supervisor, Johan Pouwelse for pushing me and providing feedback on my work. Your excitement and enthusiasm is truly contagious. Secondly, I would like to gratefully acknowledge Endowment Fund for Education / Lembaga Pengelola Dana Pendidikan (LPDP) for giving me an opportunity to purse my master's degree. I will do my best to give back everything to my lovely country. Next, I would like to thank everyone in the 7th floor, Elric, Martijn, and Ma for supporting my work. You guys made my time in doing this work more enjoyable.

Special thanks go to my beloved wife, Hilda Zaikarina. Your wonderful support and patience lead me to the point where I need to *istiqomah* for all the work I have done.

Last but not least, I would like to thank my family and friends for their support and motivation throughout not only my thesis, but also my study in The Netherlands.

Ardhi Putra Pratama Hartono

Delft, The Netherlands
March 13, 2017

Contents

Chapter 1

Introduction

Since the introduction of the Internet in the '60s and the world wide web (WWW) in the '90s, more and more people have been connected to each other. Recent research shows that peer-to-peer (P2P) Internet is already back at its prime as it is dominating the traffic as shown in Figure 1.1 [1]. User interaction in the Internet community can be expressed in various fashions. Many applications and protocols run on top of P2P system, for instance online gaming, computing, and file sharing. Specifically in file-sharing P2P applications, to ensure all users have a flawless experience, it is necessary to have all peers participate equally. In spite of that, not all of the peers consistently share the file. This phenomenon is called *freeriding*.

1.1 The freeriding phenomenon

Among all of the peer-to-peer implementations on the Internet, file-sharing is the most popular one. Gnutella was one of the most popular applications from 2000-2007. However, it was shut down because of legal and performance issues. In Gnutella, the majority of users (70%) stopped sharing their files. Moreover, about half of the requests were served by only the top 1% of the community [2]. Gnutella suffers from a social phenomenon called *freeriding* with the majority of its users.

Freeriding is defined as user behavior that selfishly consumes all the resources of a system without giving back anything in return. It may cause vulnerabilities in the system [2]. With only a few of the users providing the service for many, it eventually becomes more of a centralized rather than a decentralized system. It also may degrade system performance [2]. Adar and Huberman showed that many P2P peers always show self-interest and rationalization that can be considered as freeriding. If freeriders become the majority in a file-sharing peer-to-peer system, they will occupy a significant amount of resources, and eventually bottlenecks in the system will occur. As the time goes on, an honest, important peer may feel dissatisfied and decide to leave the system, taking crucial files with them. The system then becomes degraded, and sooner or later will be completely abandoned by all of its peers.

	Upstream		Downstream		Aggregate	
Rank	Application	Share	Application	Share	Application	Share
1	BitTorrent	21.08%	YouTube	24.44%	YouTube	21.16%
2	HTTP	12.53%	HTTP	15.39%	HTTP	14.94%
3	YouTube	7.51%	Facebook	7.56%	BitTorrent	8.44%
4	SSL - OTHER	7.43%	BitTorrent	6.07%	Facebook	7.39%
5	Facebook	6.49%	SSL - OTHER	5.51%	SSL - OTHER	5.81%
6	Skype	4.78%	Netflix	4.82%	Netflix	4.18%
7	eDonkey	3.67%	MPEG - OTHER	3.82%	MPEG - OTHER	3.51%
8	MPEG - OTHER	1.89%	iTunes	2.24%	iTunes	2.03%
9	Apple iMessage	1.70%	Flash Video	1.85%	Skype	1.78%
10	Dropbox	1.44%	Twitch	1.65%	Flash Video	1.59%
		68.54%		73.35%		70.84%

(a) Sandvine data for 2015 internet usage in Europe

	Upstream		Downstream		Aggregate	
Rank	Application	Share	Application	Share	Application	Share
1	BitTorrent	48.22%	YouTube	29.31%	BitTorrent	24.95%
2	QVoD	8.89%	BitTorrent	19.20%	YouTube	24.64%
3	Thunder	3.91%	HTTP	9.65%	HTTP	8.39%
4	HTTP	3.29%	Facebook	3.65%	Facebook	3.27%
5	Skype	2.10%	MPEG - OTHER	3.11%	Thunder	2.32%
6	Facebook	1.71%	Thunder	1.93%	QVoD	2.31%
7	SSL - OTHER	1.21%	SSL - OTHER	1.66%	SSL - OTHER	1.57%
8	PPStream	0.81%	Flash Video	1.21%	iTunes	1.26%
9	Dropbox	0.70%	Valve's Steam Service	1.16%	Skype	1.12%
10	Apple iMessage	0.57%	Dailymotion	0.88%	Flash Video	1.09%
		71.41%		71.76%		70.92%

(b) Sandvine data for 2015 internet usage in Asia Pasific

Figure 1.1: Traffic of the Internet by Sandvine [1]

.

Freeriding can lead to a systematically worse problem known as “the tragedy of the commons” [3]. This problem was popularized by Hardin [3] in 1968. This social dilemma emerges because of the overuse and overexploitation of shared resources by the user without any feedback from them. As Hardin stated in his paper: “Freedom in a commons brings ruin to all”, the uncontrolled participants could selfishly take common, limited resource to fulfill their goals.

Extreme freeriding is a behavior where one does not upload anything while constantly downloading data. Under current standards, this rarely happens. Instead, it is more common to find *hit and run* behavior [4]. Hit and run (HnR) is a situation in which a user finishes downloading, and then immediately stops their contribution, i.e uploading [5]. Hit and run is also often cited as one of the freeriding behaviors that peer-to-peer communities want to prevent.

1.2 BitTorrent protocol

BitTorrent [7], nowadays stands as the *de facto* file-sharing protocol on top of the peer-to-peer network. The BitTorrent protocol can be implemented by anyone, so it is not limited to any particular service such as Gnutella.

In the general view, BitTorrent consists of peers who participate in file-sharing and the *tracker*. The *tracker* is responsible for monitoring the current distribution of files and state of peers in the swarm. The *swarm* is a set of peers formed with the common purpose of downloading or uploading certain files that is represented in the `.torrent` metadata file. The static `.torrent` file, which contains information such as tracker addresses and the unique hash value of the swarm it represents, is created by a peer who wants to publish their files. Files in a swarm consist of several *pieces* or file chunks. A piece is exchanged by the peers in a particular period. A *seeder* is a type of peer who has the complete set of files and uploads (or seeds) its pieces to other peers. A *leecher* is a type of peer who downloads from a particular swarm.

BitTorrent uses a *tit-for-tat* mechanism as both reward and punishment method for its peers' behavior. This mechanism is intended to solve the fairness issue introduced by freeriders [7]. The *tit-for-tat* mechanism always prioritizes a peer who has uploaded something. *Tit-for-tat* is valid only in a scope of a single swarm. That means the state from one swarm cannot be carried into another swarm. This factor causes *tit-for-tat* to work best only in short term transactions and with limited parties. Nevertheless, Andrade et al. showed that BitTorrent indeed increased cooperation, with less than 6% of peers having not uploaded anything (extreme freeriding) [8]. As for downloading, BitTorrent always picks the *rarest piece* first, based on its availability in the swarm. This technique ensures that a complete file is distributed in the swarm.

In order to find the rarest pieces, piece information on peers is necessary. In BitTorrent, there are four methods to discover and update peer information. Those are: using centralized trackers, distributed hash table (DHT), peer exchange (PEX), and local service discovery (LSD). Towards a "trackerless" BitTorrent system, DHT allows each peer to become a tracker. LSD is specialized to find peers in a local network. PEX is a mechanism to efficiently contact a peer directly to exchange up-to-date information.

There are many BitTorrent *communities* that serve as portals to store `.torrent` files. In general, a community in BitTorrent can be divided into two categories: *public* and *private*. A public community is one in which everybody can join the swarm served by a tracker in that community. On the other hand, private communities are closed communities which can only be accessed by passing a particular requirement [9, 5]. Typically, private communities have higher performance compared to public communities [9]. Meulpolder et al. measured that private communities have 3 to 5 times faster download speeds than public communities [9]. *Private communities* typically have higher seeder-to-leecher ratio (SLR) that affects the download speed [8]. In such a community, the administrator may enforce a policy such as *Share Ratio Enforcement* (SRE). SRE defines the amount a user needs to upload before being able to download from the community [10]. Higher performance comes with a drawback: in private communities, it is very difficult to obtain a new membership and also very easy to be kicked out [11].

Table 1.1: Overview of implemented Dispersy community in Tribler [14].

Community Name	Purpose
AllChannel	Used to discover new channels and to perform remote channel search operations.
BarterCast4[15]	While currently disabled, this community was used to spread statistics about the upload and download rates of peers inside the network and was originally created as a mechanism to prevent freeriding in Tribler.
Channel	This community represents a single channel and is responsible for managing torrents and playlists inside that channel.
Multichain [16]	This community utilizes the blockchain technology, and can be regarded as the accounting mechanism that keeps track of shared and used bandwidth.
Search	This community contains functionalities to perform remote keyword searches for torrents and torrent-collecting operations.
(Hidden)Tunnel	This community contains the implementation of the Tor-like protocol that enables anonymity when downloading content and contains the foundations of the hidden seeder services protocol, used for anonymous seeding.

1.2.1 Tribler

Tribler[1] is peer-to-peer file sharing application developed at the Delft University of Technology that is compatible with the BitTorrent protocol [12]. Tribler is a fully decentralized system focused on security and anonymity. Starting with ABC (Another BitTorrent Client), Tribler currently provides content discovery, channels concept, and reputation management in a fully distributed manner. Tribler was downloaded from the official repository on the latest stable release (6.5.2) as many as 172716 times[2].

All of Tribler's main components (such as end-to-end encryption, channel discovery, and many others) rely upon a database and dissemination system called `Dispersy` [13]. Dispersy maintains and performs the communication functions between Tribler peers in a fully decentralized manner. Dispersy can circulate the message in one-to-one or one-to-many within a group of nodes called a *community*. Important *communities* are summarized in Table 1.1 [14]. We would like to focus on the *channel* community, which is a community to distribute swarms among Tribler users. Each user can create their own *channel*, add and remove torrents to it, and maintain its activity.

[1] `https://www.tribler.org/`

[2] `http://www.somsubhra.com/github-release-stats/?username=tribler&repository=tribler` (Accessed 7 February 2017)

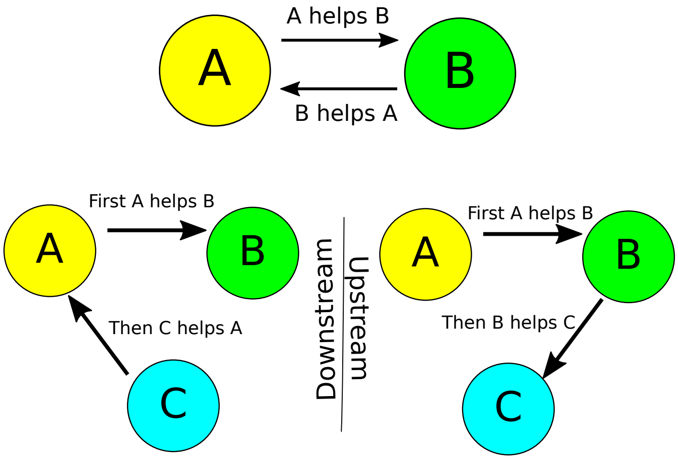


Figure 1.2: Direct and indirect reciprocation

1.3 Rewarding user contribution

BitTorrent is not enough to tackle HnR behavior. Usually, it is necessary to introduce another layer of rewarding scheme, called the *incentive mechanism*. Incentive mechanism is a method to reward the user for their contribution. In some cases, it also fines users for misconduct. The share ratio enforcement in private communities is a good example of an incentive mechanism, which increases user contribution and swarm performance. It proves that freeriding behavior can be prevented by the use of a proper incentive mechanism. For showing goodness, specifically by uploading data to others, users should get a reward. However, users are typically selfish and always try to maximize their benefit [17].

An incentive mechanism is essential as it is one of the methods by which general performance can be increased. Meulpolder discussed several kinds of incentive mechanisms that can be combined and complement each other. These are : (i) direct reciprocity, (ii) indirect reciprocity, (iii) centralized reputation, (iv) decentralized reputation, and (v) currency [4]. *Reciprocity* is focused on the relationship between peers. As shown in 1.2, it is classified into two categories: *downstream* and *upstream*. *Reputation* mechanism is more straightforward. The information of user behavior in the past is stored (either centralized or decentralized). This information is iteratively updated and spread through all the peers. The last mechanism is *currency* which uses *credit* to incentivize the user. In theory, it can emulate other incentive mechanisms by quantifying reputation/help. Because of its wide applicability, we will focus on improving the *currency* mechanism. A *private community* with SRE is the most common implementation of the currency mechanism.

The *currency* mechanism uses the concept of *credit* as a transaction unit. Therefore, it is often referred as the *credit system*. Users need to *buy* the content and can get *credit* by providing service such as uploading data to others. “Wealth”, in this case, is a collection of stored credit possessed by a particular user. The “price” of a file is the amount of credit deducted from a downloader's wealth. Credit might be asymmetrical as shown by Kash et al.[10]. Credit can also depend on the importance. For example, seeding more swarms, seeding longer and old swarms, and seeding swarms that consume large disk space [5]. Kash et al. suggest that a community should carefully declare different prices for different files. One way to do this is by lowering the price for old content, or by defining price depending on its availability and swarms' capacity [10]. However, in this work, we will assume that the credit is unrelated to the file content. It only depends on the bytes transferred between peers.

There have been several attempts to invent an incentive mechanism. Kang and Wu proposed an incentive mechanism for dynamic and heterogeneous peers using game theory. In their system, each peer can set a price for the service it provides. The buyer (downloader) is then able to negotiate with the seller (uploader) regarding the content price and its bandwidth allocation [17]. In another research, Rahman et al. proposed effort-based incentive to advocate fairness between peers [18]. In this system, the user is rewarded based on their effort, which is relative to their capacity. Currently, none of these researches are widely applicable. It either needs modification on the protocol level or only works under a certain type of application. Nowadays, the users who want to get credit may need to be on standby for a long time waiting for someone to download their files [11]. Ironically, this approach is inefficient and wastes bandwidth, but is commonly in practice[11].

1.4 Thesis structure

Our objective is to establish a layer, namely the credit mining system, on top of the existing currency system in order to lessen the hit and run (HnR) effect on BitTorrent communities, and to automatically let users gain credit efficiently by uploading only necessary data. Both of these purposes are directed at one vision : improving performance on BitTorrent communities. The system is intended to work without the need to change the existing, widely implemented credit system.

This thesis is structured as follows. Chapter 2 discusses the specific problem that we intend to solve. Chapter 3 presents the design of the credit mining system and its requirements. Chapter 4 shows the core investment algorithm we proposed. Implementation of the mechanism and its experiment setup will be elaborated in chapter 5. Chapter 6 shows the performance of the credit mining system. Chapter 7 then concludes the work and mentions possible future work.

Chapter 2

Problem Description

The nature of P2P systems is to help each other achieving a common purpose. In file-sharing P2P systems, the purpose is to obtain some particular content on the Internet. However, in reality, some peers are able to freeride. Although limited, an incentive system can push anyone to contribute. The distant future vision of this work is to create a *European Youtube*-type of system without any central authority server. Similar to Youtube's ease-of-use, our system uses BitTorrent for robust seeding, and without explicit file management or complicated settings. One of the benefits of this research is the existence of an automatic caching layer, which ensures content availability for both long-lived years-old and freshly-created content. The research includes the autonomous mechanism where everyone contributes, and at the same time can gain benefit for themselves. Specifically, by devising a credit mining system, we take an important step to reduce the needs for human intervention in contributing to improve other users' experience.

In this chapter, the problems that are the main concern of this thesis will be elaborated. We will discuss performance problems in BitTorrent system, specifically by looking at its supply and demand misalignment. After that, the *investment* as possible behavior that can tackle this issue will be elaborated. It covers the importance, potential gains, and desired effect of the possible credit investment. After specifying problems, prior works on credit mining will be reviewed. Further improvements on those works are the core of this thesis. Lastly, two research questions on investing in credit system will be formulated.

2.1 Challenges in credit mechanisms

The use of credit in BitTorrent environment must be implemented with utmost care. Rahman et al. showed that credit dynamics in BitTorrent community may lead to system seize-up. Two seize-up statuses that are caused by credit dynamics are *crash* and *crunch*. *Crash* and *crunch* is the condition where there is too much credit and lack of credit, respectively [19, 20]. In order to preserve swarm sustainability, two aspects need to be considered. The first aspect is the swarm

Table 2.1: Supply and demand in public and private communities [9]

community	download speed (kbps)			max s/l	avg s/l	seeding duration (hours)		
	mean	median	top 10%	ratio	ratio	mean	median	top 10%
The Pirate Bay	1037	333	>2134	32	2.6	11.7	1.8	>31.4
EZTV	928	294	>1575	46	6.6	18.1	4.7	>52.0
TVTorrents	3590	1362	>7692	1589	104.5	44.1	17.9	>130.7
TorrentLeech	4937	1030	>7166	*N/A*	25.4	50.4	16.8	>153.9
PolishTracker	8625	1331	>14128	667	63.8	58.0	20.2	>156.0

condition, such as file size and initial credit distribution [20]. Vinkó and Najzer showed that large file size could decrease the swarm sustainability. As for initial credit configuration, a higher credit amount may increase both the throughput and the chance to be crashed. The second aspect is peer behavior [19]. Rahman et al. concluded that selfish peers who only upload just to continue downloading can badly harm the swarm. Ironically, a few high performance individuals can lead to lower community performance [20]. Therefore, it is important to balance both peers and community needs.

Despite having different performance, both public and private communities suffer from a similar issue. "*Poor downloading experience*" is widely known in public communities that have low SLR, which directly affects the swarm performance. On the other hand, private communities suffer from "*poor downloading motivation*" as described by Chen et al.[5] even though the private community was intended to solve the low SLR issue. Both issues are caused by misalignment of supply and demand.

2.1.1 Supply and Demand

Supply and demand for both public and private BitTorrent communities has been intensively studied [9, 21]. Andrade et al. showed that in the BitTorrent community, the supply mostly meets the demand. We define a supply and demand *misalignment* as a condition where supply cannot meet demand without noticeable performance degradation or wasted resources. A significantly high or low seeder/leecher ratio can lead to supply and demand misalignment.

In a public community, there is a significantly lower seeder/leecher ratio compared to a private community which enforces SRE [9, 21]. This lower ratio causes a lack of supply for demand in the swarms. On the other hand, in a private community with SRE, there are consequences for peers who do not seed. This enforcement will result in the community end up with a lot of peers who are actively seeding, or in other words, giving supply. Meulpolder et al. stated that in private communities, *tit-for-tat* is almost irrelevant as nearly all of the data comes from the seeders

[9]. This is not surprising because as shown in Table 2.1, the ratio of seeders to leechers in a private community can reach up to 1589 with the average reaching more than 100. By contrast, in the public community, there are only 2-7 seeders per leecher, and the maximum ratio is under 50 [9]. The download speed of private communities is also 3-8 times faster than in public communities.

In classical file-sharing peer-to-peer system, it is common to see that a swarm is *undersupplied*. Undersupply means that there are not enough resources shared within the swarm to be distributed to the peers who want it. Two possible reasons why this happens are: (i) an asymmetric number of seeders and leechers, for which seeders cannot compensate; and (ii) a lack of incentive mechanism in the higher level aside from BitTorrent *tit-for-tat* [21]. With the introduction of private communities which enforce a policy such as SRE or any incentive mechanism, the problem is shifted to a phenomenon called *oversupply*. The main reason a swarm is *oversupplied* is that swarm has an overwhelming number of seeders. Jia et al. mentioned that an *oversupplied* swarm might result in lower bandwidth allocation for other users [11]. Both undersupply and oversupply are the sub-cases of supply and demand misalignment. The undersupply condition can be solved by simply adding more high-performance peers to seed. On the other hand, the oversupply problem is not as trivial to solve.

In an oversupplied swarm, users may find it difficult to earn credit. This is because of the problem described by Meulpolder known as “upload competition” [4]. Two conditions from the peers’ perspective must be fulfilled to make a P2P system sustainable, which is: peers must be cooperative, and cooperative peers must stay in the swarm as long as possible [4]. The Table 2.1 shows that on average, users in private communities standby for seeding for 50 hours. Jia et al. found out that the common way to survive expulsion is by seeding longer. However, if this behavior happens over a long period, it might produce significant imbalance between supply and demand, as a seeder keeps seeding a particular torrent without switching to another swarm. Moreover, users’ resources may be wasted. Over a long period, the imbalance will gradually degrade user motivation to keep active in the community [5].

Meulpolder in his work illustrated the relation between various P2P system properties and its relation to system balance. The illustration is shown in figure 2.1. In his work, Meulpolder showed that by using naive random seeding behavior, it is not sufficient to make the P2P system balanced [4]. An unbalance system can not only lead to an unsustainable community but also worsen the user download experience. Therefore, it is important to study seeding behavior for each peer to find out how to balance supply and demand in BitTorrent swarms.

2.1.2 Investing as seeding behavior

The activity of spending credit can be divided into three categories: (i) *trading*, (ii) *investing*, and (iii) *gifting* or *donating*. When a user wants to spend their credit to get something, we define it as *trading* or buying. *Gifting* is the case in which a peer

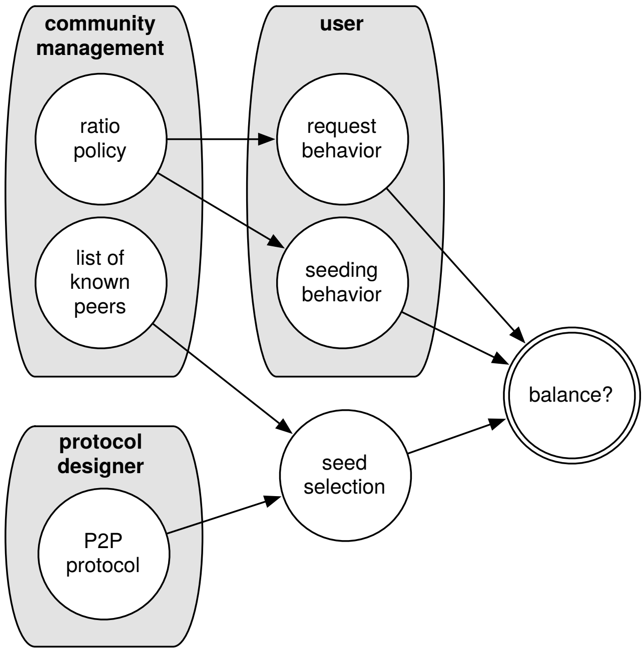


Figure 2.1: System properties and its relation to P2P balance [4].

.

consciously intends to not get any direct, immediate, or obvious return for their spent credits [22]. The purpose of this action is usually to improve the performance of a community, or simply to demonstrate altruism. Investment is the activity of spending credit with the expectation of obtaining credit to use later on.

The ideal situation to balance the performance and sustainability in BitTorrent communities is obtained by aligning supply and demand as discussed in section 2.1.1. Aside from *trading* which is a natural occurence in BitTorrent communities, by *gifting* undersupplied swarms adequately, the optimal situation can be achieved, and the tragedy of the common can be prevented [23]. However, P2P users are typically selfish in an economic perspective [24]. Andrade et al. also shows that users who contribute more to the community, actually consume more from it than lesser contributors [21]. This explains why BitTorrent users are not sufficiently altruistic. Therefore, *investment* as a seeding behavior is the most feasible method to balance user and community needs.

We define the activity of downloading with the expectation of obtaining credit in the future (by uploading) as *investment*. A user can *prospect* which swarm he will invest in irrespective of his resource. The process of identifying the swarm that needs to be seeded is important to balance content availability and personal gain. In a good investing algorithm, users can gain credits and help each other. By providing proper prospecting function, the investment can be more accurate.

In classical economic principle, the key to gain benefit is to buy low and sell high. By finding popular items and suitable swarms, the potential of investment become huge. For example, in the case flashcrowd, an item can become so popular that undersupply might happen. The flashcrowd effect is the sudden increase in demand due to various reasons. For instance, a recently published torrent is one of the cases in which the flashcrowd effect takes place [25]. Investing in the flashcrowd swarms is more beneficial compared to the old, saturated swarms. Furthermore, supply and demand misalignments can be avoided.

2.2 Prior Credit Mining Research

Our work is based on the preliminary work by Capotă et al. from 2013 till 2015 [26, 27, 28]. Based on the prototype they made, a complex method with a speculative download to assess the swarms was implemented [27]. Extending this work, they introduced a *helper* peer to seed low capacity swarms using libtorrent *share mode* [28]. Recently, they moved to a multiple swarm approach and used a public community as their research object. With swarm selection policy, they observed whether helper peers could generate high credit with less downloading [26]. Capotă et al. conducted emulation and simulation in their work.

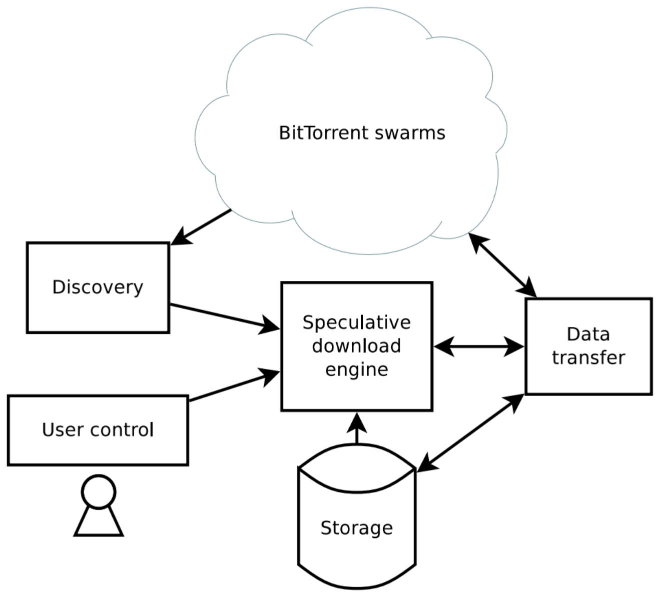


Figure 2.2: Speculative download mechanism [27]

.

In 2013, Capotă et al. introduced bandwidth investing in BitTorrent private communities. They applied speculative download (as shown in figure 2.2) on a prospective swarm. This research used activity data crawled from Bitsoup[1] to evaluate

[1] `https://www.bitsoup.me`

their system. Every swarm is analyzed as to whether the system keeps the swarm in *cache* or discards it. The swarm is scored by predicting future upload speeds defined in the multiple regression model [27]. One of the limitations is that the more swarms that need to be assessed, the less chance there is that the algorithm will find a suitable cache to replace. Another limitation is that *multivariate adaptive regression splines* (MARS) implemented in this system is very costly and complex.

A year later, in 2014, research to align supply and demand in BitTorrent network was conducted. Each peer monitored their swarms to detect potential undersupply. If such a condition is found, a peer broadcasted a *help request* to specialized peers in order to seed that particular swarm. Specialized peers, called *helpers*, try to download as little as possible while uploading as much as possible using *libtorrent* share mode. They implement multiple helpers and observe its effect on the swarm. The result of their experiment shows that using share mode in a closed environment increases download performance by shifting the bottleneck in the swarm if the bandwidth is underutilized [28].

The most recent work was conducted in 2015 [26]. Capotă et al. incorporated their previous work into a Credit Mining System. The credit mining system is able to monitor multiple swarms in one moment, and can then decide to which swarm this system will give its bandwidth. It uses a simpler policy to choose a swarm compared to the multivariate regression model in previous work [27]. The overview of the mining process is shown in figure 2.3. The experiment was conducted in a live fashion on the *etree.org*[2] public community. They observed a *net upload gain* which is defined as a positive difference in uploaded bytes and downloaded bytes. The proposed policy and framework resulted in positive effects to both the community and individuals.

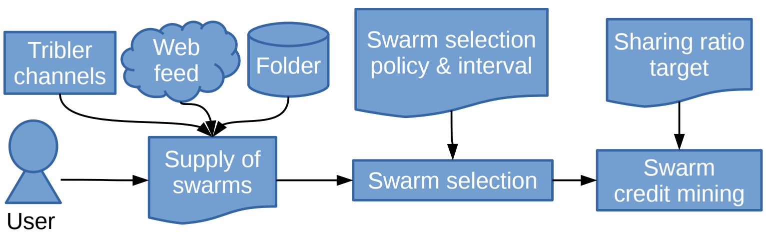


Figure 2.3: The overview of credit mining process [26]

.

2.3 Prospecting good investment

Investing can not be separated from another activity known as "prospecting". We define *prospecting* as the activity of identifying and measuring a swarm in the hope of getting good *return-of-investment*. In practice, not all undersupplied swarms

[2] `http://bt.etree.org/rss/bt_etree_org.rdf`

need to be seeded, even more oversupplied swarms. A swarm can be chosen by considering its seeder-to-leecher ratio, piece availability, resource availability, and many more factors. In a credit based community, on point investment may spark the community thus improving the performance. From the user perspective, a good prospecting algorithm can result in an increased possibility of high return-of-investment.

Getting swarm information is crucial for prospecting. Most researchers have measured BitTorrent swarm by crawling its respective community pages [11, 8, 24, 9, 5, 10, 27, 21, 29]. This way, the researchers can get the data summarized by the pages. Some researchers contact the tracker regularly or use its dump logs [30, 8, 6, 29]. Most of the research that has used logs as its dataset only used a single tracker to monitor a particular torrent. BTWorld[3] has identified four measurement techniques in BitTorrent [31] as shown in table 2.2. As investment needs real-time data, both *swarm-level* and *peer-level* measurement seems to be the most compatible with prospecting method implementation. Both *internet-level* and *community-level* needs compiled data from the ISP company and community administrators, respectively.

Table 2.2: BitTorrent measurement techniques [31]

Level	Advantage	Disadvantage
Internet	Excellent coverage	ISP collaboration
Community	Implementation	Peer details
Swarm	Details	Context
Peer	Details	Scalability

2.3.1 Credit mining as investment tool

The idea of the credit mining system is to help undercapacity swarms, while at the same time getting credit for uploading data. The system needs to find swarms that might have high return in *prospecting*. The *investing*, which relies upon prospecting, is more complex. It has to consider used, limited resources as an additional requirement. The resource can be in several forms, such as bandwidth, memory, or storage. Although the term "good" may be relative, we wish to demonstrate the performance of the system by measuring how much net credit it gained. Therefore, we define the first research question as:

How to prospect swarms and what is a good investment?

In order to answer the question, we formulate technical challenges that need to be solved. The challenges include the engineering and performance evaluation aspect. Both prospecting and seeding processes may disrupt user activity. Therefore,

[3] `http://btworld.nl/`

it is important to take advantage of any unused bandwidth. Many characteristics of a swarm, such as seeder ratio and swarm completeness, need to be considered. The system will need to quantify those characteristics into a value which can be compared and evaluated. The goal is to build an autonomous prospecting and investing system which can yield a high return of investment. In evaluating the system, it is necessary to observe the effect of the credit mining system as a whole.

2.4 Substituting investment cache

In the first question, we have addressed how to gain as much credit as possible in a non-disruptive manner. However, this does not consider the limited resources available at the user's disposal. Investment is a tedious activity if it is done manually. Users are often forced to seed for an excessively long time to maintain adequate credit [11]. By seeding unproductively, the user wastes their effort and resources, such as bandwidth and storage capacity.

Before seeding can be started, the data must be available locally in the *cache*. By having many data, there is a higher chance to seed multiple swarms as well. Eventually, it is necessary to replace obsolete investment. There are several reasons to do so such as gaining less profit, unstable credit, or unreliable swarms. By replacing an old swarm with a new swarm, the credit gained must remain stable or higher than before. Monitoring multiple swarms manually is unproductive. For that reason, it is desirable for us to do this automatically. Therefore, we define the second research question as:

When to delete a downloaded swarm and replace it with a new investment?

The technical challenge that arises from this question covers several aspects. The characteristics of an unproductive swarm need to be defined. With regard to the previous question, it needs to numerically comparable with another swarm. Because replacing a swarm is costly, there must be a precaution in place to help an unproductive swarm to yield more credits. In other words, a preventive mechanism needs to be formulated. If all the methods fail, a swarm that potentially gains higher credit needs to be chosen to replace the obsolete one.

Chapter 3

Credit Mining System Design

The goal of this thesis is to implement a system which can be used to both gain credit and improve swarm performance. In this thesis, we introduce “Credit mining system”, an automatic investment framework on the swarm with multidimensional properties. With the credit mining system, a user can gain credit with limited bandwidth allocation without any intervention necessary. We assume the *credit* is the amount of bytes from the system to the community (and vice versa). From a higher perspective, credit mining system will help to keep a swarm alive by providing integral pieces to the peer who might need it.

Firstly, the dependencies of the credit mining system, which is *libtorrent*'s s*hare mode*, will be elaborated. `Share mode` is a module which can be activated with the intent of helping a swarm, instead of normal content downloading. This module will be explained in detail in section 3.1. After that, the design of the credit mining system is presented. This system consists of several subroutines which will be explained in section 3.2.

3.1 Libtorrent's share mode

BitTorrent is simply a collection of specifications. It is free to be implemented in any language. One of most popular implementations is *libtorrent*. *Libtorrent* is written in `C++` and has `python` binding. *Libtorrent* was started in 2003 by Arvid Norberg, and it implemented most of the BitTorrent specifications. Most of the well-known extensions, such as DHT, PEX, magnet link, multi-tracker, and webseed have also been implemented. *Libtorrent* is widely used by many torrent clients such as Deluge, qBittorrent, Free Download Managers, and many others.

One of the crucial features used in this work is *share mode*[1]. Initial work performed by Capotă et al. also used this feature [26]. Enabling share mode denotes that one is not interested in downloading the file in a swarm, but instead in gaining

[1]Core code of share mode can be found in `https://github.com/arvidn/libtorrent/blob/master/src/torrent.cpp#L9586-L9727`

a higher share ratio. This is done by downloading as little as possible and uploading as much as possible. A swarm downloaded in share mode may never finish as *libtorrent* will only download the pieces of a torrent which satisfy the share mode requirements.

The share mode algorithm works heuristically as it estimates the rarest piece available in the swarm based on the participating peers. The algorithm is presented in Algorithm 1. For clarity, we divide this algorithm in two parts. The first part is to pass all the restrictions. It tries to find missing pieces for each peer (line 4), disconnects some of the seeders because of connection limit (line 8), and reduces the number of missing pieces with twice the number of seeders (line 10). For the last, it is based on the assumption that seeders can upload as fast as the system. Share mode will fail if all missing pieces are expected to be provided by the seeders (line 11), upload ratio could not be reached (line 14), or too many parallel downloads (line 17).

The second part of share mode is to determine the rarest piece. *Libtorrent* counts the number of peers for each piece to find the lowest one. The number of peers on the rarest piece is termed the *rarity* of the piece. Share mode ensures that only the rarest piece available is downloaded (line 22). The routine ends prematurely if there are not enough peers to upload the rarest piece (line 27). Otherwise, it will randomly download the rarest pieces if there is more than one option (line 30).

There are two limitations in this feature that we observed. Firstly, share mode did not check whether a swarm is good enough to perform this operation. It only tried to find popular pieces regardless of the swarm condition. Consequently, there is a possibility that the operation will not go well. For example, it is difficult to gain credit in a saturated swarm as there is not enough demand. Therefore, the user is fully responsible for whether share mode will yield high credit, or if it will waste his resource.

Secondly, the biggest limitation of share mode is the possibility of getting a bottleneck due to its strict policy. In the early stage of joining a swarm, share mode downloaded very few pieces at a time. For example, until the system has downloaded at least 20 pieces, it will only download one piece (5%) at a time in share mode (line 17). It is also necessary to wait for that single piece to be uploaded (line 14) to at least T peers. The combination of line 14 and 17 can result in slower decision making, by which time the rarity of pieces may have changed. If the system is too late to completely receive the piece, or the piece is not uploaded fast enough, this piece may be obsolete by that time as nobody wants it anymore. Therefore, the condition in line 14 may never be satisfied. If the other pieces cannot cover this condition (by uploading to more than T peers), then the operation will be unable to continue. Then, the system will neither download nor upload pieces anymore.

Algorithm 1 Libtorrent share mode algorithm

Require: T as share mode target

———————————— ▷ Part 1

$missing\_piece \leftarrow 0$
for all $p \in connected\_peers$ **do**
    if p is a $leecher\ and\ p$ is not in share_mode **then**
        $missing\_pieces$ += $total\_pieces - pieces(p)$
    end if
end for
if $|connected\_seeders|$ in $connected\_peer > 90\%$ **then**
    disconnect excess seeder
end if
$missing\_pieces$ -= $2 \times |connected\_seeders|$
if $missing\_pieces \leq 0$ **then**
    return
end if
if $num\_downloaded \times T > uploaded$ **then**
    return
end if
if $downloading > 5\% \times num\_downloaded$ **then**
    return
end if

———————————— ▷ Part 2

$rarest\_rarity \leftarrow MAX\_INTEGER$
for all $pc \in pieces()$ **do**
    if pc not in $collected\_piece\ and\ peer\_count(pc) \leq rarest\_rarity$ **then**
        $rarest\_rarity \leftarrow peer\_count(pc)$
        $rare\_piece$.push(pc)
    end if
end for
if $|connected\_peers| - rarest\_rarity < T$ **then**
    return
end if
download $random(rare\_piece)$

3.2 Credit Mining Architecture

The credit mining system is intended to accomplish the assigned task automatically, with minimal user intervention. The way this system is designed is to align supply and demand of chosen swarm. The short term advantage of this approach is to gain credit by minimizing download and maximizing upload. In the long term, this potentially increases the overall performance of other users as well.

The system can be implemented on any torrent client. In Figure 3.1, it shows the compulsory elements and the relation between the *credit mining system* and the *torrent client*. Currently, we assume that every torrent client also tracks how much data a user has downloaded and uploaded in *credit storage*. Naturally, any torrent client must have a so-called *client downloader* module as well. *Libtorrent* library also must exist as part of the dependencies. Another required feature is the ability to discover peers by all methods (DHT, PEX, LSD, and so on). In some cases, the peer discovery function is disabled for security reasons. While disabling any one method should not affect the credit mining system, it will reduce overall prospecting accuracy.

Credit mining system consists of several elements. Those are *credit mining manager*, *miners*, *mining sources*, *settings* object, and *prospecting engine*. The *manager* receives the *settings* from user in the initial phase. To control the mining process, the user can only interact with the elements specified in *settings*. User action is also limited to only adding and removing a *mining source*. Each of the sources will be assigned with a *miner*, and this assigning depends on the type of the source. The *miner* also has sub-elements as part of the system. An in-depth explanation of mining source and *miners* will be discussed in Section 3.2.1. In the *prospecting engine*, prospecting mechanism takes place. Prospecting mechanism as part of the investment methodology will be discussed in Chapter 4.

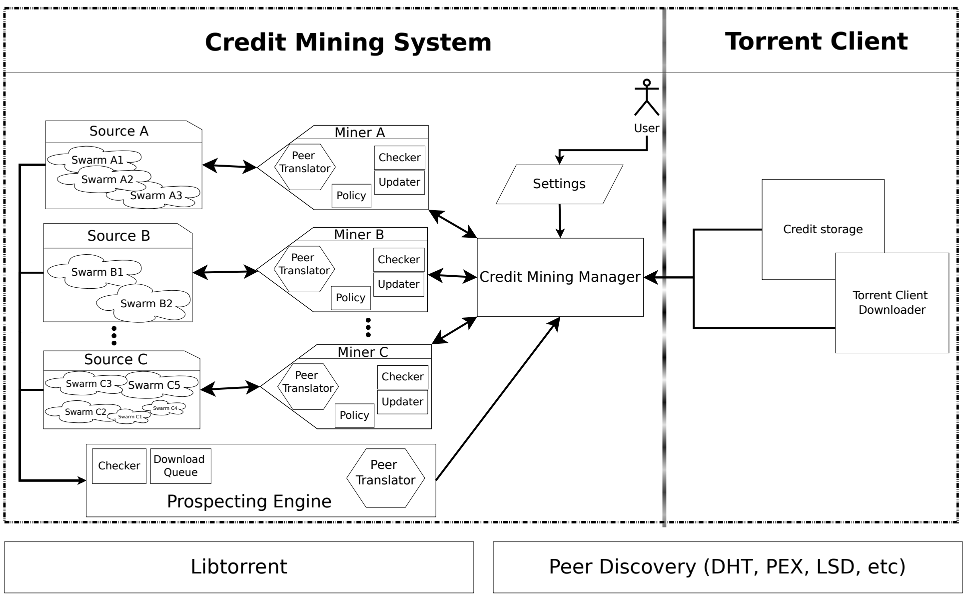


Figure 3.1: Credit mining components.

Before the credit mining system is executed, a user can change the settings used in the credit mining system. For example, if a user has a lot of memory available, it is desirable to mine many swarms at once by changing the *max_torrents_active* parameter. Another example is if a user has a large amount of storage, decreasing *share_mode_target* may yield in a better return. Some settings cannot be changed when the mining process is already running. Table 3.1 shows the settings used in this system.

After the user provides the setting and runs the *credit manager*, the user can start mining by adding sources. A source usually consists of several swarms with different sizes, availability, and capacity. As mentioned previously, each of the sources will be assigned with one *miner*. Before the miners start mining, the *prospecting engine* fetches swarm information and then give the results to the manager .The manager will propagate the swarm information to miners and let the miners decide which swarm to mine based on that information. The manager also monitors the main torrent downloader to adjust the miners' bandwidth allocation. The credit gained from each of the miners will be reported to the manager, which then forwards the results to credit storage in the torrent client if necessary.

Table 3.1: Credit mining settings

No.	Name	Description	Changeable at runtime
1	*max_torrents_active*	The maximum number of simultaneous swarms that will be downloaded	Yes
2	*max_torrents_per_source*	The maximum number of stored torrents in a miner that will be considered for mining	Yes
3	*source_interval*	The interval needed to check for updates in the swarm	Yes
4	*swarm_interval*	The interval to re-evaluate the swarm and start/stop the swarm	Yes
5	*share_mode_target*	Libtorrent share mode target (See Section 3.1)	No
6	*policy*	The policy used in mining (See Chapter 4)	No
7	*tracker_interval*	The interval to check for a new peer by peer discovery methods	No
8	*timeout_torrent_activity*	The maximum time threshold to mark a swarm as 'inactive'	Yes
9	*piece_download*	The number of piece what will be downloaded in the *prospecting* phase	No

3.2.1 Mining Sources

Currently, the credit mining system can accommodate three types of sources. These are: directory source, RSS source, and channel source. The miner will be initiated the moment the source is defined and added to the manager. A miner periodically monitors all of the swarms in the source.

The first type of source is called *directory source*. The system takes and verifies the local directory path containing the `.torrent` file. Each of the files is examined and validated. Any corrupt or invalid file will be discarded and automatically deleted from the disk. The miner then sorts the files alphabetically and puts them into the queue. Both the directory and the queue are periodically monitored by the miner in case of the new incoming swarms. The miner will eventually pop an item from the queue one by one, build a suitable format for mining, and notify the manager to include this swarm.

The next possible mining source is from RSS (Rich Site Summary). RSS is a well-known method to fetch newly published data from the web. An RSS document contains the list of affected content which usually has summarized text and metadata. An RSS from a torrent portal such as etree[2] and mininova[3] usually has the title, publication date and a link to the swarm. RSS also can easily be generated from private trackers for various purposes. We call the source of the RSS document as an *RSS feed*.

In *RSS source*, we assume that the RSS link provided by the user is both available and valid. If by any case the retrieval of the content fails, the miner will stop immediately, notify the manager to disable the said source, and shut itself down. If the initial content retrieval is successful, the update mechanism will be launched periodically to fetch the newest content from the RSS feed. This content is then parsed, resulting in a list of swarm links and its associated information. The miner then asynchronously downloads the swarm metadata either via `.torrent` or magnet link. The same metadata will not be downloaded twice. After fetching the data, the miner will build a defined format for mining, and then notify the manager to include this swarm.

Channel source is the last type of source which is tightly related to the Tribler environment. As mentioned in section 1.2.1 (Table 1.1), a channel is responsible for managing torrents and playlists in the Tribler community. A single channel can be discovered in the *AllChannel* community. A channel is identified by a unique 40-length hexadecimal string. A Tribler user can create their own channel, put torrents into the channel, and share the channel with other users. When a user subscribes to a channel, they will be notified if a new torrent is added to that channel. Moreover, all torrents in the subscribed channel will be automatically downloaded into Tribler's database.

Given the identifier of a channel, the miner will continuously try to find and join the channel in *AllChannel*. By joining the channel, the miner can get a list of torrents and its properties. After a miner joins the channel, the swarm metadata will be downloaded by a mechanism in Tribler. The miner then monitors the local database to determine whether new data has been fetched and whether there is a space for adding a new torrent to the manager. After knowing swarm information, the mining format will be built, and the manager will be notified of a ready swarm.

[2] `http://bt.etree.org/rss/bt_etree_org.rdf`

[3] `http://www.mininova.org/rss.xml`

3.2.2 User activity awareness

Credit mining is an automatic system to download/upload to a swarm. If, at the same time, a user is downloading a swarm outside the one in the credit mining system, the bandwidth will be split. The user may experience a slower download speed and see this as a problem. We define *user download activity* as an activity that is intentionally initiated by the user in order to participate in a particular swarm. Usually, this is the true purpose of having a torrent client.

In response to that issue, we implemented another module in the credit mining system to adjust its mining activity to the user download activity. The credit mining system periodically observes whether there is user downloading activity. If there is none, then it can notify the miners to use all the bandwidth available. Otherwise, the system will limit the download and upload rate of mining activity to the remaining bandwidth available. At the period of observation, the mining download and upload rate will be set to zero.

3.2.3 Resource Optimization

In this section, we focus on several optimizations that can be implemented in the credit mining system. These optimizations are optional and can be omitted. However, as the system itself is not perfect, an improvement to support this system is advantageous. There are two optimizations in the credit mining system: duplicate elimination and swarm blacklisting.

In preliminary work by Capotă et al., credit mining system is able to distinguish duplicate content in P2P communities. It is highly possible that the same file might have a different *infohash* as its identifier. The infohash of a swarm is an SHA1 hash consisting of 40-length hexadecimal string. An *infohash* of a torrent can come from many factors such as different piece size, categorization as private or public swarm, or even the directory name of the files [26]. We used *Levenshtein distance* to measure the difference between one swarm and another by considering the files in it, specifically its names and length. In the end, we only mine the swarms which have a higher number of seeders. Eliminating duplicate swarms can lead to the peer interaction being concentrated in one of the swarms. Thus, the performance of participating peer might be improved.

Despite all of the features in the credit mining system, there is no guarantee that it can constantly gain credit from a particular swarm. The bottleneck in share mode is one such example. We introduce *swarm blacklisting* to remove and block low-performance swarms. The miner periodically watches whether data are constantly being downloaded or uploaded from a particular swarm. If no activity is detected from that swarm for a long period of time, the miner will remove the swarm from its library and block it. That means this particular swarm cannot be chosen by any of the swarm selection policies. It will be added to the library again only after several rounds. If, by any chance, the library is empty and there are swarms blacklisted, those will be added to the library again.

Chapter 4

Investment Algorithm

Investing in a swarm is one of the key elements of the credit mining system. The investing module can be divided into two main stages, namely the *prospecting* and *mining* stage. After new swarm has been discovered, the credit mining system will estimate its return-of-investment potential on the *prospecting stage*. Subsequently, in the *mining stage*, the system will selectively join some of the high-prospected swarms to gain credits. The combination of both stages is the foundation of the whole investing algorithm.

4.1 Prospecting stage

The cardinal problem of knowing a swarm's potential is to obtain reliable swarm size and piece availability information. That information is essential to predict future demand and estimate its return of investment. Given a large number of swarms, joining all of them is costly. Therefore, we introduce the *prospecting algorithm*: a means to download a few pieces to roughly determine peers and piece information. Prospecting cost per swarm is limited, allowing us to estimate and compare the return-on-investment of a few thousand swarms easily.

The pseudocode of the *prospecting* procedure is shown in Algorithm 2. There are n pieces that need to be downloaded. If it succeeds within a certain threshold, then the swarm has potential. Otherwise, it does not have any potential and is then safe to discard.

To get swarm information, there is a piece that needs to be downloaded. Although any piece will do, we choose only the first piece. Picking the first piece can result in a smaller storage allocation. The system will then look for $n - 1$ rarest pieces. By finding the rarest pieces, *prospecting* can filter more swarms. Moreover, this method is consistent with the core functions of BitTorrent. While *prospecting*, the system actively asks for new peers to tracker or DHT, and stores them afterwards. The information collected then will be used in the *mining* stage.

In line 22 in Algorithm 2, the algorithm tries to find at most n rarest pieces. These are obtained by querying peers for their pieces. However, there are several

Algorithm 2 *Prospecting* procedures

```
function PROSPECT_SWARM(infohash, n)
    if |download_queue| > 100 then
        recall PROSPECT_SWARM(infohash, n)
    end if
    PUSH(download_queue, infohash)
    SET_PIECES(infohash, 0, 0)
    UNSET_PIECES(infohash, 1, PIECES(infohash))
    return CHECK_PROSPECT(infohash, n − 1)
end function
function CHECK_PROSPECT(infohash, n)
    peerlist ← GET_PEERS(infohash)
    ADD_TO_PEERLIB(peerlist)
    if wait long enough and not finished yet then
        POP(download_queue, infohash)
        return False
    end if
    if wait long enough and already finished then
        POP(download_queue, infohash)
        return True
    end if
    if PIECE_DOWNLOADED(infohash) = 1 then
        rarest_pieces ← FIND_RARE_PIECE(GET_PEERLIB(), n)
        for all rarest_pieces as p do
            SET_PIECES(infohash, p, p)
        end for
    else if PIECE_DOWNLOADED(infohash) ≥ n then
        mark infohash as finished
    end if
    return CHECK_PROSPECT(infohash, n)
end function
```

cases in which this cannot be found. In the first case, all of the rarest pieces have been owned by all peers. In second case, there is no information regarding peers or pieces yet. In third case, it is possible that the system already has all the rarest pieces. In all these cases, the function may return empty and needs to wait for more peers or piece information.

Peer translation

For the system to be fully decentralized, it is important to not be completely reliant on the tracker. In a case of multi-tracker[1], some swarms may have an entirely

[1]Defined in : `http://www.bittorrent.org/beps/bep_0012.html`.

different number of peers for each of the tracker as shown in figure 4.1. Because of this, we alternate the swarm information source by looking directly at the connected peers. This procedure, by which swarm information from both currently and previously connected peers will be interpreted, is known as *peer translation*.

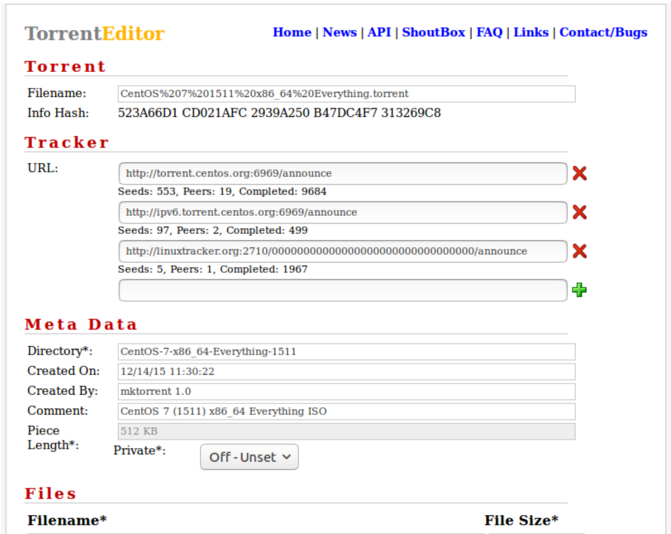


Figure 4.1: Different number of seeders and peers/leechers reported by different trackers.

The *peer translation* receives a list of peers from *libtorrent* and classifies them into seeders and leechers. This procedure is shown in Algorithm 3. This algorithm considers `upload_only` and `progress` flags for each peer. To determine whether the peer is a leecher or not, extra flag (`interested`) is necessary. If the result still zero, the algorithm will consider the downloaded or uploaded amount of that peer from or to the miners. Whichever classification gives the highest number will be picked. Finally, both the projected number of seeders and leechers will be returned to the caller.

4.2 Mining stage

After the prospecting has been done, the mining stage will take place at regular intervals. The mining stage occurs when the credit mining system already has a pool of swarms with potentials. In a fixed interval, the system evaluates the swarms in *swarm selection* algorithm which will be explored in 4.2.1. As a contribution, we proposed a *scoring policy* that will be elaborated more in 4.2.2.

The credit mining system monitors those swarm continuously. The purpose is to look for swarms that are not performing well in a particular time frame. Under certain requirements, this swarm then will be *stimulated* by optimistically downloading few rare pieces at once. The objective of this approach is to eliminate

Algorithm 3 Peer translation algorithm

```
function TRANSLATE_PEER(peer_list)
    num_seeder ← 0
    num_leecher ← 0

    upload_only ← |GET_PEER(UPLOAD_ONLY)|
    finished ← |GET_PEER(PROGRESS > 0.8)|
    unfinish ← |GET_PEER(¬UPLOAD_ONLY & PROGRESS <0.8)|
    interested ← |GET_PEER(INTERESTED)|
    num_seeder ← MAX(upload_only, finished)
    if num_seeder = 0 then
        num_seeder ← number of peer which upload to us > downloaded
    end if
    num_leecher ← MAX(interested, unfinish)
    if num_leecher = 0 then
        num_leecher ← number of peer which download from us > uploaded
    end if
    return num_seeder, num_leecher
end function
```

idleness caused by the bottleneck of share mode. This approach will be elaborated in 4.2.3.

4.2.1 Swarm Selection

Swarm selection is the periodic process of selecting swarms based on their mining potential. At some point, the previously selected swarm may not be beneficial anymore and thus needs to be substituted. The swarm selection process is determined by the policy containing rules to sort swarms by their criteria and potential. Currently, all of the miners need to comply with one defined policy.

Three policies have been defined in the preliminary work [26]. Those are based on *Random policy*, *Swarm Age policy*, and *Seeder Ratio policy*. Specifically for *Seeder Ratio policy*, it is specialized to help undersupplied swarms and gave the best credit gain of the three [26].

4.2.2 Scoring Policy

The scoring policy is a proposed method to quantify a swarm's prospects and to reduce a possible identical result from two swarms or more. It was brought up with the *seeder ratio policy* as its base. It can be customized with its *score multiplier*.

Scoring policy consists of several elements. The first is the *seeder ratio* which is defined as the ratio of seeders to the total number of peers. The second is the *availability* of a swarm. This represents the piece shortage, which shows the potential to gain more credit for the longer term. The third is the number of peers as a tie-breaker. It is useful to target large swarms instead of small ones as there are comparably more options to give the pieces to. The last is the recent swarm activity. Inspired by *tit-for-tat*, the policy will prefer a swarm in which any of its peers had interacted with the miners previously.

Algorithm 4 Scoring policy algorithm

Require: M_leech as leecher multiplier
Require: M_pratio as peer ratio multiplier
Require: M_avail as peer availability multiplier
Require: S_low as extra score for lower activity
Require: S_high as extra score for higher activity

Require: $peerlist$ as the list of stored peers
Require: $swarmlist$ as the list of swarm in the miners

1: **for all** $s \in swarmlist$ **do**
2: $\quad rleech \leftarrow 1-$SEEDER_RATIO(s)
3: $\quad rpeer \leftarrow |$PEERS(s)$|/|peerlist|$
4: $\quad ravail \leftarrow 1-$AVAILABILITY(s)$/|peerlist|$
5: $\quad score[s] \leftarrow M\_leech * rleech + M\_pratio * rpeer + M\_avail * ravail$
6: $\quad total\_speed[s] \leftarrow 0$
7: $\quad$ **for all** $p \in$PEERS(s) **do**
8: $\quad\quad total\_speed[s] \leftarrow total\_speed[s] +$ GET_SPEED(p)
9: $\quad$ **end for**
10: **end for**
11: SORT$(total_speed)$
12: **for all** $s \in swarmlist$ **do**
13: $\quad$ **if** $index(total\_speed, s) < |total\_speed|/2$ **then**
14: $\quad\quad score[s] \leftarrow score[s] + S\_low$
15: $\quad$ **else**
16: $\quad\quad score[s] \leftarrow score[s] + S\_high$
17: $\quad$ **end if**
18: **end for**
19: **return** $score$

Scoring policy, as shown in Algorithm 4, starts with examining all of the swarms registered in a miner. Then it decides the score individually as shown in line 2-5. Variable $rpeer$ is the ratio of the number of peers in this swarm to the total number of peers that is already known from all the swarms. *Availability* is the number of complete copies of a piece plus the fraction of the non-seeder peers that provide a subset of a piece. The availability algorithm is explored in Algorithm 5. If *availability* is 0, this means that there is not any single piece from any of the discovered peers. It will also return 0 in a case where the piece/peer information could not be received. If all of the peers have a complete files, the *availability* will reach its maximum value which is equal to the number of discovered peers. After the individual score is assigned, the activity of each peer on each of the swarms is calculated. The activity, which we assume as the peer's total download speed to the miner, is sorted in ascending order. Then, the first half of the sorted activity is marked as low activity, and given the lower activity score (line 14). Similarly, this happens with the second half of the list, but with higher activity and a higher score (line 16).

Algorithm 5 Swarm availability

Require: $plist$ as list of stored peers

$mbit \leftarrow$ POPULATE_PIECE$(plist)$

$complete_peer \leftarrow |GET_SEEDER()|$

$min\_peer \leftarrow$ MIN$(mbit)$

$more\_piece \leftarrow$ number of piece which has more peer than min_peer

return $complete\_peer + min\_peer + more\_peer/|mbit|$

The scoring policy is designed in such a way so that it can be easily customized based on user preferences. This can be done by providing value to the following constants: M_leech, M_pratio, M_avail, S_low, and S_high. Changing the multiplier will affect its behavior. For example, if a user does not consider the number of peers to be important, he can set the multiplier for M_pratio as 0. Setting other parameters except M_leech as 0 will make scoring policy behave like seeder ratio policy. Similarly, setting other parameters except M_avail as 0 will prioritize swarms with very low availability. This behavior is similar to when it is applied with swarm age policy in the flashcrowd case.

4.2.3 Swarm stimulation

We introduced a method to stimulate the mining activity on idle swarms. It starts by looking at which swarms that are already idle for some amount of time. Those swarms are now suspected to be in the *stale* state. We then download several of the rarest pieces on that swarm. We called these pieces the *stimulant*. The stimulant can be downloaded only if the seeder-to-leecher ratio (SLR) for this swarm is higher than the predefined threshold. This threshold should be lower than `share_mode_target` as in both the credit mining system and *libtorrent*. If the ratio is already too low, miners should wait for a piece to be uploaded first. Then, if it is not possible, this swarm will be blacklisted in the next round and be substituted by another swarm. This process is called *stimulating* the swarm. By optimistically download several rarest pieces simultaneously, we hope to stimulate the mining activity in the long term.

Chapter 5

Credit mining Implementation and Experimental Setup

In the previous chapter, we discussed how the credit mining system was designed. In this chapter, we show how the credit mining system is implemented in Tribler, a python torrent client that was built at the Delft University of Technology. Based on this implementation, we can come up with a suitable experiment design to answer our research question in the previous chapter.

This chapter consists of elaboration on both the implementation and its experiment execution plan. First, in section 5.1, we will describe how the credit mining system is implemented within Tribler. As an open source project, Tribler has guidelines for a new submodule that will be integrated. We comply with those guidelines as we will describe later. To evaluate the system, we introduce *gumby* on section 5.2, the experiment runner developed by the in-house Tribler team. The section 5.3 will follow to explain the actual experiment setup plan. We will elaborate the environment condition and code alteration regarding the experiment that needs to be fulfilled.

5.1 Tribler integration

As a proof of concept, the credit mining system was implemented as a module in Tribler. Tribler was built using python, compatible with version 2.x and 3.x. At the time that credit mining system was implemented in Tribler, Tribler still used the WX as GUI (Graphical User Interface) framework. In the future, Tribler will move its GUI to use Qt starting from version 7.0 onwards. All of those components made Tribler work cross platform (Linux, MacOS, and Windows).

In the prior work, some of the credit mining system code was implemented by Capotă et al. and Egbert Bouman in his Tribler fork[1] instead of the main repository. This made the compatibility and stability between Tribler and the credit mining

[1] `https://github.com/mihaic/tribler/tree/channel_boosting_new_exp`

system break, thus making the system unusable. At this point, the credit mining code was 1528 line long with 51 deletions compared to the main branch.

5.1.1 Contribution on software engineering

As part of the software engineering process, the credit mining code needs to pass several steps before being merged into the main repository. In Tribler, there are two main branches, which are `devel` for all new features and fixes, and `next` which contains bug fixes for the stable release. The first credit mining prototype was directed to `devel` branch as it was a new feature at that point. Before it can be merged, the code must pass the peer review and unit tests on Jenkins[2]. This process is repeated until there is no other feedback. As shown in Figure 5.1, the first credit mining prototype was heavily discussed by 6 other participants and more than 450 comments. It also took almost 3 months to accommodate all of the feedback and reviews. The contribution of this integration is worth more than 4200 added lines and 140 deletions. The code portion is quite balanced with 1425 lines going to the GUI part of the code, 1290 lines to the credit mining system itself, 1160 lines to the tests, and the rest to other Tribler components to accommodate the credit mining system. At the time of merging it had passed the necessary code coverage and allowed number of violations. Therefore, it confirmed that the credit mining system can be deployed in all systems that are supported by Tribler.

Figure 5.1: Merged pull request on credit mining prototype[3].

To ensure the quality of the main branch, any code submitted through a pull request is tested by a unit test mechanism. There are two categories in the credit mining unit tests. The first test checks its basic function such as policies, peer translation, RSS parser, similarity function, and mining configuration, along with its dependencies. It also tests for the unwanted/error case and how the credit mining system will react. The second test is more complex because it emulates the

[2] `http://jenkins.tribler.org/`
[3] Available in : `https://github.com/Tribler/tribler/pull/2064/`

whole credit mining flow for each mining source type. For an RSS source, the test deploys a local server acting as an RSS feeder. As for a *channel* source, the test suite prepares the environment by fabricating both local channel and torrent, inserting torrent metadata into the *channel*, and pushing the created channel to *AllChannelCommunity*.

5.1.2 Graphical user interface revampment

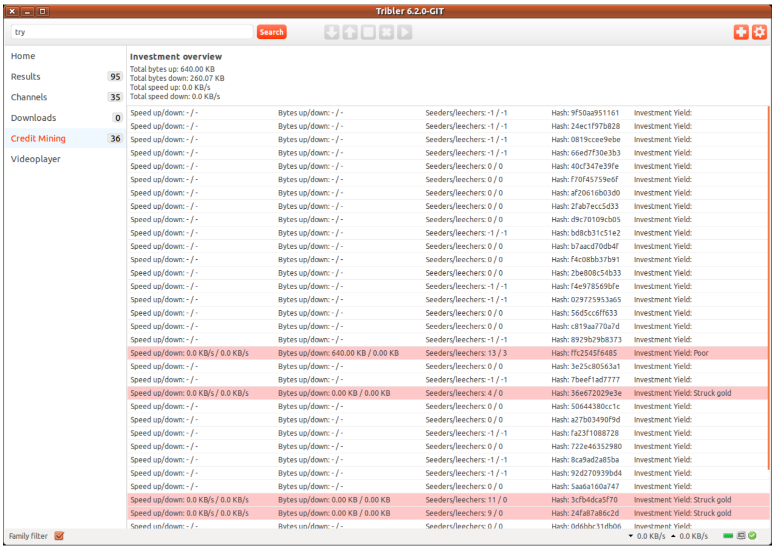


Figure 5.2: The GUI for showing information from prior work [26].

In the prior version, it was not possible to add the mining sources except by changing the Tribler configuration file. There are also several limitations such as incompatible source and instability. Figure 5.2 shows the only interface available from the previous work. As for our work, the credit mining main screen is shown in 5.3. We improved the investment summary by adding more mining source information. In the same window, we also integrated an interface to easily add or remove mining sources. Adding RSS and directory sources can be done by clicking the upper left option. This action will trigger a popup window like shown in Figure 5.3. Adding *channel* as a source can be done by checking the boxes in the channel list.

As an experimental feature, the credit mining system is disabled by default in Tribler. Activating the credit mining module made the Tribler home screen change. We put several channels sorted by their popularity on the home screen, as shown in Figure 5.4. The purpose is to encourage user to altruistically mine. To show the channel information, we provided two details. The first is the popularity, which is shown by the number of stars. The second is a random swarm that resides within a particular channel. A user can simply click on which channel they want to mine,

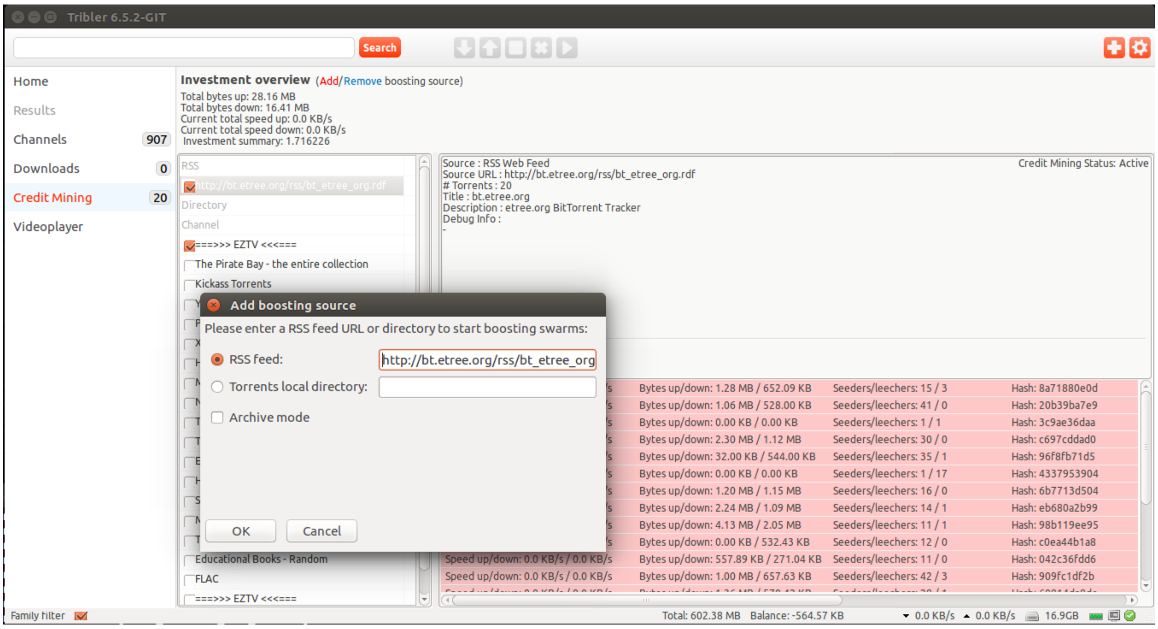


Figure 5.3: Credit mining main window with adding new source.

either on this screen or the credit mining main screen. Both actions will also be reflected on the other screen.

5.2 Gumby

Gumby[4] is an experiment runner framework for both Tribler and Dispersy. Gumby can run on both local and cluster computer to emulate the experiment. Gumby runs on different scenarios, which consists of many commands, for each experiment. It also uses a configuration file to define all of the settings needed for running the experiment. The developer can easily specify the number of peers needed, the post-process script after running the experiment, the value that is needed to be distributed to all of the peers, and many other specifications. The most important part is to code the *client* that is written in `python`. In the *client* file, one must define how the experiment will run and behave, including the commands interpretation.

Gumby runs in a sequential manner with several steps as follows. First, gumby reads the scenario and configuration file. After deciding what type of experiment it has to run, it will clear the output directory. Moreover, in case gumby is running in cluster computer, it also need to synchronize the *client* on multiple nodes. Next, the setup script will be executed. After that, gumby spawns Dispersy and the experiment tracker to monitor the nodes in case of an error occurring. All of the experiment nodes communicate with the server using a specified IP address and

[4] `https://github.com/Tribler/gumby`

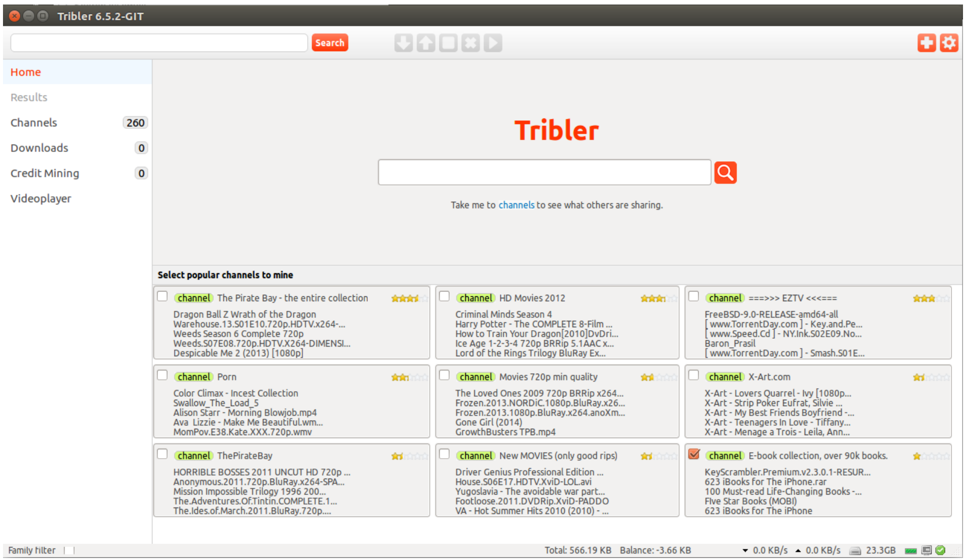


Figure 5.4: The home interface of Tribler with credit mining active.

port. Finally, both local and remote processes are started in parallel. Upon finishing the experiment, the server will wait for all node instances to exit and disconnect gracefully. Then it will copy the data to a predefined directory. Those data now can be processed using specified post-experiment script to generate items such as graphs and tables.

5.2.1 Scenario and Configuration

In gumby, it is not possible to intervene in the experiment on the fly. What the developer can do is specify commands in the scenario file. The scenario notation consists of the time of an action and the command itself. The specific node that needs to run the command can also be specified with curly brackets. Figure 5.5 shows the example of the gumby scenario. For example, command `@0:36 set_boost_settings boosting.ini.1 {3}` means at second 36, gumby will run command `set_boost_settings` with `boosting.ini.1` as a parameter on node number 3.

In contrast to the scenario file, the gumby configuration file only contains variables that need to be filled. These variables can be accessed from inside the *client*. Figure 5.6 shows an example of configuration format in gumby. There are some necessary variables such as the experiment name and `tracker_cmd`. Some of the variables are required by specific conditions. For example, if variable `local_instance_cmd` is `'das4_reserve_and_run.sh'`, it is necessary to call the other 4 sub-variables that are recognized by `das4_` precedence. There are also variables that are completely optional. In this case, specifying variable `scenario_file` is optional to point which scenario gumby need to run.

```
@0:0 set_master_member 3081a73010...e75
@0:2 start_dispersy {1-3}
@0:10 start_session
@0:22 online
@0:23 set_speed 0 0 {3}
@0:32 create {1}
@0:35 publish file1gb_1 1524288077 {1}
@0:36 set_boost_settings boosting.ini.1 {3}
@0:37 start_boosting {3}
@0:40 add_source http://bt.etree.org/rss/bt_etree_org.rdf {3}
@1:15 start_download file1gb_1 {2}
@1:33 reset_dispersy_statistics
@0:43100 stop
```

Figure 5.5: Scenario format example

```
experiment_name = "CreditRunner_base_DAS"
experiment_server_cmd = 'experiment_server.py'
local_setup_cmd = 'das4_setup.sh'
local_instance_cmd = 'das4_reserve_and_run.sh'
output_dir = '/var/scratch/aputra/cmining'
das4_node_amount = 2
das4_node_timeout = 3600
das4_instances_to_run = 5
das4_node_command = "creditmining.py"
tracker_cmd = 'run_tracker.sh'
use_local_venv = True
scenario_file = "creditmining_base.scenario"
post_process_cmd = "gumby/scripts/post_credit_mining.sh"
```

Figure 5.6: Configuration format example

5.3 Experimental setup

We now focus on what setup the credit mining system will be evaluated. In general, there are two aspects we want to address. The first one is how the credit mining system can gain benefit for its user. This means that a user is expected to get a considerable amount of credit with a relatively small investment. The second aspect is to find out how the credit mining system can benefit the swarms as a whole. This can be done by monitoring the performance of each of the peers. If each of the peers' performance is increasing, then the swarm itself has its capacity increased as well.

5.3.1 Experiment conditioning

The experiments were conducted in different scenarios and architectures. We handcrafted the environment needed for all of the experiments. In this thesis, all of the scenarios are presented in the appendix. Most of the experiments are conducted in a closed environment using *channel* as dissemination method. A swarm with fabricated files as content is created. This swarm is then inserted into a particular *Channel*. This *channel* can be accessed from all the nodes using Dispersy. In the end, the user can get the metadata of this swarm such as files list, infohash, and other information typically found in the `.torrent` file. To be able to compare the system performance to prior work, we used etree.org (`http://bt.etree.org/rss/bt_etree_org.rdf`) as a mining source. This is because the prior work's system is not compatible with gumby. Etree.org is a legal community that shares music with permission from authors. This community is relatively active, and newly published swarms usually have sufficient supply and demand for testing.

We have two different sites to accommodate our experiment. First is DAS-4[5] (The Distributed ASCI Supercomputer 4) cluster which runs the CentOS Linux operating system. DAS-4 nodes have a dual quad-core processor with 24 GB memory. The interconnection speed between nodes is 1Gbit/s. The DAS site is used to run experiments that need many peers in a closed environment. It has *libtorrent* version 1.1.1 installed. The second site is the local computer named DUTIJC running Arch Linux with *libtorrent* version 1.0.10. This site has 6 GB memory and quad-core *i7-920* processor. The DUTIJC site is used for running long experiments.

In our controlled environment, a node can be categorized as publisher, seeder, downloader, or credit miner. A single node will act as a *publisher* of this swarm. It creates a *channel* and fabricated files, generates metadata, pushes it into the *channel*, and seeds for the rest of the experiment. Another node can help become a *seeder* for the swarm if necessary. For other nodes, it can be either download or can activate the credit mining system. This *channel* can be added to the credit mining system as a mining source. As for *downloaders*, they can both start and stop downloading from a swarm identified by its name.

5.3.2 Code modification for experiments

In this section, we want to focus on the assumption and code modification for the experiment in a closed environment. As we limit the download and upload rate, we assume the system knows this limit. This makes, for example, finding leftover bandwidth trivial. We also defined the multiplier in scoring policy. The value of M_leech, M_pratio, M_avail are 5, 3, and 4, respectively. The reason behind this number is as follows. We intend to make all multipliers relatively equal and small. However, it is important to distinguish the features of the policy. The difference between multipliers should not be so significant, for example as twice as much

[5] `http://www.cs.vu.nl/das4`

as another. M_leech and M_avail show the performance shortage in swarms. M_leech is used in previous work, so it has a bigger multiplier than M_avail. M_pratio is a tie-breaker, thus assigned the smallest multiplier.

We then altered the code on three occasions. Firstly, the system will aggressively connect to each other in a closed environment. The IP address and port for each node are determined prior to launch. We use this information to build full mesh connection topology. Secondly, any peer information outside the predetermined range is rejected. The third alteration is only applied in the *prospecting* experiment. In this experiment, we increase the maximum swarm per source to one hundred, and set the number of active swarm to zero. Moreover, after a swarm has been *prospected* in this experiment, instead of sending it to miners, we retrieve the information and then delete it afterward. By this approach, the swarm per source slot will be freed faster, and the experiment results will still be valid.

Torrent crawler

For the *prospecting* experiment to succeed, a large number of swarms are needed. Although many swarms can be retrieved from anywhere including illegal sources, we want to contribute to the society by providing support for the legal ones. We implemented a legal torrent crawler that can be accessed at `https://github.com/ardhipoetra/legal-torrent-crawler`. It uses *scrapy*[6] as a scraper for the torrent portal sites. The crawler will access these sites, find any link to `.torrent` file, then download and categorize it. So far, we have implemented the crawler for 8 sites as shown in Table 5.1. The crawler is completely unrelated from the credit mining system. It can be executed independently. The crawler is executed before the *prospecting* experiment is started. The output of this crawler is a collection of `.torrent` files in a single directory which act as the input for the prospecting experiment.

Table 5.1: Legal torrent source.

Source	Description
`etree.org`	Live music trading community.
`legittorrents.info`	Self-moderated torrent tracker and portal.
`librivox.org`	Public domain audiobooks read by volunteers.
`linuxtracker.org`	Linux distro torrent aggregator.
`distrowatch.com`	Linux distro torrent aggregator.
`mininova.org`	Torrent directory site. Used to host copyrighted material but now is no more.
`sxswtorrent.com`	Sample music sharing on SXSW events.
`vodo.net`	Media distributor. Offers legal films, books, and music.

[6]`https://scrapy.org/`

Chapter 6

Performance Evaluation

This chapter will focus on performance evaluation of the credit mining system implementation in Tribler with both synthetic and real-world swarms. We start with simple and easy to understand experiments with predictable outcomes to validate the correctness of our work. We then increase the complexity of our experiments in several steps towards evaluation within the real Internet environment. We will cover both the core components and proposed optimization. All the experiments in this chapter comply with the specifications mentioned in Section 5.3.

6.1 Evaluation metrics

Throughout the experiments, we used several metrics to refer the credit mining system evaluation. In order to measure how many credits the user has already gained, *Net upload gain*[26] is used. Net upload gain is defined as the difference between uploaded and downloaded bytes. To show how efficient a miner can get the credit after putting in the investment, *upload ratio* is also used. Upload ratio is the ratio between uploaded and downloaded bytes.

In order to measure whether the miner consumes resources efficiently, both maximum upload and download rate are considered. In most cases, maximum download rate and upload rate is 250 kB/s and 100 kB/s, respectively. We also combine this metric with how frequent a resource is used. For example, a miner that consumes 80% of the maximum upload rate for 70% of its lifetime is using the resource more efficiently than another miner that consumes the same amount for only 50% of its lifetime. A higher number of these metrics means that fewer resources are wasted.

6.2 Validating the credit mining system

The following experiments are specifically designed to be simple, and to be able to validate all the core components and algorithms of our credit mining research.

All conditions are controlled and do not rely on external elements such as trackers or DHT. Our validation experiments test the basic swarm selection algorithm. The experiments will be conducted for one hour with a 5 minute swarm selection interval.

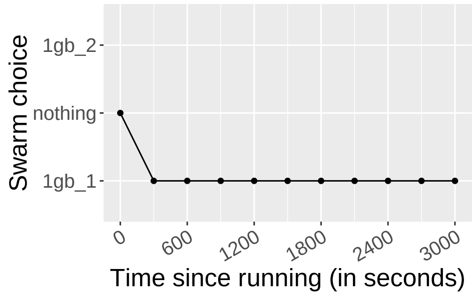
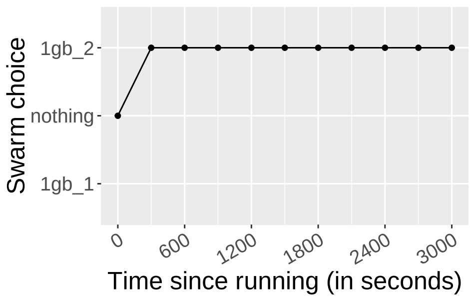
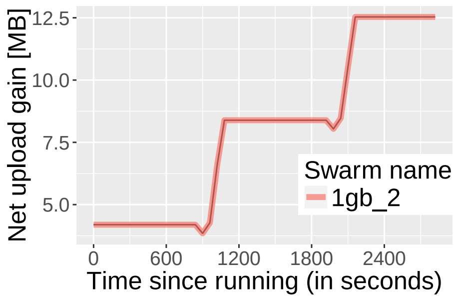


Figure 6.1: Swarm selection in first experiment.

In the first experiment, there are only two swarms: `1gb_1` and `1gb_2`. Swarm `1gb_1` has one seeder, while `1gb_2` has two. Neither swarm has any downloader. Swarms are distributed via *channel* mechanism in Tribler. The miner is then subscribed to this channel and gets notified as to whether the swarms are added to the library. We expect the miner to choose the swarm with the lowest number of seeders (`1gb_1`). Figure 6.1 shows the miner's swarm selection. The result is as expected. Although the miner has already discovered the swarms in 10 seconds after it started the system, it can only start mining in the next 5-minute period. Therefore, in the first mining round, the miner could not choose anything because of lack of information.



(a) Swarm selection in second experiment.



(b) Credit gained in second experiment.

Figure 6.2: Results for second experiment.

We intend to run our second experiment with a minimum number of downloaders. Both swarms now have one seeder. The number of downloaders is one for swarm `1gb_1` and two for `1gb_2`. We expect the miner to select the swarm with the higher number of downloaders (`1gb_2`). In the meantime, the gained credit and upload ratio will also be observed. The `share_mode_target` is set as one.

Again, the result fits with our expectation. As shown in Figure 6.2a, the miner correctly chooses undersupplied swarm (`1gb_2`). Again, it starts mining after the second selection round. As for credit gain, as shown in Figure 6.2b, the miner get 12.5 MB with 1.99 as upload ratio. This upload ratio result is exceeding our target as specified in `share_mode_target`.

6.3 Prospecting hit experiment

The key in prospecting is the ability to find a swarm that is likely to give a high investment return. In the following section, we will asses how well our prospecting algorithm can discard low-potential swarms on the Internet. Furthermore, we will also measure how fast and accurate the prospecting algorithm can find high-potential swarms.

6.3.1 Filtering swarms on the Internet

In the following experiment, we observe how the prospecting algorithm filters swarms on the Internet. The program uses a directory as a source. The `.torrent` files in the directory are collected by the crawler presented in Section 5.3.2. The result will then be compared to *random* and *sequential* methods when fetching the rest of the pieces. Both methods are expected to filter less swarms than our method.

In this experiment, 1 swarm is inserted for every 7 seconds until the maximum amount of active swarms is reached. The number of maximum attempts to find the rarest pieces is 60 times with 30 second intervals. The number of pieces that need to be downloaded is 4 pieces in a 1 hour threshold. Swarm with small content size will be automatically discarded. We divide the failing result into four categories: *timeout, zero peers, no information*, and *no leecher*.

Timeout is the condition in which the threshold is reached, and the system could not finish the prospecting. *Zero peers* happens when the system could not get any peers' information. *No information* means that there is no piece information from known peers. In *No leecher*, we cannot find prospective downloaders, which made investment impractical.

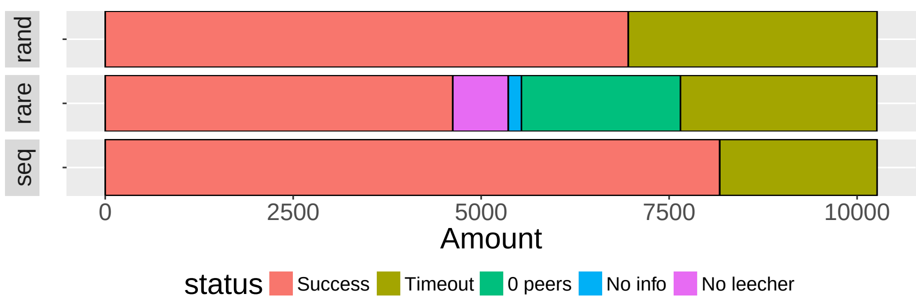


Figure 6.3: Prospecting success percentage with three methods.

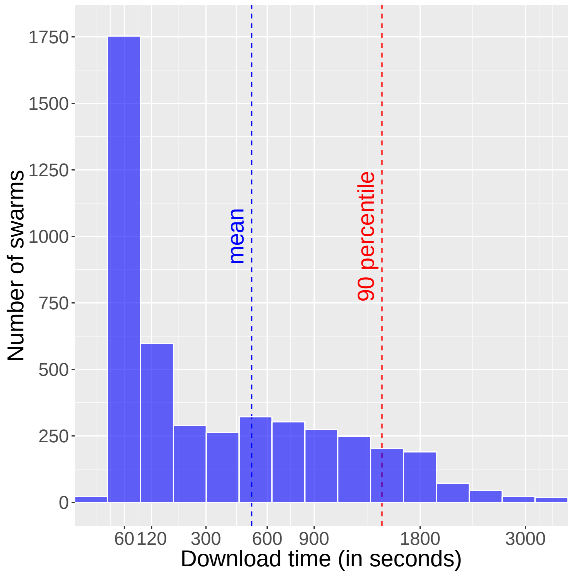


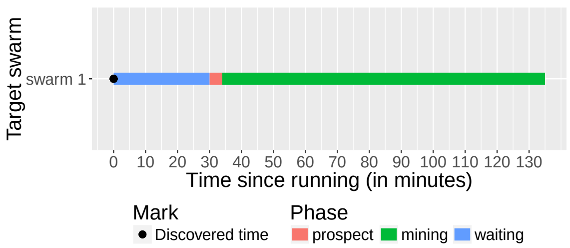
Figure 6.4: Prospecting distributed time

In the Figure 6.3, the portion of swarm that has been successfully *prospected* is shown. Out of 13956 observed swarms, 3690 swarms have a very small file size. This left us with 10266 swarms with 45% finished, 25% timed out, and 20% with zero peers. Figure 6.4 shows the distribution of time needed to download 4 of the rarest pieces on successfully prospected swarms (4623 swarms). Most of the swarms can be prospected in less than 2 minutes. The average time is 514 seconds, and the 90% percentile relies in 1440 seconds as shown in vertical blue and red line, respectively. By looking at this figure, the ideal threshold time should be around 30 minutes instead of 60 minutes. We arrived at this conclusion since, in less than 30 minutes, 90% of the successfully *prospected* swarm is finished.

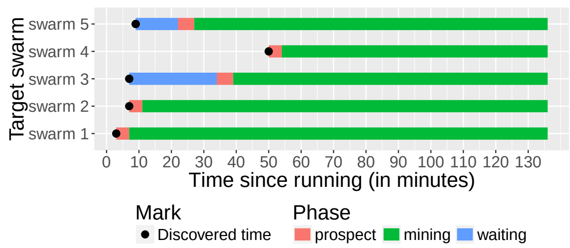
Now, we compare the result to both the *random* and *sequential* method. Figure 6.3 shows that the number of swarms that are successfully prospected has increased significantly in both methods. The top and bottom part of the figure represents the random and sequential method, respectively. Both approaches are resulting in 0 for attempted failure swarm. This fact clarifies that most of the swarms in this experiment are alive. However, some of them do not have any downloaders at the time of the experiment, thus making them inactive. The proposed method can safely discard those swarms as they are not suitable for investing. By discarding 54% of the swarms, it filters 70% and 160% more than the random and sequential method, respectively. Moreover, its behavior is compatible with BitTorrent piece policy. Just after prospecting is finished, the upload ratio may be very high because the pieces we collected are prioritized to be uploaded as shown in the previous experiment.

6.3.2 Finding undersupplied swarms

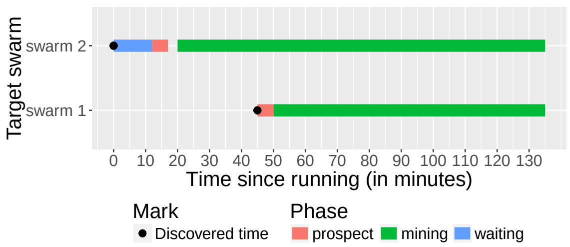
In the following section, the speed and accuracy of the prospecting algorithm to find undersupplied swarms will be evaluated. The experiments are conducted in a closed environment and have a single community containing many swarms. Each swarm has 2 seeders and 1 downloader, except for a few swarms, which only have 1 seeder. We denote the number of undersupplied swarms as a percentage of all the swarms in the community. The maximum number of concurrent swarm for active prospecting is 30. The maximum number of active swarms for mining is the same as the number of expected undersupplied swarms. The experiments only consider seeder and leecher ratio in the *scoring* policy. Figure 6.5 shows the time results with various portion of undersupplied swarms and community size. The black dot is the moment when the miner discovers the swarm. The blue, red, and green lines are the elapsed time for the miner when waiting for prospecting, prospecting, and mining, respectively.



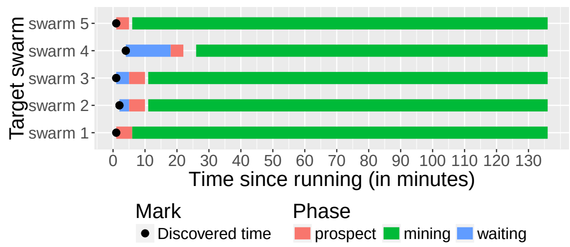
(a) Investing timeline for 1% of 100 swarm.



(b) Investing timeline for 5% of 100 swarm.



(c) Investing timeline for 1% of 200 swarm.



(d) Investing timeline for 2.5% of 200 swarm.

Figure 6.5: Investing timeline results.

The first experiment shown in Figure 6.5a has 1 undersupplied swarm (1%) out of 100 swarms. The miner needs 34 minutes before it can start mining this swarm, mainly caused by the waiting phase (88%). The prospecting results an accurate swarm choice because in the *mining* phase, the miner continuously selects the targeted swarm until its downloader has finished downloading. In the next experiment, the number of undersupplied swarms is increased to 5% and its result is shown in Figure 6.5b. Likewise, the prospecting accuracy is equally accurate. However, swarm waiting time and late discovery hold back the miner from mining all of the targeted swarms. The average time of waiting and prospecting are 20 and 4.4 minutes, respectively. Moreover, `swarm4` was discovered very late on the 50th minute.

Next, we changed the community size to 200. Figure 6.5c and Figure 6.5d shows the result for 1% (2 swarms) and 5 swarms (2.5%) for an undersupplied swarm, respectively. For 5 underseeded swarms, the average prospecting time and waiting are 4.6 and 7 minutes, respectively. The *mining* phase that starts regularly every 5 minutes causes the gap between the *prospecting* and *mining* phase. The accuracy of this experiment is similar to the previous one. There is also a swarm that was discovered very late which holds the miner to mine all of the underseeded swarms.

From these results, we can draw several conclusions. First, provided sufficient information, the prospecting algorithm is both fast and accurate. It only takes around 4-5 minutes which is much lower compared to both waiting and mining time. In general, its accuracy makes the *mining* phase does not need to switch to other swarms. Second, waiting time and late discovery are the main causes that restrict the miner from starting to mine all of the undersupplied swarms as early as possible. Waiting time may be reduced by increasing the maximum number of concurrent prospecting swarms or by changing the prospecting queue system. Late discovery time is introduced by an external factor, which in this case, is the swarm dissemination method by *channel* in Tribler.

6.4 Evaluating Scoring policy

In the following experiments, the *mining* stage as part of our investing algorithm is evaluated. We focus on the *scoring* policy that is used in swarm selection. We wish to show that this policy selects the undersupplied swarms correctly. Furthermore, the mining performance from this selection will be evaluated. We will confirm that our policy results in a positive upload gain and ratio.

The experiments were run for three hours to compare *scoring* and *seederratio* policy. There are 10 swarms, and each has the same content size. Each swarm has a various number of seeders and a fixed number of 5 downloaders. A single credit miner should select the underseeded swarms without relying on tracker or DHT. The credit mining system actively chooses at most 3 of these swarms to mine at a time. We set `share_mode_target` as 2. In the experiment, we use the *peer translation* function to interpret the number of seeders and leechers in a swarm.

6.4.1 The selected swarms

In this section, we focus on what swarm the policy chooses. Two communities are presented. One is the community where there are a different number of seeders and leechers for all the swarms, and with equal content size. Another community is the opposite, i.e. different content sizes with equal peers. *Scoring* policy should perform equally with *seederratio* policy in the first community. For the second community, the *scoring* policy is expected to outperform *seederratio* policy. We will specifically focus on the top three swarms that need to be seeded.

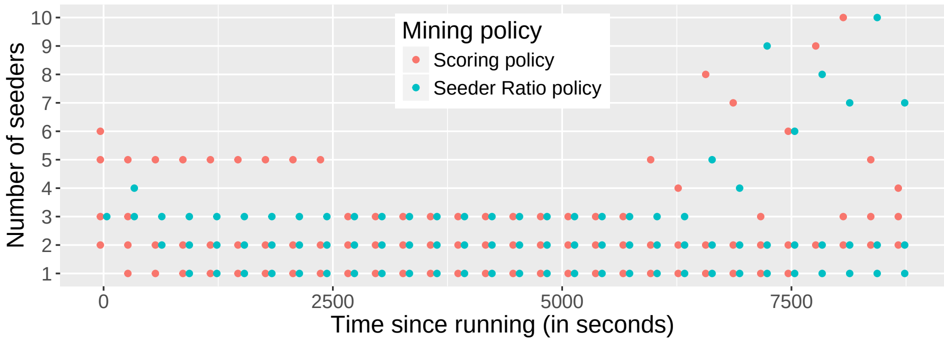
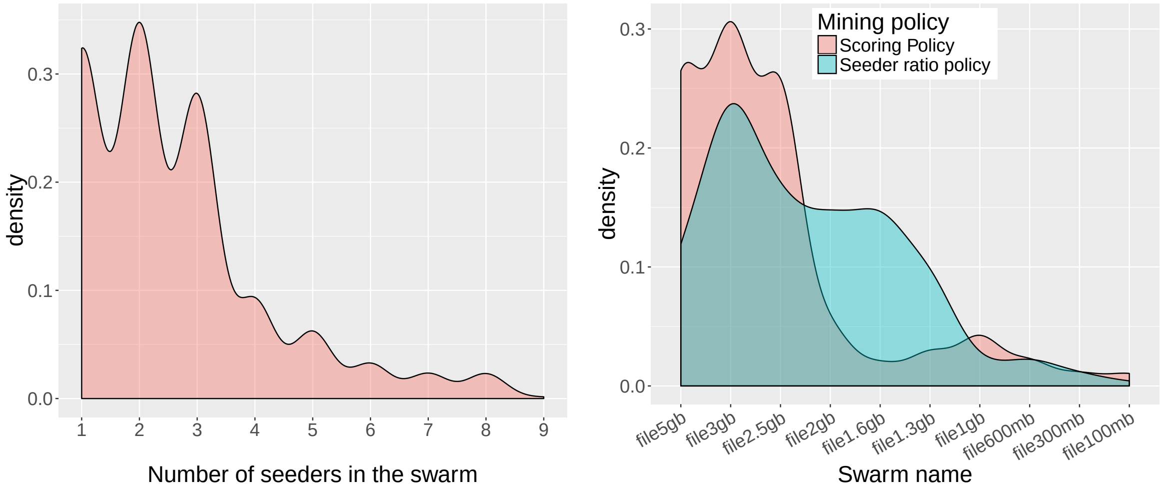


Figure 6.6: Seeder ratio and scoring policy swarm selection.

Regarding the first community, Figure 6.6 shows which swarms the policies select. Focusing on the three targeted swarms, they are chosen for 79% and 89% for scoring policy and seeder ratio policy, respectively. At the beginning of the experiment (timestep 0-2500), the lack of information causes peer translation function to be less accurate. However, as the time goes on, more information can be collected, swarm information is stabilized, and both policy and peer translation function become more accurate. When the current swarm is saturated, the effect is inverted. The information of other swarms become very outdated and causes both policy and peer translation function to have inaccurate results. The system has to rejoin other swarms to fetch the latest information. That explains the policy choice dispersion in the last 40 minutes (2500 seconds) near the end.

The policy is still valid when there is more than one miner in the community. To support this argument, we conducted a similar experiment with 10 credit miners with the scoring policy. Shown in Figure 6.7a, the miners still choose the first three lowest-seeded swarms for the 80% of the experiment.

Figure 6.7b shows what the policies choose in the community with different file size among the swarms. For each policy, we conducted 3 experiments with only a single miner each. From the result, the scoring policy mainly chooses swarms with a large file. On the contrary, in seeder ratio, it frequently chooses other swarms as well. A swarm that has large files will have slower completion rate on its peers compared to one that has small files. Slower completion rate also means there are more rare pieces in a swarm, hence low piece availability. In this experiment, the target swarms are `file5gb`, `file3gb`, and `file2.5gb`. Scoring policy correctly detects the piece shortage problem and addresses it by choosing the swarm with the lowest piece availability. From the three targeted swarms, scoring policy chooses 82%. On the other hand, the seeder ratio sees all the swarm as equal, and the behavior is unpredictable. It chooses 67% of the three targeted swarms.



(a) Scoring ratio swarm selection with 10 miners.

(b) Swarm selection on swarms with different content size.

Figure 6.7: Swarm selection result on extended experiment.

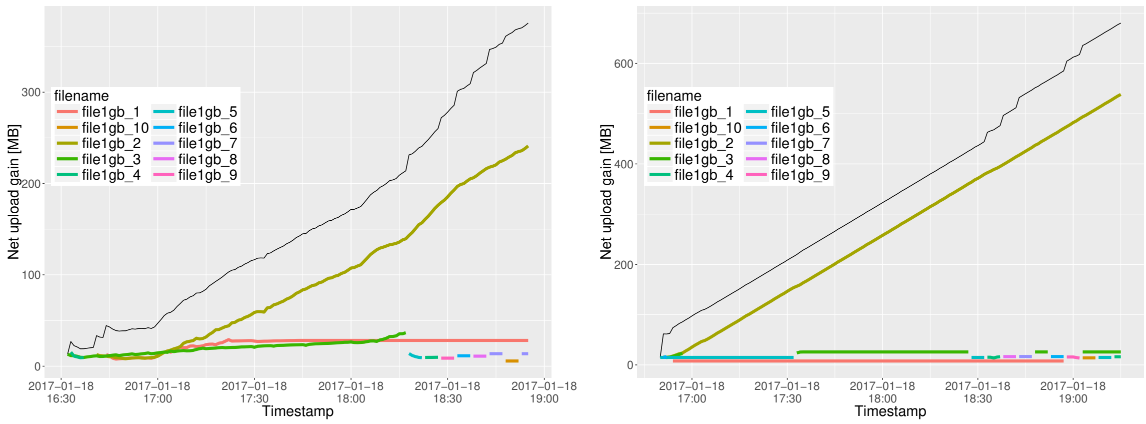
6.4.2 Obtained gain by the selection

The following results are presented to determine the actual gain obtained as a result of swarm selection presented in the first community of Section 6.4.1. Furthermore, we intend to determine the swarm stimulation effect on credit gain. We expect that the credit gain can possibly be increased.

The obtained gain of applying the seeder ratio policy is shown in Figure 6.8a. A swarm with 2 seeders (`file1gb_2`) is dominating the result. The rest of the selected swarms are relatively constant most of the time. Swarm `file1gb_2` average seeding speed is 61 kB/s, which is more than half of the maximum speed on a single peer. This swarm also used a significant resource, which is more than 80% of the maximum upload rate, for 44.67% of its lifetime. At the end of the experiment, it reaches 241 MB gain and 2.005 upload ratio. As a comparison, the average upload ratio is 2.650.

In Figure 6.8b, the scoring policy is applied. Unsurprisingly, the trend is similar. This time, the gain is 538 MB, almost twice that of the seeder policy with the same swarm. Although `file1gb_2` returned the highest gain, its upload ratio is not the highest at only 2.99. The highest ratio is returned by `file1gb_7` at 4.91, and the average from all the swarms is 3.718. Average upload speed for this swarm is 94.4 kB/s with 90% of the observation taking more than 80% of the maximum upload rate. By these results, it is clear that the factor that limits the credit mining system obtaining higher gain is the maximum upload rate.

After we observed those, we come to three conclusions. First, our hypothesis about the similar choice in this particular experiment on both of the policies is proved to be correct. Although the gain is significantly different, it was not directly



(a) Seeder ratio policy gain.

(b) Scoring policy gain.

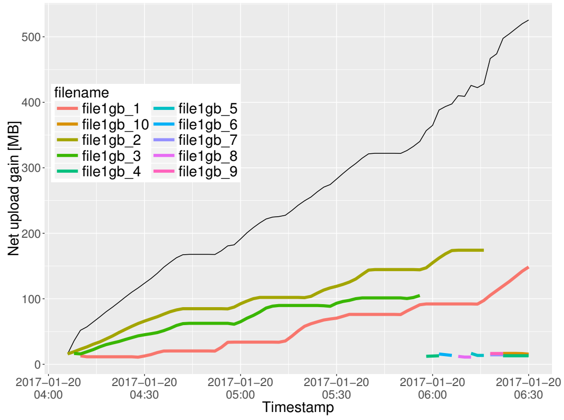
Figure 6.8: Net upload gain of both policies.

caused by the mining system. Instead, it is part of the BitTorrent protocol, one that builds up the download and upload speed. Second, although the resource may be used to its full capacity as shown in Figure 6.8b, it is entirely possible that it is not used efficiently. This is shown by the inactivity from the other swarms. Net upload gain for other swarms is barely increased in both cases. This locked-up condition is caused by the *libtorrent*'s *share mode* algorithm and its cause and possible solutions already discussed in Section 3.1. Third, the swarm that gets the highest net upload gain does not necessarily have the highest ratio. We believe that this is caused by the prospecting mechanism that downloads only a few of the rarest pieces and then uploads those pieces to most of the peers. After downloading these rarest pieces, some of the swarms are not chosen by policy. Therefore, the ratio remains high because there is no downloading activity.

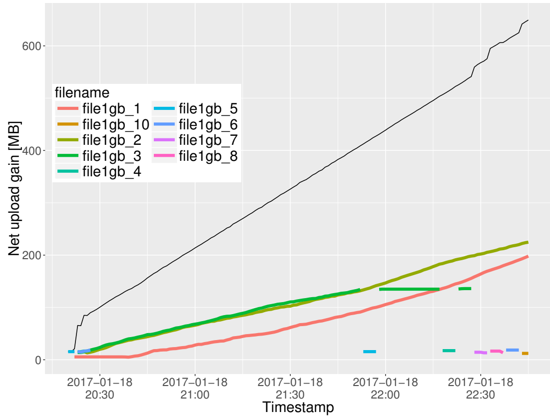
The effect of stimulating swarms

With swarm stimulation, inactive swarms tend to gain more credits when mined. Figure 6.9a shows the case for seeder ratio policy and Figure 6.9b for scoring policy. We specifically focus on swarm `file1gb_1` and `file1gb_3` because they are the most affected swarms by stimulation mechanism. Table 6.1 shows the stimulation effect of those swarms.

Compared to previous results, inactive swarms have their performance increased. From the figure, we can see many bumps that lead to the increasing amount of gain. The total stimulant of each swarm is 20, 12, and 12 for `file1gb_1`, `file1gb_2`, and `file1gb_3`, respectively. Also, both upload ratio and net upload gain are significantly increased. This result is in line with our expectations. On the contrary, in scoring policy, the effect of stimulation is not significant. Although upload gain is increased, the ratio is similar to that of when stimulation was disabled. This happened because in the previous experiment with scoring policy, most of the re-



(a) Seeder ratio policy gain with stimulation enabled.



(b) Scoring policy gain with stimulation enabled.

Figure 6.9: Net upload gain of both policies with stimulation enabled.

Table 6.1: Stimulation effect on gained credit.

Swarm name	**Policy**	**Upload ratio**		**Net upload gain (in MB)**	
		St. Enabled	**St. Disabled**	**St. Enabled**	**St. Disabled**
`file1gb_1`	Seeder ratio	3.37	1.95	148	28
`file1gb_3`	Seeder ratio	2.88	1.95	105	26
`file1gb_1`	Scoring	3.07	2.84	198	77
`file1gb_3`	Scoring	3.00	3.21	134	25

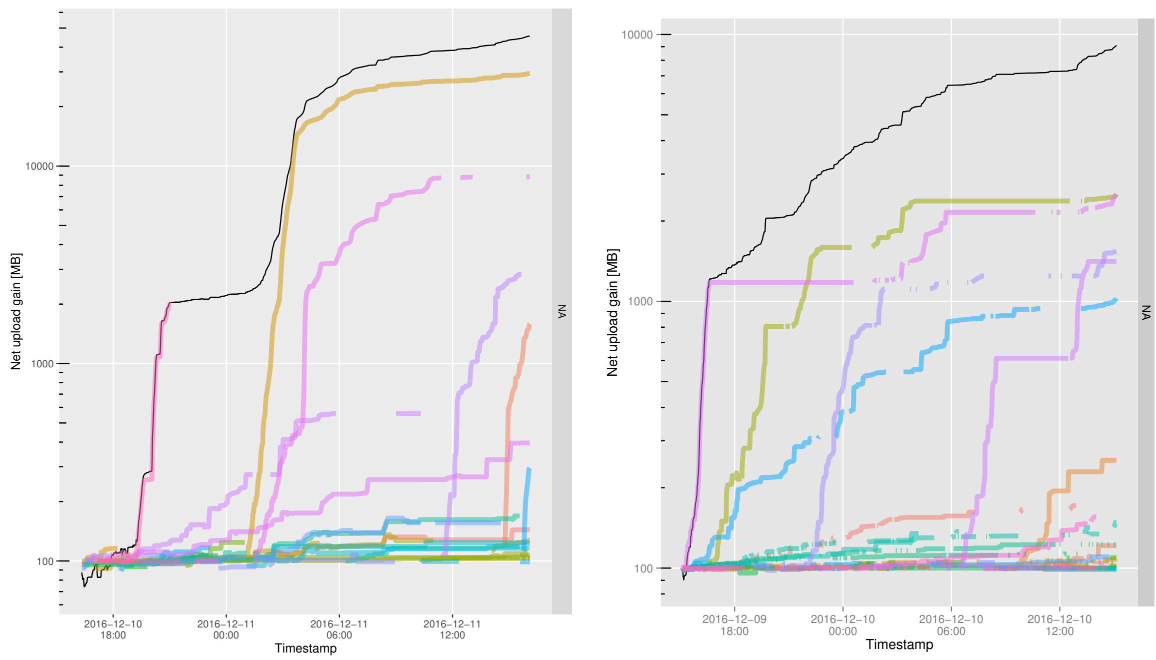
source was already in use. Therefore, we conclude that swarm stimulation works best when the resource in a miner is not fully used, and there are idle swarms in the community.

6.5 Comparing obtained gain with prior work

In this experiment, we will evaluate the result of the proposed system compared to prior work. The comparison experiment run for 24 hours. *Etree.org* will be used as the mining source because the prior version could neither handle other sources nor use the experiment framework in a closed environment. The recommended parameter on the prior work is *SeederRatio* as policy, target ratio is 3, and a 5 minute swarm interval. The result will be shown in Figure 6.10b.

For the proposed system, we applied the scoring policy with stimulation enabled. The other parameters and configuration will be kept identical with that of the prior work. The result will then be compared to the prior work. Figure 6.10a shows the result of the proposed system. Note that the axis that represents net upload gain in both results is logarithmic.

From the figures presented, we can find one similarity. Both of the systems are



(a) 24 hour new experiment.

(b) 24 hour old experiment

Figure 6.10: New vs old experiments (run separately) result on 24 hour (*seederratio* policy)

dominated by only a few swarms. It means that not all of the recent swarms are popular. A popular swarm means that it has more leechers, which impacts the overall credit gain. For the main difference, we found two aspects that show that the new credit mining system is better: policy stability and reducing idleness. In the new system, thanks to the investment algorithm, the miners do not need to unnecessarily switch swarms. It shows that the lines tend to be more continuous compared to ones in prior work. When the line is invisible, it means that the particular swarm was not selected in this round. Secondly, it is the idleness of a swarm which is represented in straight horizontal line in the figures. We can successfully reduce the idle swarm by enabling stimulation. In the experiment of the proposed system, when net upload gained is more than 500 MB, none of the swarms are idle. On the contrary, some swarms in prior work are idle, even when their upload gain is already high. It can be seen in three straight horizontal lines above the 1000 MB gain.

6.6 Sustaining user experience on downloading

As mentioned in 3.2.2, the credit mining system that is implemented within Tribler needs to accommodate user download activity when mining. This experiment will validate that feature. The expectation is that when both credit mining and user download is active, the bandwidth used in user download will remain stable. If only credit mining is active, then the system will maximize the bandwidth if possible.

The experiment runs in the environment as follows. We launched 40 peers and 4 swarms in a single community. One of the swarms contains a file with size 1 GB and rest of the swarms have a 2 GB file. Each swarm has 4 dedicated seeders. We then arbitrarily decide the number of downloaders for each swarm. All of the peers have credit mining disabled except the one in which we have our interest. This peer activates the credit mining system early, without any user download activity. In the middle of the experiment, this peer intentionally downloads a swarm outside of the credit mining system. We will then observe the behavior of our implementation on this event based on the peer's perspective. As for comparison, we launch a similar experiment but the peer mentioned above does not activate credit mining system.

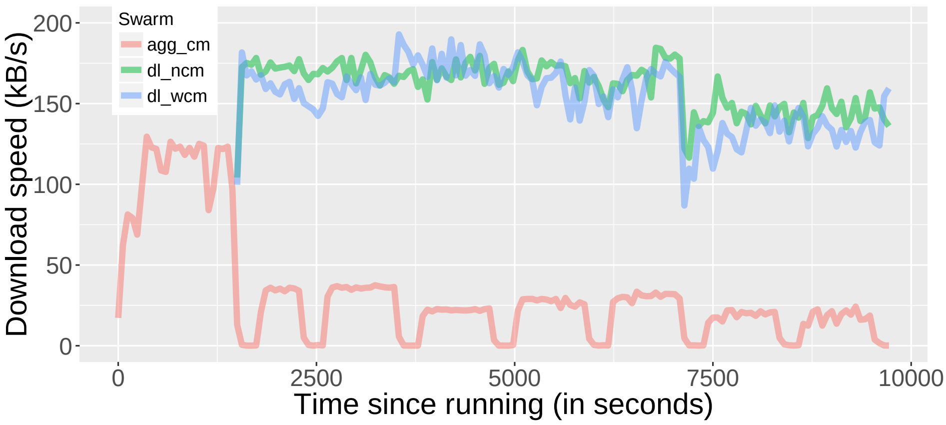


Figure 6.11: Download speed of user download activity.

The results in Figure 6.11 show the download speed on selected peers with and without activating credit mining. The line which shows user download activity is colored as blue and green. The blue line is run on the experiment with credit mining, while the green line is on the experiment without it. We find that the results match our expectations regarding bandwidth distribution. Before the new batch of peers arrived to download, both findings resulted in similar speed constantly in the 150-170 kB/s range. After the new batch of peers arrived, the speed reached up to 150 kB/s for both cases. We conclude that credit mining system has a negligible effect on the overall user experience.

In Figure 6.11, the red line is shown as the accumulative download speed for all of the mined swarms. At first, the mining speed is quite high by reaching 125 kB/s. After the user promptly downloads a particular swarm, the speed is adjusted to around 45 kB/s. Totaling both mining and downloading activity results, the speed is more than 200 kB/s, which is 80% of the total download bandwidth. Sometimes, the mining speed is 0. This is when the observation phase occurred as elaborated in Section 3.2.2. Although not obvious, the general trend of user download and mining speed is contradictory. This can be observable in between timestamp 2500s and 5000s. When the user download speed increased, the credit mining system adjusted its mining speed by allocating lower bandwidth. This allocation method is also valid in the opposite manner, which is when the download speed is decreasing. All of these results show that the credit mining system is able to run with unused bandwidth.

6.7 Swarm performance with credit mining

After we confidently get a high return gain from the previous results, it is worth finding out what are the effects of credit mining implementation on the community. The experiment runs with the same setting as in the Section 6.4, except we add more than one miner. We also use the scoring policy as default, and enable the stimulation. The other parameters are left default.

Figure 6.12 shows the average download speed of all of the members in each of the swarm. In this experiment, no credit mining systems are active. Some of the swarms have reached their maximum download speed such as `file1gb_8`, `file1gb_9`, and `file1gb_10`. We will take this result as a base figure for the next experiment.

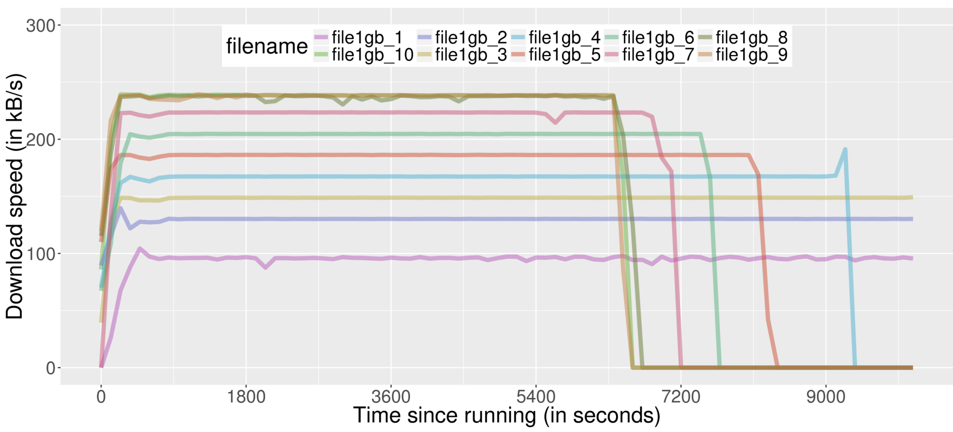


Figure 6.12: Swarm performance without credit mining

Next, we will introduce the credit mining system in the swarms. We start by spawning credit mining system for half of the number of downloaders, which is 25 nodes. Those are dedicated miners, and all of them started simultaneously. Figure 6.13 shows the average download speed from all the downloaders for each of the swarms. We define the swarm as *covered* by credit mining if the performance is significantly either increased or dropped, i.e. by more than 5%. In this experiment, the credit mining covers swarms from `1gb_2` to `1gb_5`. Compared to the previous experiment without credit mining, the download speed is increased from 18.3% (`1gb_4`) to 29.8% (swarm `1gb_2`). For an unaffected swarm, the speed can decrease up to 0.8%.

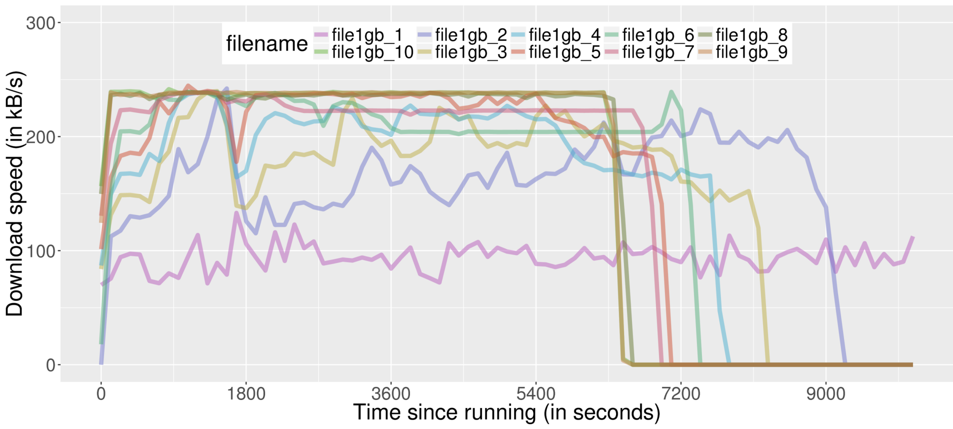


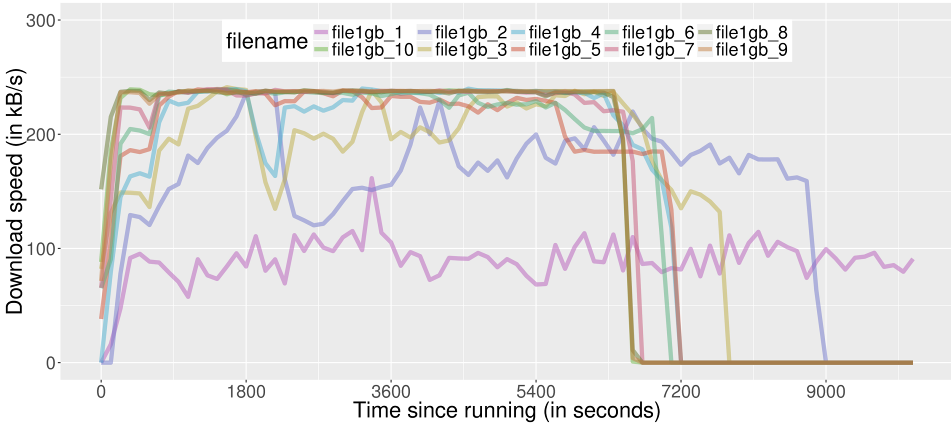
Figure 6.13: Swarm performance with credit mining in the swarm (25 peers)

There are two notable drawbacks when introducing the credit mining system to the swarm, one of which is that the download speed has become unstable. This occurs because of two reasons. First, because of swarm stimulation, there is a higher chance for the miners to become more active on downloading pieces. Second, when switching to a new swarm, miners have incomplete information on the rarest pieces. Therefore, they need to download those pieces to be able to mine. Both reasons are intended to maximize the upload gain as soon as possible. When the miner downloads any piece, it takes the seeder's bandwidth and then observes that the speed for other peers looks like decreasing. These factors combined explains the instability in the Figure 6.13.

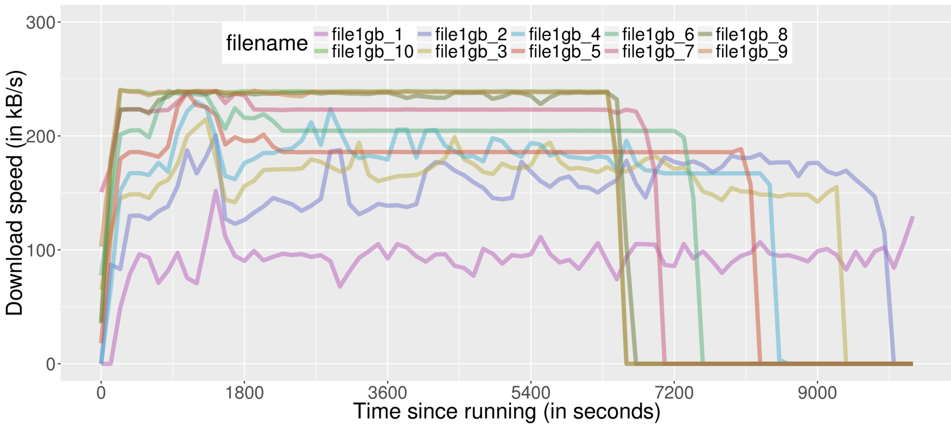
The second drawback is the fact that not all the swarms are boosted by the system. The swarms with a high number of seeders starting from `1gb_6` to `1gb_10` are not boosted in half of the experiment. With the way miners prioritize the swarms, only the lowest-seeded swarm will be filled with miners. In other words, there are not enough miners to boost all of the swarms in this experiment. Note that the lowest number of seeders gives a worse performance than the base experiment. The reason is that the miner takes the seeder's bandwidth that is supposed to be the downloader's. Therefore, some of the downloaders get lower speeds. However, miners can not give back its downloaded pieces to all downloaders. Thus the observed speed is decreasing. In other words, the bottleneck is on the seeder's bandwidth because the seeder cannot deliver the piece fast enough for the miners to distribute the pieces efficiently.

6.7.1 Varying the number of credit miners

In this experiment, we will change the number of credit miners in a swarm. First, we double the number to 50 nodes. The average download speed observed from peers can be seen in Figure 6.14a. Although the download instability is still present, not only the speed is faster than the previous experiment with 25 miners, but also it overcomes the second drawback. Compared to the previous experiment, this shows that the number of miners relates to the boosting coverage of swarms. With 50 miners, it is enough to cover all the possible swarms in this environment. The exceptional case of the swarm with a single seeder still holds. Moreover, with 50 swarms, it can increase the swarm's download speed up to 34.6% (swarm `1gb_3`). The lowest increasing performance is on swarm `1gb_7` with 3.9%. The average increase for the covered swarm is 17.98%.



(a) Swarm performance with credit mining in the swarm (50 peers)



(b) Swarm performance with credit mining in the swarm (10 peers)

Figure 6.14: The effect of different number of miners on swarm.

Second, we also conducted an experiment with fewer miners. In this case, the number of credit miners in the swarm is set to 10 nodes. From Figure 6.14b, it is shown that the coverage is lessened. Now, half of the swarms (from `1gb_5` to `1gb_10`) are not boosted by the miners. The average speed is slightly higher than that of the base experiment, but lower than the 25-miners experiment. The highest performance increase is on swarm `1gb_2` with 20.5%, and the lowest is on `1gb_4` with 8.21%.

6.7.2 The effect of swarm stimulation

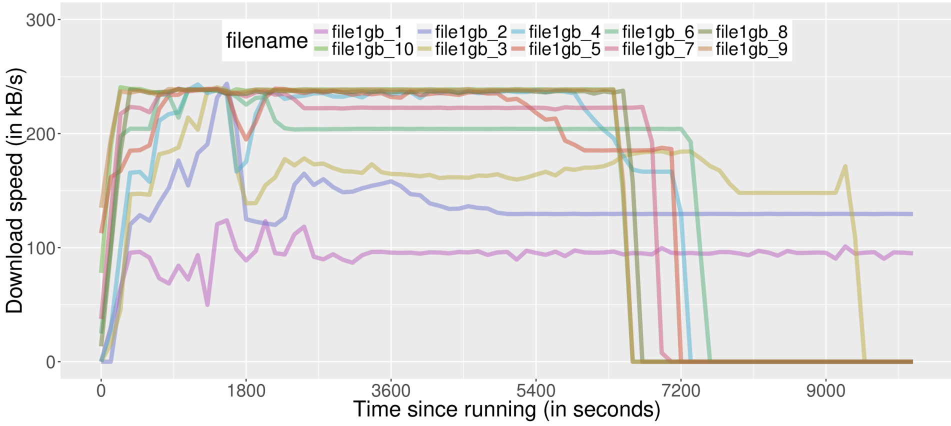


Figure 6.15: Swarm performance with credit mining in the swarm (25 peers) without stimulation.

Our next experiment is conducted to understand the effect of swarm stimulation on the community. As shown before (Section 6.4.2), swarm stimulation can increase the credit gain for the user. A notable difference from this experiment is that the average speed instability is gone, as can be seen in Figure 6.15. However, as a trade-off, the average speed is slightly lower on the boosted swarm. Compared to the experiment with 25 miners, only swarm `1gb_4` is better with 8% increased download speed. The rest have their performance decreased starting from 1% (`file1gb_5`) to 17% (`file1gb_2`) . Even so, it is still better than the base experiment. Swarm `file1gb_4` improves as much as 28% and swarm `file1gb_2`, despite having the worst decreasing performance when compared to the one with 25-miners, still improves by 7.25%. Other disadvantages are on the lower-seeded swarm, namely `1gb_1` and `1gb_2`, which has negligible difference compared to the base experiment, especially in the latter half of the experiment. Therefore, the coverage of the boosted swarm is also reduced.

When swarm stimulation is enabled, it will consume the bandwidth from both the seeder and downloader if necessary. After it finishes downloading rarest pieces, it immediately returns the bandwidth it consumed back to the community in an equal or higher amount. That explains the bumps which represented instability of the downloaders' download speed. However, we argue that stimulate swarm with very few seeders is counterproductive. Although the stimulation can be widely enabled on all the swarms which gives very little drawback, it seems to be unsuitable for some swarms.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis aims to solve the BitTorrent supply and demand misalignment introduced by freeriders. We devised an automatic mechanism to effectively gain credit in the existing credit system. We showed that it benefits both individuals and communities. The user can gain the credit without the need to seed for a long time. The swarms in the community also can get higher download speeds and content availability.

We then proposed the *credit mining system*, an autonomous system to download pieces from selected swarms in order to gain high upload ratio in the future. It can run on top of the traditional credit system that is already widely implemented in both private and public communities. The credit mining system finds swarms with high return potential, picks the rarest pieces, and uploads those to the community using all the available bandwidth.

We focused on the investment algorithm which is the core of credit mining. Two stages of the algorithm were presented. Both stages are important. The *prospecting* stage filters a huge number of swarm while maintaining its resilience by not being solely dependent on a centralized tracker. The *mining* stage only selects the best swarms, those that have a high gain potential. If necessary, it also stops problematic or low potential swarms. We also proposed the *scoring* policy, a highly customizable method to quantify swarms into a score that can be compared with each other, which reduces any possible identical result.

The credit mining system now is fully integrated with Tribler. When enabled, it is tailored to not interfere with users' activity while downloading. Provided with an accessible GUI, a user can easily interact with the system to start investing. Before the system is implemented, it passed several tests and has proven to be stable across platforms. All the components designed so as to not hinder user experience while the credit mining system is active.

The performance of the credit mining system has fulfilled our expectations. All the components were did their tasks properly. Prospecting is both fast and accurate to find and filter swarms on the Internet. The scoring policy successfully selects the most undersupplied swarms and surpasses previous policy accuracy. Moreover, it also stops both saturated and low potential swarms. With stimulating enabled, in most cases, the system use a large portion (80%) of its resources. The upload-/download ratio can reach up to 4.91 with an average of 3.71, although the target was only 2.0. After making the comparison, the current system can gain more credit than in prior work. We also showed that credit miners have a beneficial impact on the community as a whole. When the number of miners is half that of the peers, the credit miners can boost more than half of the swarms in a particular community by up to 29%. Increasing the number of miners can increase the swarm coverage as well as the average peers' download speed.

7.2 Future Work

Currently, credit miners still see other miners as a normal peer. There might be a case in which a miner seeds to another miner, which is unnecessary. The key problem of "Co-Investors" is to recognize and utilize the existence of other miners. When recognizing other miners, investment can be more selective. There is less need to boost a swarm if there are already miners there, for example.

Although we proposed the scoring policy in this thesis, the optimal *multipliers* to reach the highest gain possible are still unknown. With many parameters, a study to find the weight and importance of those parameters is desired. Furthermore, this policy can be extended by adding more parameters while adopting the same calculation method.

Another aspect that still unknown is whether using different policies for different swarms is beneficial. Currently, the policy cannot be changed, and all of the swarms need to comply with a single policy. Moreover, as we have shown in the previous chapter, not all swarms are suitable for stimulation. The *partial mining* mechanism is a method to apply different policies and optimizations to different swarm, in order to get the highest credit gain possible.

Bibliography

[1] Sandvine. Global internet phenomena report 2015 - europe and asia-pasific. [Online]. Available: https://www.sandvine.com/trends/global-internet-phenomena/

[2] E. Adar and B. A. Huberman, "Free riding on gnutella," *First monday*, vol. 5, no. 10, 2000.

[3] G. Hardin, "The tragedy of the commons," *Science*, vol. 162, no. 3859, pp. 1243–1248, 1968. [Online]. Available: http://science.sciencemag.org/content/162/3859/1243

[4] M. Meulpolder, "Managing Supply and Demand of Bandwidth in Peer-to-Peer Communities," Ph.D. dissertation, Delft University of Technology, 2011. [Online]. Available: http://repository.tudelft.nl/islandora/object/uuid:ab227be8-2c68-408b-8b2f-b938cf0f8b8b?collection=research

[5] X. Chen, X. Chu, and Z. Li, "Improving sustainability of BitTorrent darknets," *Peer-to-Peer Networking and Applications*, vol. 7, no. 4, pp. 539–554, dec 2014. [Online]. Available: http://link.springer.com/10.1007/s12083-012-0149-3

[6] A. Das and A. Bhattacharjee, "On analyzing free-riding behavior in bittorrent communities," in *Proceedings of the 2015 17th UKSIM-AMSS International Conference on Modelling and Simulation*, ser. UKSIM '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 482–487. [Online]. Available: http://dx.doi.org/10.1109/UKSim.2015.111

[7] B. Cohen, "Incentives build robustness in bittorrent," in *Workshop on Economics of Peer-to-Peer systems*, vol. 6, 2003, pp. 68–72.

[8] N. Andrade, M. Mowbray, A. Lima, G. Wagner, and M. Ripeanu, "Influences on cooperation in bittorrent communities," in *Proceedings of the 2005 ACM SIGCOMM Workshop on Economics of Peer-to-peer Systems*, ser. P2PECON '05. New York, NY, USA: ACM, 2005, pp. 111–115. [Online]. Available: http://doi.acm.org/10.1145/1080192.1080198

[9] M. Meulpolder, L. D'Acunto, and M. Capotă, "Public and private BitTorrent communities: a measurement study." *Iptps*, p. 10, 2010.

[10] I. A. Kash, J. K. Lai, H. Zhang, and A. Zohar, "Economics of BitTorrent communities," in *Proceedings of the 21st international conference on World Wide Web - WWW '12*. New York, New York, USA: ACM Press, 2012, p. 221. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2187836.2187867

[11] A. L. Jia, X. Chen, X. Chu, J. A. Pouwelse, and D. H. J. Epema, "How to Survive and Thrive in a Private BitTorrent Community," in *Distributed Computing and Networking: 14th International Conference, ICDCN 2013, Mumbai, India, January 3-6, 2013. Proceedings*. Springer Berlin Heidelberg, 2013, pp. 270–284. [Online]. Available: http://link.springer.com/10.1007/978-3-642-35668-1{_}19

[12] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. R. Van Steen, and H. J. Sips, "TRIBLER: A social-based peer-to-peer system," *Concurrency Computation Practice and Experience*, vol. 20, no. 2, pp. 127–138, 2008.

[13] N. Zeilemaker, B. Schoon, and J. A. Pouwelse, "Dispersy bundle synchronization," Delft University of Technology, Delft, Tech. Rep., 2013. [Online]. Available: http://www.ds.ewi.tudelft.nl/fileadmin/pds/reports/2013/PDS-2013-002.pdf

[14] M. de Vos, "Identifying and Managing Technical Debt in Complex Distributed Systems," Master Thesis, Delft University of Technology, 2016. [Online]. Available: http://resolver.tudelft.nl/uuid:e5a817a4-ce0a-4dd3-afd4-d70660b63d16

[15] M. Meulpolder, J. A. Pouwelse, D. H. J. Epema, and H. J. Sips, "Bartercast: A practical approach to prevent lazy freeriding in p2p networks," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, May 2009, pp. 1–8.

[16] S. D. Norberhuis, "MultiChain: A cybercurrency for cooperation," Master Thesis, Delft University of Technology, 2015. [Online]. Available: http://repository.tudelft.nl/view/ir/uuid{%}253A59723e98-ae48-4fac-b258-2df99d11012c/

[17] X. Kang and Y. Wu, "Incentive mechanism design for heterogeneous peer-to-peer networks: A stackelberg game approach," *IEEE Transactions on Mobile Computing*, vol. 14, no. 5, pp. 1018–1030, May 2015.

[18] R. Rahman, M. Meulpolder, D. Hales, J. A. Pouwelse, D. H. J. Epema, and H. J. Sips, "Improving efficiency and fairness in P2P systems with effort-

based incentives," *2010 IEEE International Conference on Communications (ICC)*, 2010.

[19] R. Rahman, D. Hales, T. Vinko, J. A. Pouwelse, and H. J. Sips, "No more crash or crunch: Sustainable credit dynamics in a P2P community," in *2010 International Conference on High Performance Computing & Simulation*. IEEE, jun 2010, pp. 332–340. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5547112

[20] T. Vinkó and H. Najzer, "On the sustainability of credit-based P2P communities," *Central European Journal of Operations Research*, vol. 23, no. 4, pp. 953–967, 2015. [Online]. Available: http://link.springer.com/10.1007/s10100-015-0407-6

[21] N. Andrade, E. Santos-Neto, F. Brasileiro, and M. Ripeanu, "Resource demand and supply in BitTorrent content-sharing communities," *Computer Networks*, vol. 53, no. 4, pp. 515–527, mar 2009. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1389128608003800

[22] M. Ripeanu, M. Mowbray, N. Andrade, and A. Lima, "Gifting technologies: A bittorrent case study," *First Monday*, vol. 11, no. 11, 2006. [Online]. Available: http://firstmonday.org/ojs/index.php/fm/article/view/1412

[23] M. Milinski, D. Semmann, and H.-J. Krambeck, "Reputation helps solve the 'tragedy of the commons'," *Nature*, vol. 415, no. 6870, pp. 424–426, jan 2002. [Online]. Available: http://www.nature.com/doifinder/10.1038/415424a

[24] A. L. Jia, X. Chen, X. Chu, J. a. Pouwelse, and D. H. J. Epema, "User behaviors in private BitTorrent communities," *Computer Networks*, vol. 60, pp. 34–45, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.bjp.2013.12.010

[25] M. Su, H. Zhang, B. Fang, and L. Ye, "A Measurement Study on Resource Popularity and Swarm Evolution of BitTorrent System," *International Journal of Communications, Network and System Sciences*, vol. 06, no. 06, pp. 300–308, 2013. [Online]. Available: http://www.scirp.org/journal/PaperDownload.aspx?DOI=10.4236/ijcns.2013.66032

[26] M. Capotă, J. A. Pouwelse, and D. H. J. Epema, "Decentralized credit mining in P2P systems," *Proceedings of 2015 14th IFIP Networking Conference, IFIP Networking 2015*, 2015.

[27] M. Capotă, N. Andrade, J. A. Pouwelse, and D. H. J. Epema, "Investment Strategies for Credit-Based P2P Communities," in *2013 21st Euromicro International on Parallel, Distributed, and Network-Based Processing*. IEEE, feb 2013, pp. 437–443. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6498587

[28] M. Capotă, J. A. Pouwelse, and D. H. J. Epema, "Towards a peer-to-peer bandwidth marketplace," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8314 LNCS, pp. 302–316, 2014.

[29] M. Capotă, N. Andrade, T. Vinkó, F. Santos, J. A. Pouwelse, and D. H. J. Epema, "Inter-swarm resource allocation in BitTorrent communities," in *2011 IEEE International Conference on Peer-to-Peer Computing*. IEEE, aug 2011, pp. 300–309. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6038748

[30] M. Yoshida and A. Nakao, "A resource-efficient method for crawling swarm information in multiple bittorrent networks," in *2011 Tenth International Symposium on Autonomous Decentralized Systems*, March 2011, pp. 497–502.

[31] M. Wojciechowski, M. Capotă, J. Pouwelse, and A. Iosup, "Btworld: Towards observing the global bittorrent file-sharing network," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 581–588. [Online]. Available: http://doi.acm.org/10.1145/1851476.1851562

[32] A. Loewenstern and A. Norberg, "Dht protocol," *BitTorrent.org. http://www.bittorrent.org/beps/bep_0005.html. Accessed: 22 August 2016*, 2008.

[33] G. Hzel and A. Norberg, "Extension for peers to send metadata files," *BitTorrent.org. http://www.bittorrent.org/beps/bep_0009.html. Accessed: 13 October 2016*, 2008.