# Autonomous prospecting and investing mechanism in distributed system

Ardhi Putra Pratama Hartono

Delft University of Technology

# Autonomous prospecting and investing mechanism in distributed system

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

February 15, 2017

**Author**
Ardhi Putra Pratama Hartono

**Title**
Autonomous prospecting and investing mechanism in distributed system

**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 BitTorrent community is mainly caused by the existence of freerider. Credit system, as one of the widely implemented incentive mechanism, is designed to tackle this issue. However, in some cases, to gain 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 credit mining system, an autonomous system to download pieces from selected swarms in order to gain high upload ratio.

Our main work is to build credit mining system. Specifically, we focus on prospecting algorithm to put an investment in swarms. Two stages are composed : *probing* and *mining*. In *probing*, swarm information is extensively collected and then filtered. In *mining*, swarm is sorted by its potential and then selected. We also proposed *scoring policy* as a method to quantify swarms into numerical score. Each detail of the sub-algorithm is presented and elaborated.

Finally, we implement and evaluate credit mining system in both live and controlled environment. The system is fully integrated with Tribler and able to adapt with user activity. It correctly selects undersupplied swarms. In the terms of advantages, user can gain upload/download ratio up to 4.54 by using 80% of its resource. Majority of the swarms in the community also get its average download speed increased by up to 34%. Based on the results, we showed that the implementation of credit mining system is beneficial for both parties, especially to cover 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
February 15, 2017

# Contents

# Chapter 1

# Introduction

Since the introduction of the Internet on 60's and world wide web (WWW) on 90's, more and more people are connected to each other. Recent research shows that peer-to-peer (P2P) Internet already back at its prime by dominating the traffic as shown in figure 1.1 [1]. User interaction in the Internet community can be expressed in various fashion. Many applications and protocols run on top of P2P system, for instance online gaming, computing, and file sharing. Specifically in file-sharing P2P application, to ensure user has a flawless experience, it needs high throughput by maximizing all the connection a user has. However, not all the peers consistently share the file. This phenomenon is called *freeriding*.

## 1.1 The freeriding phenomenon

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

Freerider is defined as the user behavior that selfishly consumes all the resource without giving back. It may add vulnerabilities in the system [2]. With only a few of the user provide the service for many, it eventually becomes more centralized than a decentralized system. It also may degrade the system performance [2]. Adar and Huberman showed that lots of P2P peers are always shown self-interest and rationality that can be considered as freeriding. If freeriders become the majority in file-sharing peer-to-peer system, as they occupy a significant amount of resource, eventually bottleneck in the system will occur. As the time goes, an honest peer may not feel satisfied and decided to leave the system. With important peers leave the system, it will degrade more. Moreover, the system may lost the file that used to be served by the leaving peers. The system becomes unhealthy and sooner or later will be completely left by 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% |

sandvine

(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% |

sandvine

(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 called "tragedy of the commons" [3]. This problem was popularized by Hardin [3] in 1968. This social dilemma emerges because of the overuse and overexploitation in the shared resource without feedback from the user. As Hardin stated in his paper: "Freedom in a commons brings ruin to all", uncontrollable participant in an environment with limited shared resource could take something out of the common to fulfill their own gain.

*Extreme freeriding* is a behavior where one does not upload anything while keep downloading data. In current standards, it is rarely happened. Instead, it is more common to find *hit and run* behavior [4]. Hit and run (HnR) is a situation where a user has finished downloading then immediately stop his contribution [5]. Hit and run also often referred as one of the freeriding behavior that peer-to-peer communities wanted to prevent. Das and Bhattacharjee also studied the freerider behavior in BitTorrent communities. They conclude that freerider in BitTorrent may not degrade performance as long as the swarm has at least one dedicated and available seeders. The potential availability of seeder also become a factor that keeping the swarm alive [6].

## 1.2 BitTorrent protocol

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

In general view, BitTorrent consists of peers who participated in file-sharing and *tracker*. *Tracker* is responsible for monitors the distribution and progress of the file and peers in the swarm. *Swarm* is a set of peers formed with the same purpose of downloading or uploading certain files represented in `.torrent` metadata file. Static `.torrent` file, which contains information such as tracker addresses and unique hash value of this swarm, is created by a peer who wants to publish their files. Files in a swarm consist of several *piece* or file chunks. A piece is exchanged by the peers in a particular period. A peer actively participated in many swarms on the uninterrupted time-frame called *session*. *Seeder* is a type of peer who has complete files and upload (seed) its pieces to other peers. Leecher is a type of peer who download from a particular swarm.

BitTorrent uses *tit-for-tat* mechanism to reward good behavior and punish bad behavior. This mechanism tried to solve fairness issue introduced by freeriders [7]. *Tit-for-tat* mechanism always prioritize peer who has uploaded something. As for downloading, BitTorrent always picks *rarest-first* piece based its availability in the swarm. This technique makes sure that a complete file is distributed in the swarm. *Tit-for-tat* valid only in a scope of single torrent. That means, the state from one community can not be carried to another community. This causes *tit-for-tat* works best only in short term transaction and limited parties. Nevertheless, Andrade et al. showed that BitTorrent is indeed increased cooperation with only less than 10% peer is uploading something[8].

In order to find rarest pieces, piece information on peers is necessary. In BitTorrent, there are four methods to discover new or updated peer. 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 local network. PEX is a mechanism to efficiently contact a peer directly in order to exchange up-to-date information.

There are a lot of BitTorrent *communities* that served as a portal to store `.torrent` file. In general, community in BitTorrent can be divided into two categories: *public* and *private*. Public tracker means everybody can join the swarm served by that tracker. In the other hand, private communities are closed community which can be accessed by passing particular requirement [9, 5]. Typically, private communities have higher performance compared to private communities [9]. Meulpolder et al. measured that they have 3-5 times higher download speed compared to public communities [9]. *Private community* typically has higher seeder-to-leecher ratio (SLR) that determines the faster download speed [8]. In such a community, the administrator enforces several policies such as *Share Ratio Enforcement* (SRE). SRE define the amount a user need to upload before able to download from the commu-

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 has originally been created as a mechanism to prevent free-riding 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. |

nity [10]. Higher performance comes with a drawback: in private communities, it is also very difficult to get a new membership and very easy to be kicked out [11].

### 1.2.1 Tribler

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

All of the Tribler main components such as end-to-end encryption, channel discovery, and many others relied upon database and dissemination system called `Dispersy` [13]. Dispersy maintains and performs the communication between Tribler peers in fully decentralized manner. Dispersy is able to circulate the message in one-to-one or one-to-many within a group of node called *community*. Users can adapt and implement its desired *community* by itself. It is including how, what, and where the communication will occur.

Tribler implements several Dispersy *communities* on its core function. de Vos summarize the recent community in Tribler. Important features such as channel

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

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

discovery, search within communities, end-to-end Tor-like operations, and currency mechanism are shown in table 1.1. *Channel* is a collection of torrent that has extra capabilities such as vote system, spam prevention, and comment (social) attributes. Every user can create his own channel, add and remove torrent to it, and maintain its activity.

## 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 *incentive mechanism*. Incentive mechanism is a method to reward user for its contribution. In some cases, it also fines user for its misconduct. SRE in private community is a good example which increases user contribution and swarm performance. It proves that freeriding behavior can be prevented by proper incentive mechanism. By showing goodness, specifically, by uploading data to others, users should get a reward. However, users are typically selfish and always try to maximize their own benefit [17].

Incentive mechanism is essential as it is one of the property to increase general performance. Meulpolder discussed several kinds of incentive mechanism techniques. The technique can be combined and complement each other. Those are : (i) direct reciprocity, (ii) indirect reciprocity, (iii) centralized reputation, (iv) decentralized reputation, and (v) currency [4]. *Reciprocity* focused on the relationship between peers. As shown in 1.2, it categorized into two : *downstream* and *upstream*. *Reputation* technique is more straightforward. The information of user behavior in the past is stored (centralized or decentralized). This information iteratively updated and spread through all the peers. Last technique is *currency* which uses *credit* to incentivize user. By theory, it can emulate other incentive method by quantifying reputation/help. Because of its wide applicability, we will focus on improving the *currency* technique. *Private communities* with SRE is the most common implementation of currency mechanism.

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

There are several attempts of inventing incentive mechanism. Kang and Wu proposed an incentive mechanism for dynamic and heterogeneous peers with game theory. In their system, each peer can set a price for service it provides. The

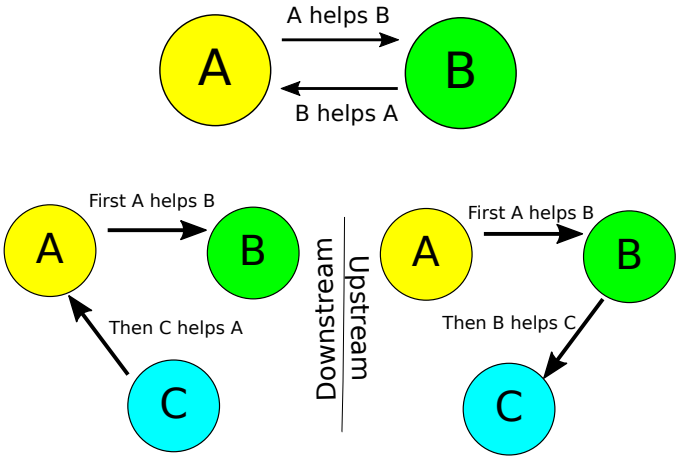



Figure 1.2: Direct and indirect reciprocation

buyer (downloader), then able to negotiate with the seller (uploader) regarding the content price and its bandwidth allocation [17]. In other research, Rahman et al. proposed effort-based incentive to advocate fairness between peers [18]. In this system, user awarded based on its effort, which is relative on to its capacity. Currently, none of these researches are widely applicable. It either needs modification on the protocol, or only works under certain type of applications. Nowadays, the users who want to get credit may need to standby for a long time waiting for someone to download their files [11]. Ironically, this approach is inefficient, bandwidth wasting, but commonly in practice[11].

## 1.4 Thesis structure

Our objective is to establish a layer, named credit mining system, on top of existing currency system in order to lessen the hit and run (HnR) effect on BitTorrent communities and to automatically let user gain credit efficiently by uploading only necessary data. Both of the purposes are directed to one vision : improving performance on BitTorrent communities. The layer is supposed to work without 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 credit mining system and its requirements. Chapter 4 shows the core algorithm that we propose in order to prospect and invest. Implementation of the mechanism and its experiment setup will be elaborated in chapter 5. Chapter 6 shows the performance of credit mining system. Chapter 7 then concludes the work and mentions possible future work.

# Chapter 2

# Problem Description

The nature of P2P system is to help each other achieving common purpose. In file-sharing P2P, it is to get a particular content in the Internet. However, in the 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 an *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 benefit of this research is the existence of automatic caching layer which ensures content availability for both long-lived years-old and freshly created content. This includes the automatic mechanism where every one contribute and in the same time can gain benefit to themselves. Specifically, by devising a credit mining system, we take an important step to reduce the needs of human intervention on contributing to improve other user's experience.

In this chapter, the problems that are the main concern of this thesis will be elaborated. We will discuss performance problem in BitTorrent system, specifically by looking at its supply and demand misalignment. After that, the *investment* as potential behavior that can tackle this problem will be elaborated. It covers the importance, potential gaining, and desired effect of the possible credit investment. After specifying problems, prior works on credit mining will be reviewed. Further improvements on those work are the core of this thesis. Lastly, two research questions on investment and how to prospect 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 P2P community, especially BitTorrent, may lead to system seize-up. There are three statuses observed: *crash*, *crunch*, and *sustain*. *Crash* and *crunch* is the condition where there are too much credit and lack of credit, respectively [19, 20]. To preserve swarm sustainability, there are two aspects that need to be considered. The first one is the swarm condition such as

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

| community | download speed (kbps) | | | avg % unconn | avg s/l ratio | seeding duration (hours) | | |
|---|---|---|---|---|---|---|---|---|
| | mean | median | top 10% | | | mean | median | top 10% |
| The Pirate Bay | 1037 | 333 | >2134 | 47.0 | 2.6 | 11.7 | 1.8 | >31.4 |
| EZTV | 928 | 294 | >1575 | 48.3 | 6.6 | 18.1 | 4.7 | >52.0 |
| TVTorrents | 3590 | 1362 | >7692 | 32.5 | 104.5 | 44.1 | 17.9 | >130.7 |
| TorrentLeech | 4937 | 1030 | >7166 | 33.9 | 25.4 | 50.4 | 16.8 | >153.9 |
| PolishTracker | 8625 | 1331 | >14128 | 20.6 | 63.8 | 58.0 | 20.2 | >156.0 |

file size and initial credit distribution [20]. Vinkó and Najzer showed that large file size could decrease the sustainability of a swarm. As for initial credit configuration, it depends on the community itself. The wrong amount can crash the system, while with the right amount, overall throughput can increase. Secondly, it is the peer behavior [19]. Rahman et al. concluded that selfish peer who only upload in order to continue downloading (freeriding) can badly harm the swarm. Ironically, few high performance individual can lead to lower community performance [20]. Therefore, it is needed to balance both peers and community needs.

Despite has different performance, both public and private community suffer from a similar issue: "Poor downloading experience". It is widely known that public community generally has low SLR which directly affects the swarm performance. In the other hand, private tracker suffers from "*poor downloading motivation*" as described by Chen et al.[5] although private community was intended to solve low SLR issue. Both issues are caused by the misalignment of supply and demand.

### 2.1.1 Supply and Demand

Supply and demand for both public and private BitTorrent communities have been intensively studied previously [9, 21]. Andrade et al. showed that in BitTorrent community, the supply is mostly meet the demand. We define a supply and demand *misalignment* is a condition where supply could not meet the demand without noticeable performance degradation or wasted resource. Significantly high or low seeder/leecher ratio can lead to supply and demand misalignment.

In public community, there is significantly less seeder/leecher ratio compared to a private community which enforces SRE [9, 21]. This will result in a lack of supply for demand in the swarms. On the other hand in private community with SRE, there are consequences for peers who do not seed. This enforcement will make the community end up with a lot of peers who 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 seeder and leecher in private community can reach up to 1589 (not shown in the table) with average can reach more than 100. By contrast, in the public community, there is only 2-7 seeder per leecher and its maximum ratio is 46 [9]. The download speed of private communities are 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 resource shared within the swarm to be distributed over the peers who wanted it. Two possible reasons why this happened are : (i) an asymmetric number of seeder and leecher, which seeder cannot compensate; and (ii) lack of incentive mechanism in the higher level aside from BitTorrent *tit-for-tat* [21]. With the introduction of private communities which enforce policy such as SRE or any incentive mechanism, the problem shifted to a phenomenon called *oversupply*. The main reason a swarm is *oversupplied* is because swarm has overwhelming number of seeders. Jia et al. mentioned that *oversupplied* swarm may result in lower bandwidth allocation for other users [11]. Both undersupply and oversupply is the sub-case of supply and demand misalignment. Undersupply condition can be solved by adding more high performance peers to boost download experience. In the other hand, oversupply problem is not trivial to solve.

In oversupplied swarm, users may find it difficult to earn the credit. This is because the problem described by Meulpolder called "upload competition" [4]. Two conditions from peers perspective must be fulfilled to make P2P system sustain, which is peers must be cooperative, and cooperative peers must stay as long as possible in the swarm [4]. Table 2.1 shows that in average, users in private community standby for seeding for 50 hours. Rahman et al. also stated that oversupply swarm may result to system seize up by *crashing* [19]. Jia et al. find out that common way to survive expulsion is by seeding longer. However, if this behavior happened in a long period, it might produce significant imbalance on supply and demand as seeder kept seeding a particular torrent without switching to another swarm. Moreover, user's resource may be wasted. Sustainability of a swarm is in a risk in this situation as the misalignments between supply and demand can worsen the downloading experience in BitTorrent. The imbalance of demand and supply will harm new members of private communities. In 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 shown in figure 2.1. Request and seeding behavior is another term of user downloading and uploading behavior, respectively. In his work, Meulpolder showed that using naive random seeding behavior is not sufficient to make P2P system balance[4]. Unbalance system can lead to unsustainable community. Therefore, it is important to study seeding behavior for each peer in order to balance supply and demand in BitTorrent swarms.

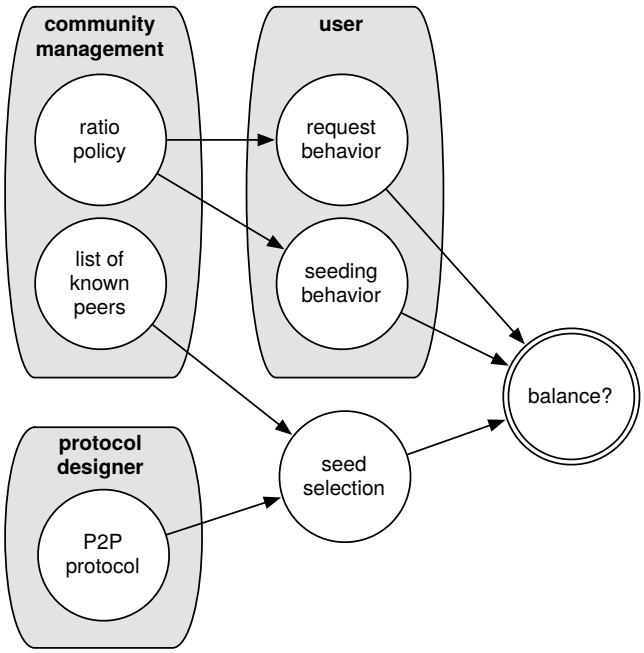



Figure 2.1: System properties and its relation to P2P balance [4].

.

### 2.1.2 Investing as seeding behavior

The activity of spending credit can be categorized as three : (i) *trading*, (ii) *investing*, and (iii) *gifting* or *donating*. When a user want to spend his credit to get something, we define it as *trading* or buying. This is the most common case because in credit-based community, every content has its price. *Gifting* is the case when a peer consciously intend to not getting any direct, immediate, or obvious return of his credits [22]. The purpose of this action is usually to improve the performance of a community or just the act of altruism. Gifting can also be considered as a reward because a peer is satisfied with the community. Investment is the activity of spending credit with the expectation of obtaining credit to use later on.

The ideal situation to balance performance and sustainability in BitTorrent community is by aligning supply and demand as discussed in section 2.1.1. By gifting and helping undersupply community adequately, the optimal situation can be achieved and tragedy of the common can be prevented [23]. However, P2P user are typically selfish in economical way [24]. Andrade et al. also shows that user who contribute more to the community, actually consume a lot from it [21]. This explains that BitTorrent users are not altruistic enough to seed continuously. Therefore, investment is the most feasible method to balance user and community needs.

We define the activity of downloading with the expectation of obtaining credit

(by uploading) in the future as *investment*. A user can *prospect* which community he will invest regardless of his own resource. Not all the swarm need to be seeded as we shown the drawbacks of oversupply. The process of prospecting and identifying swarm that needs to be seeded is important to balance content availability and personal gain. By providing proper prospecting function, users could help each other to improve a swarm quality. In good investing algorithm, users can make better use of its resource to gain credits.

In classical economic principal, the key to gain benefit is to buy low and sell high. By finding popular item and suitable swarm, the potential of investment become huge. For example, in flashcrowd case, an item become so popular and undersupply might happen. Flashcrowd effect is the sudden increase in demand due to various reason. For instance, recently published torrent is one of the reasons where flashcrowd effect take place [25]. If all user can download from oversupplied swarm and also upload to undersupplied swarm, it will reduce the misalignments on those two cases. Specifically, uploading to undersupplied swarm will increase the probability of gaining high credit.

## 2.2 Prior Credit Mining Research

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

In 2013, Capotă et al. introduced bandwidth investing in BitTorrent private communities. He applied speculative download (as shown in figure 2.2) on prospected swarm. This research used activity data crawled from Bitsoup[1] to evaluate their system. Every swarm is analyzed whether it will keep the swarm in *cache* or discard it. Swarm is scored by predicting future upload speed defined in multiple regression model[27]. One of the limitation is that the more swarm need to be assessed, there is less chance the algorithm will find suitable cache to replace. Other limitation is that *multivariate adaptive regression splines* (MARS) implemented in this system is very costly and complex.

A year after, in 2014, a research to align supply and demand in BitTorrent network is conducted. The idea is each peer monitor their swarms to detect whether there will be potential undersupply. If such a condition is found, one will broadcast *help request* to specialized peers in order to seed this swarm. Specialized peers, called *helpers*, tries to download as little as possible while upload as much as possible using *libtorrent* share mode. They implement multiple helper and observe its

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

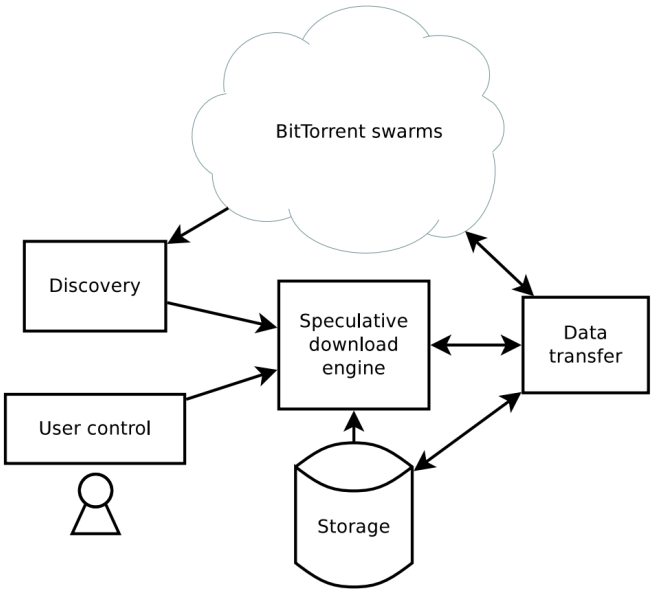



Figure 2.2: Speculative download mechanism [27]

.

effect to swarm. Their experiment result shows that using share mode in closed environment will increase 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 his previous work into Credit Mining System. Credit mining system able to monitor multiple swarm in one moment, and then decide which swarm this system will give its bandwidth to. It uses simpler policy on choosing swarm compared to multivariate regression model in previous work [27]. The overview of mining process is shown in figure 2.3. The experiment was conducted in live fashion on RSS *etree.org*[2] public community. They observed *net upload gain* which defined as difference of uploaded bytes and downloaded bytes. Proposed policy and framework resulted in positive effect to both the community and individuals.

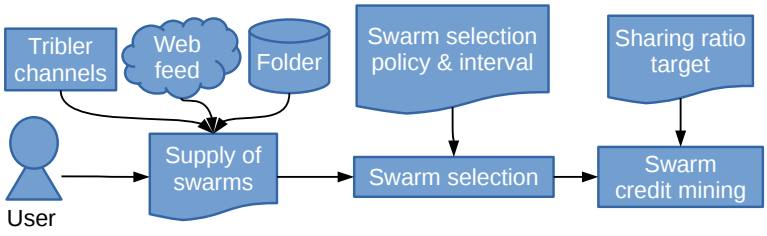



Figure 2.3: The overview of credit mining process [26]

.

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

## 2.3 Prospecting good investment

Investing can not be separated by another activity called "prospecting". We define *prospecting* as the activity to identify and measure a swarm in the hopes of getting more credit by putting some credits as capital investment. In practice, not all undersupplied swarm need to be seeded, even more oversupplied swarm. A swarm can be chosen by considering its seeder-to-leecher ratio, piece availability, our available resource, and many more. In credit based community, correct investment may spark the community thus improving the performance. From user perspective, good prospecting algorithm can result a high return of the resource used from both investing and prospecting.

Swarm information is crucial in prospecting. Most researches have measured BitTorrent by crawling its community pages [11, 8, 24, 9, 5, 10, 27, 21, 29]. This way, they can get the data summarized by the pages. Some researches contact the tracker regularly or using its dump logs [30, 8, 6, 29]. Most of the research using logs as its dataset only use 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 need 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* need compiled data from 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 credit mining system is to help undercapacity swarm, while at the same time to get credit for uploading data. The system need to find which swarm that might have high return in *prospecting*. The *investing*, which relies in prospecting, is more complex. It has to consider used-limited resources and its initial capital as additional requirement. Resource can be in several forms such as bandwidth, memory, or storage. Although the term "good" may be relative, we intend to show the performance of the system by counting how much net credit it gained. Therefore, we define the first research question as :

*How to prospect swarm and what is good investment?*

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

In order to answer the question, we formulate technical challenge that need to be solved. The challenges include engineering and performance evaluation aspect. Both prospecting and seeding process may disrupt user activity. Therefore, it is important to take advantage of unused bandwidth. Many characteristics of swarm such as low seeder, practically dead swarm, and new published swarm 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 autonomous prospecting and investing system which can yield high return of investment. In evaluating the system, it is necessary to observe the effect of credit mining system in a whole. This observation can be done by deploying the system in closed environment.

## 2.4 Substituting investment cache

In the first question, we have addressed how to gain credit as much as possible efficiently and in non-disruptive manner. However, this is not answering the limited resource available at the user disposal. Investment is a tedious activity if being done manually. Users are often forced to seed for excessively long time to maintain adequate credit [11]. By seeding unproductively, user wastes his resources, such as bandwidth, storage capacity, and computer power. In this issue, we will specifically focus on the storage limitation. The term *storage* and *cache* can be used interchangeably. It points to the container used to store the swarm data as the source of investment.

To start seeding, the data must be available locally in the storage. By having many data, there are higher chance to seed many swarms as well. Storage capacity is limited and a mechanism to optimize its usage is needed. Eventually, it is necessary to replace obsolete investment. Several reasons to do so such as gaining less profit, unstable credit, or unreliable swarm. Moreover, as mentioned before, it is better to avoid seedinng oversupplied swarm because it will affects swarm's sustainability. By replacing old swarm by the new swarm, the gained credit must be remained stable or higher. Although user can control the investing process, it is desirable 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?*

Technical challenge that arise by this question is covering several aspects. The characteristics of unproductive swarm need to be defined. In concern with previous question, it needs to numerically comparable with other swarm. Because replacing swarm is costly, there must be a precaution to help unproductive swarm to gain more credits. If all the methods failed, a swarm that potentially gain higher credit need to be chosen.

# Chapter 3

# Credit Mining System Design

The goal of this thesis is to implement a system where it can be used to both gain credit and improve swarm performance. In this thesis, we introduce "Credit mining system", an automatic investment framework on swarm with multidimensional properties. With credit mining system, locally, a user can gain credit with limited bandwidth allocation without any intervention needed. We assume the *credit* is the amount of bytes from the system to the community and vice versa. From higher perspective, credit mining system will help a swarm to keep alive by providing integral pieces to the peer who might need it. Although credit mining system will be implemented in Tribler system, it is possible to apply this feature to any file-sharing BitTorrent client.

Firstly, the dependencies of credit mining system which is *libtorrent*'s s*hare mode* will be elaborated. `Share mode` is a module which can be activated as an intent to helping a swarm instead of normal content downloading. This module will be explained in detail in section 3.1. Afterwards, the design of 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 just a collection of specification. It is free to be implemented with any languages. One of most popular implementation is *libtorrent*. *Libtorrent* is written in `C++` and has `python` binding. *Libtorrent* started in 2003 by Arvid Norberg and it implemented most of the BitTorrent specification. Most of the well known extension such as DHT, PEX, magnet link, multi-tracker, and webseed also have been implemented. *Libtorrent* is widely used by many torrent client such as Deluge, qBittorrent, Free download managers, and many others.

One of the crucial feature used in this work is *share mode* [1]. Initial work performed by Capotă et al. also used this feature[26]. By enabling share mode, it

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

means that one is not interested in downloading the file in a swarm, but in gaining higher share ratio. It is done by download as little as possible and upload as much as possible. A swarm downloaded in share mode may never finish as *libtorrent* only download piece of a torrent which satisfied the share mode requirements.

Share mode algorithm works heuristically as it estimates the rarest piece available in the swarm based on participated peers. The algorithm is presented in Algorithm 1. For clarity, we divide this algorithm in two part. First part is to pass all the restrictions. It tries to find missing pieces for each peers (line 4), disconnects some of the seeder because of connection limit (line 8), and reduces the number of missing pieces with twice the number of seeder (line 10). For the last, it is based on assumption that seeder 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 reached (line 14), or too many parallel download (line 17).

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

There are two limitations on this feature as we observed. Firstly, share mode did not check whether a swarm is good enough to perform this operation. It only tries to find popular pieces regardless of the swarm condition. It has high possibilities that swarm with poor capacity will have very low uploading rate. The bandwidth used to check torrent pieces is wasted and the "investment" will not go well. Therefore, user is responsible of whether share mode will yield high credit or not.

Secondly, the biggest limitation of share mode is the possibilities of getting bottleneck because of its strict policy. In 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 1 piece (5%) at a time in share mode (line 17). It is also necessary to wait 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 and the rarity of pieces may have changed. If the system is too late to completely receive piece, or the piece is not uploaded fast enough, this piece could be obsolete as nobody wants it anymore. Therefore, the condition in line 14 may never be satisfied. If other pieces could not cover this condition (by uploading to more than $T$ peers), then share mode unable to continue. The system then will not download nor upload any pieces anymore.

## 3.2 Credit Mining Architecture

Credit mining system is intended to do its task automatically with minimal user intervention. The way this system designed is to align supply and demand of chosen swarm. Short term advantage of this approach is to gain credit by minimizing

**Algorithm 1** Libtorrent share mode algorithm

**Require:** $T$ as share mode target

▷ Part 1

1: $missing\_piece \leftarrow 0$
2: **for all** $p \in connected\_peers$ **do**
3: **if** $p$ is a $leecher\ and\ p$ is not in share_mode **then**
4: $missing\_pieces$ += $total\_pieces - pieces(p)$
5: **end if**
6: **end for**
7: **if** $|connected\_seeders|$ in $connected\_peer > 90\%$ **then**
8: disconnect excess seeder
9: **end if**
10: $missing\_pieces$ -= $2 \times |connected\_seeders|$
11: **if** $missing\_pieces \leq 0$ **then**
12: **return**
13: **end if**
14: **if** $num\_downloaded \times T > uploaded$ **then**
15: **return**
16: **end if**
17: **if** $downloading > 5\% \times num\_downloaded$ **then**
18: **return**
19: **end if**

▷ Part 2

20: $rarest\_rarity \leftarrow MAX\_INTEGER$
21: **for all** $pc \in pieces()$ **do**
22: **if** $pc$ not in $collected\_piece\ and\ peer\_count(pc) \leq rarest\_rarity$ **then**
23: $rarest\_rarity \leftarrow peer\_count(pc)$
24: $rare\_piece$.push($pc$)
25: **end if**
26: **end for**
27: **if** $|connected\_peers| - rarest\_rarity < T$ **then**
28: **return**
29: **end if**
30: download $random(rare\_piece)$

download and maximizing upload. In the long term, this potentially increase overall performance of other user as well.

The system can be implemented on any torrent client. In Figure 3.1, it shown the compulsory elements and the relation between *credit mining system* and *torrent client*. Currently, we assume that every torrent client also track how much data a user has been downloaded and uploaded in *credit storage*. Naturally, any torrent client must have *Torrent client downloader* module as well. *Libtorrent* library also must exist as part of dependencies. Other required feature is the ability of discov-

ering peer by all method (DHT, PEX, LSD, etc). In some cases, peer discovery function is disabled for security reasons. While disabling one should not affect 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 *predownloader*. The *manager* receives the *settings* from user in the initial phase. To control mining process, user can only interact with values specified at *settings* elements. User action also limited to only add and remove *mining source*. Each of the source will be assigned with a *miner* depend on the type of the source. Miner also has sub-elements as part of the system. In-depth explanation of mining source and miners will be discussed in Section 3.2.1. We introduce the *predownload* mechanism as a way to evaluate a source as early as possible. Predownload mechanism as part of prospecting methodology will be discussed in Chapter 4.

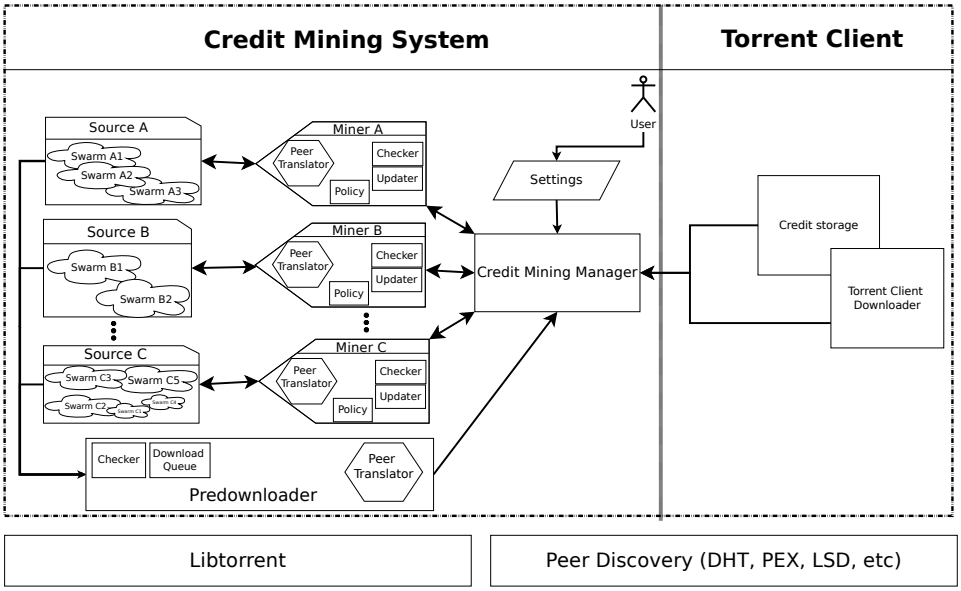



Figure 3.1: Credit mining components.

Before credit mining system is executed, user can change the default settings used in the credit mining system. For example, if user has a lot of memory available, it is desirable to mine many swarms in one time by changing *max_torrents_active* parameter. Another example is if user has large storage, decreasing *share_mode_target* may result a better return. Some settings can not be changed when mining process run already. Table 3.1 shows the settings used in this system. After user provide the setting and run the *credit manager*, user can start mining by adding sources. A source usually consists of several swarms with different size, availability, and capacity. Each of the source will be assigned with one *miner*. Before the miners start mining, *predownloader* fetches swarm information and then give the results to the manager. Manager will propagate the swarm information to miners and let

the miners decide which swarm to mine based on that information. Manager also monitor the main torrent downloader to adjust miners bandwidth allocation. The credit gained from each of the miner will be reported to manager, then it forwards the results to credit storage in 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 swarm that will be downloaded | Yes |
| 2 | *max_torrents_per_source* | The maximum number of stored torrent 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 swarm and start/stop 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 *predownload* phase | No |

### 3.2.1 Mining Sources

Currently, credit mining can accommodate three types of sources. There are directory source, RSS source, and channel source. In credit mining system, one source is assigned to one miner. The miner will start working the moment the source is defined and added to the manager. A miner periodically perform checking and other operations on all the swarms in the source.

First type of source which called *directory source*, is very straightforward. It takes a path as an argument and verify it before run the miner. The miner starts by looking at all the file in specified directory that have `.torrent` file type. Each of the file is examined and validated. Corrupt or invalid file will be discarded and deleted automatically from the disk. In order to keep the performance and prevent disk bottleneck, the miner sort the files alphabetically and put it into queue one by one. Miner periodically monitor both the directory if there is a new file and the queue to assign it to the manager. Miner eventually will pop an item from the queue, build a suitable format for mining, and notify manager to include this swarm.

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

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

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

Provided the identifier of a channel, miner will continuously try to find and join the channel in *AllChannel*. By joined the channel, miner can get the list of torrents and its properties. After miner joined the channel, the swarm information will be handled and downloaded by Tribler. Miner then monitors the local database whether a new data has been fetched and there is a space for adding possible new torrent to the manager. After knowing swarm information, miner will proceed to download the swarm metadata (`.torrent` file or magnet link). Afterwards, the mining format will be built and manager will be notified of a ready swarm.

### 3.2.2 User activity awareness

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

[2] `http://bt.etree.org/rss/bt_etree_org.rdf`
[3] `http://www.mininova.org/rss.xml`

In response to that issue, we implemented another module in credit mining system to adjust its mining activity to user download activity. Credit mining system periodically observe whether there is a user downloading activity. If there is not any, then it can notify the miners to use all the bandwidth available. In the other case, the system will limit the download and upload rate of mining activity to left-over 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 credit mining system. These optimizations are optional and can be left out. However, as the system itself is not perfect, an improvement to support this system is advantageous. There are three optimization we proposed : duplicate elimination and swarm blacklisting.

In preliminary work by Capotă et al., credit mining system able to distinguish duplicate content in the P2P communities. It is highly possible that an exact same file have different *infohash* as its identifier. Infohash of a swarm is an SHA1 hash consists of 40 hex character. An *infohash* of a torrent can come from many aspects such as different piece size, categorized as private or public swarm, or even directory name of the files [26]. We used *Levenshtein distance* to measure the different between one swarm and another by considering the files in it. Specifically, its names and length. In the end, we only mine the one who has higher number of seeder. The swarm comparison executed whenever there are new swarm reported by the miners. Eliminating duplicate swarm can lead the peer interaction to be concentrated in one of the swarms. Thus, the performance of participating peer might be improved.

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

# Chapter 4

# Prospecting Algorithm

Prospecting a swarm is one of the key elements of credit mining system. The prospecting module can be divided into two main stage, named *probing* and *mining* stage. Both of the stages rely on the swarm measurement by looking at its evolution [25]. Credit mining system *translates* the peer information it receives to predicted number of potential seeder and leecher. It also stores the *availability* and rarest piece of the swarm by looking at the pieces possessed by each peers.

When a swarm is marked as potential source of investment, credit mining system will extensively fetch information of that swarm in *probing stage*. It only executed once and then decides whether the swarm is accepted or discarded. If the swarm is accepted and is possible to get the return of investment, credit mining system activate the *mining stage*. This stage runs periodically as long as the system still running. The combination of both stages resulting the whole prospecting algorithm.

## 4.1 Probing stage

Probing stage is important to decide which swarm that need to be mined. Wrong or sub-optimal decision will lead to low gained credit and less effect on the swarm. If a miner is given the wrong swarms, unnecessary overhead will likely to occur for just joining, putting investment, and extracting information from them.

The challenge of probing is the limited and unverified information of the newly discovered swarm. Mindlessly download and participate in the swarm may waste the resource we have. Given a large number of swarms, it is impossible to mine all of them and get good results. A method to filter those swarms is essential in credit mining system. Some swarms might not complete (no one has complete files), only have *webseeds*, or have invalid or duplicate content. Therefore, in credit mining system, we proposed the procedure called *predownloading*. On top of that, to both tackle the tracker unavailability and get thorough swarm information, the mechanism to predict swarm health just by looking at the peers is necessary.

*Predownloading* is a means to download a predefined number of pieces in a

**Algorithm 2** *Predownload* procedures

```
1: function PREDOWNLOAD(infohash, n)
2:     if |download_queue| > 100 then
3:         recall PREDOWNLOAD(infohash, n)
4:     end if
5:     PUSH(download_queue, infohash)
6:     SET_PIECES(infohash, 0, 0)
7:     UNSET_PIECES(infohash, 1, PIECES(infohash))
8:     return CHECK_PREDOWNLOAD(infohash, n)
9: end function

10: function CHECK_PREDOWNLOAD(infohash, n)
11:     peerlist ← GET_PEERS(infohash)
12:     ADD_TO_PEERLIB(peerlist)
13:     if wait long enough and not finished yet then
14:         POP(download_queue, infohash)
15:         return False
16:     end if
17:     if wait long enough and already finished then
18:         TRANSLATE_PEER(GET_PEERLIB())
19:         POP(download_queue, infohash)
20:         return True
21:     end if
22:     if PIECE_DOWNLOADED(infohash) = 1 then
23:         rarest_pieces ← FIND_RARE_PIECE(GET_PEERLIB(), n)
24:         for all rarest_pieces as p do
25:             SET_PIECES(infohash, p, p)
26:         end for
27:     else if PIECE_DOWNLOADED(infohash) ≥ n then
28:         mark infohash as finished
29:     end if
30:     return CHECK_PREDOWNLOAD(infohash, n)
31: end function
```

particular torrent in such a way that it only consumes small portion of resource while trying to get as much information as possible of the swarm. Pseudocode of this procedures shown in Algorithm 2. Assume that there are $n$ pieces that need to be *predownloaded*. To be able to join swarms, the pieces that intended to be downloaded need to be stated. Although any piece will do, we choose only the first piece. Picking the first piece can result a smaller storage allocation. The system will then look for $n-1$ rarest pieces available in the swarm and reject the rest of the pieces. By finding rarest pieces, it can filter more swarm. Moreover, this method is consistent with the BitTorrent core functions. While *predownloading*, the system

actively ask for new peers to tracker or DHT and store it afterwards. The mining activity will continue at the point where *predownloading* finished.

### 4.1.1 Finding rare pieces

The existence of rarest pieces is one of the important aspect for both BitTorrent and credit mining system. Rarest piece is a piece that has minimum number of peer who owned it. Note that rarest pieces is not necessarily only one in the whole swarm. Piece usually identified by its index. Algorithm 3 shows how to find those pieces. This algorithm receives known peer list and the number of desired piece as parameter. It will return the list of rarest pieces in the swarm.

**Algorithm 3** Finding rare pieces procedures

**Require:** $plist$ as list of stored peers
**Require:** $n$ as desired rare pieces amount
1: **function** FIND_RARE_PIECE($plist, n$)
2: $mbit \leftarrow$ POPULATE_PIECE($plist$)
3: **if** MIN($mbit$) = MAX($mbit$) **then**
4: **return** []
5: **end if**
6: $rbit \leftarrow mbit$ where its value equal to MIN($mbit$)
7: remove owned piece from $rbit$
8: $ret \leftarrow$ choose randomly maximum $n$ piece from $rbit$
9: **return** $ret$
10: **end function**

1: **function** POPULATE_PIECE($plist$)
2: $ret \leftarrow []$
3: **for all** $p$ in $plist$ **do**
4: **for all** $pc$ in GET_PIECES($p$) **do**
5: $ret[pc]$++
6: **end for**
7: **end for**
8: **return** $ret$
9: **end function**

In Algorithm 3, there are several cases in which rarest pieces could not be found. Variable $mbit$ contains the number of peer that has a particular piece. Minimum value of $mbit$ means the number of peers that has rarest piece. In line 4, it means that all the pieces has been owned by all peers. In line 6, there is possibility that $mbit$ is empty. For example if there is no information of peers or pieces. Line 7 also can strip $rbit$ to empty if by any chance the system already had all the rarest pieces. Lastly, in line 8 it is possible that the number of rarest piece is less than $n$. In this case, the difference between $n$ and $|ret|$ will be computed in the next turn.

### 4.1.2 Peer translation

For able to be fully decentralized, it is important to not rely on the tracker. Instead, trackless BitTorrent environment described in 2008 which contains DHT protocol [32] and magnet link[33] are recommended. In case of multi-tracker[1], some swarms may have entirely different number of seeder and peers for each of the tracker like shown in figure 4.1. Because of this reason, we alternate the swarm information source by looking directly from the connected peers. We called the procedure to interpret swarm information from both currently and previously connected peers as *peer translation*.

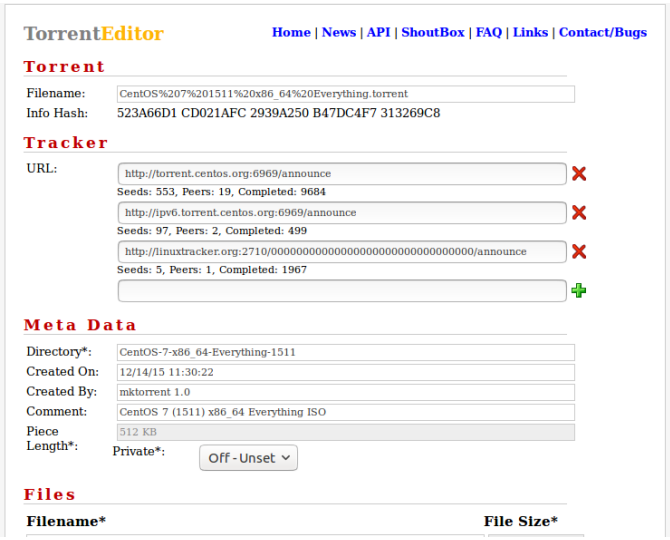



Figure 4.1: Different number of seeder and peers/leecher reported by different trackers.

The *peer translation* receives list of peers as input and returns the predicted number of seeder and downloader. This procedure is shown in Algorithm 4. We categorize a peer is a seeder if it satisfy at least one of three conditions. The conditions are: it only wants to upload right now, it is 80% finished with its download, and it is known to upload more data to us than download. Whichever category gives the highest number will be picked.

In the other hand, we define a peer is a leecher if it satisfies at least one of three conditions : it is currently interested in our pieces, its download process still under 80% and not declaring as “upload only”, and it is known to download more data from us than uploading. Likewise, any category gives the highest number will be picked. Finally, both projected number of seeder an leecher will be returned to the caller.

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

**Algorithm 4** Peer translation algorithm

```
1: function TRANSLATE_PEER(peer_list)
2:     num_seeder ← 0
3:     num_leecher ← 0

4:     upload_only ← |GET_PEER(UPLOAD_ONLY)|
5:     finished ← |GET_PEER(PROGRESS > 0.8)|
6:     unfinish ← |GET_PEER(¬UPLOAD_ONLY & PROGRESS <0.8)|
7:     interested ← |GET_PEER(INTERESTED)|
8:     num_seeder ← MAX(upload_only, finished)
9:     if num_seeder = 0 then
10:        num_seeder ← number of peer which downloaded > uploaded
11:    end if
12:    num_leecher ← MAX(interested, unfinish)
13:    if num_leecher = 0 then
14:        num_leecher ← number of peer which uploaded > downloaded
15:    end if
16:    return num_seeder, num_leecher
17: end function
```

## 4.2 Mining stage

After the probing has been done, the mining stage will take place regularly. Mining stage occurred when the credit mining system already decide which swarm that need to be mined. In a fixed interval, the system evaluates the swarms in *swarm selection* algorithm. It is based on *selection policy* which determines the criteria that need to be considered. We explored the currently implemented policy in 4.2.1. All the miner need to comply to one defined policy. In all policies, swarm with low prospect will be paused, and one with higher prospect replace the old one. Paused swarm may be chosen again in the future. Likewise, the miners may always stick with swarms with very good prospect. As a contribution, we proposed *scoring policy* that will be elaborated more in 4.2.2.

Credit mining system monitors those swarm continuously. The purpose is to look for swarms that are not well-performed in a particular time frame. Under certain requirements, this swarm then will be *stimulated* by optimistically download few rare pieces at one time. Objective of this approach is to eliminate idleness caused by bottleneck of share mode. This approach will be elaborated in 4.2.3.

### 4.2.1 Swarm Selection

The number of swarm for a single source should not be limited. However, with limited resource one may want to limit the number of active swarms at a time. Swarm selection is the process of selecting swarm based on its mining potential.

The selection need to be run periodically because swarm information is constantly changing. At some point, previously selected swarm may not beneficial to mine for both the miners and the community. Eventually, it has to be substituted by another swarm. Swarm selection process is controlled by the applied policy. The policy contains rules to sort which swarm to start and stop.

Although it is possible for user to create their own policy, three policies have been defined on the preliminary work [26]. Those are based on random, swarm age, and seeder ratio. *Random policy* mostly used as the baseline of the experiment. *Swarm Age policy* is claimed better than random policy. It selects the swarm based on its age. New published swarm known to have higher demand than other type of swarm [10]. With a lot of peer in early swarm, it will increase the chance to get more credit. *Seeder Ratio policy* selects the swarm that has lower number of seeder relative to all the peers participated in this swarm. This policy is specialized to help undersupplied swarm.

### 4.2.2 Scoring Policy

In this thesis, we intend to extend those policies so it can be highly customizable and balance credit gaining and helping undersupplied swarm. We propose a policy named *scoring policy* to tackle this issue. Scoring policy was brought up with seeder ratio policy as its base. It offers a method to quantify swarm prospect and to reduce possible identical result from two swarms or more. It can be customized with its *score multiplier*. The idea behind this policy is to give higher score for swarm which has lower seeder ratio, has more peers, lower piece availability, and many activities detected in the swarm. Higher score means getting higher priority to be mined.

Scoring policy consists of several features. First is the seeder ratio. As mentioned before, it is useful to measure whether a swarm is undersupplied or not. This is particularly useful to *help* a swarm. Lower seeder ratio should get higher priority in mining. Second is the number of peer in a swarm. In small swarm, seeder ratio may not relevant because there is not any significant difference between one and another. In a case where ratio of seeder and peers is almost equal for all swarms, it is useful to target large swarms instead of the small one as there are comparably more options to give our pieces to. Third aspect is the file availability of the swarm. In flashcrowd case swarm, there are a significant amount of peers who do not have a large portion of the file. Comparing this to another swarm which most of the leecher almost finished their download, mining flashcrowd swarm will give higher credit and more beneficial to the community altogether. Last element that we considered is the activity of a swarm. It is preferable to mine a swarm whose some of the peers already download or upload any data from or to us. Moreover, one which had faster speed is considered. The worst case happened when there is not any peer interested with the piece we already have. In the other hand, if a peer already interacted with us before, there is a chance we will get chosen again in the future. This feature incentivize swarm which had interacted with the miners

previously.

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

Scoring policy, as shown in Algorithm 5, starts with examining all the swarm registered in a miner. Then it decides the score individually as shown in line 2-5. Variable $rpeer$ is the ratio of a number of peer in this swarm and total number of peers that already known from all the swarm. *Availability* is the number of complete copies of a piece plus the fraction of non-seeder peer that provide a subset of a piece. Its algorithm is explored in Algorithm 6. For *availability* to be 0 it 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 the peer has completed files, *availability* will reach its maximum value which is equal to the number of discovered peer. After the individual score was assigned, the activity of each peer on each of the swarm is calculated. The activity, which identified by total download speed a remote peer to the miner, is sorted increasingly

by its speed. Then, first half of the sorted activity is marked as low activity, and given the lower activity score (line 14). Similarly happens with second half of the list, but with higher activity and higher score (line 16).

**Algorithm 6** Swarm availability

**Require:** $plist$ as list of stored peers
1: $mbit \leftarrow$ POPULATE_PIECE($plist$)
2: $complete_peer \leftarrow |$GET_SEEDER()$|$
3: $min\_peer \leftarrow$ MIN($mbit$)
4: $more\_piece \leftarrow$ number of piece which has more peer than $min\_peer$
5: **return** $complete\_peer + min\_peer + more\_peer/|mbit|$

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 multiplier and activity incentive: $M\_leech$, $M\_pratio$, $M\_avail$, $S\_low$, and $S\_high$. Changing the multiplier will affect its behavior. For example, if a user do not consider number of peers is important, he can set the multiplier for $M\_pratio$ as 0. Setting other parameter except $M\_leech$ as 0 will make scoring policy behave like seeder ratio policy. Similarly, setting other parameter except $M\_avail$ as 0 will prioritize swarm with very low availability. This behavior is similar as if it is applied with swarm age policy in flashcrowd case.

### 4.2.3 Swarm stimulation

As mentioned in Section 3.1, activating *share mode* can result in a bottleneck, especially in the beginning of joining a swarm. *Predownload* a swarm, if handled correctly, can solve this issue in probing stage. By downloading several rarest pieces beforehand, the chance those are going to be uploaded is high. Therefore, positive upload/download ratio is likely, thus, avoiding the *share mode* bottleneck. However, there is not any guarantee when the system is on mining stage.

We introduced a method to stimulate the mining activity on idle swarms. It starts by looking which swarm that already idle for some amount of time. Those swarm are now suspected for in the *stale* state. We then download several rarest pieces on that swarm. We called these pieces *stimulant*. The stimulant can be downloaded only if the share ratio for this swarm is higher than the predefined threshold. This threshold should be lower than `share_mode_target` as in both 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, then 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 rarest piece 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 credit mining system is implemented in Tribler, a python torrent client that was built in Delft University of Technology. Based on this implementation, we come up with the suitable experiment design to answer our research question in previous chapter.

This chapter consists of the elaboration of both 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 a guideline for a new submodule that will be integrated. We comply to that guideline as we will describe later. To evaluate the system, we introduce *gumby* on section 5.2, the experiment runner developed by 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 need to be fulfilled.

## 5.1 Tribler integration

As a proof of concept, 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 credit mining system implemented in Tribler, Tribler stil use WX as GUI (Graphical User Interface) framework. As for the future, Tribler will move its GUI to use Qt starting 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 were 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 credit mining system

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

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

### 5.1.1 Contribution on software engineering

As part of the software engineering process, the credit mining code need to pass several steps before merged into main repository. In Tribler, there are two main branch, 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 is 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 feedbacks. 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 takes almost 3 months to accommodate all the feedbacks and reviews. The contributions of this integration worth more than 4200 added lines and 140 deletions. The code portion is quite balanced with 1425 lines goes to 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 credit mining system. At the time of merging it has passed the necessary code coverage and allowed number of violations. Therefore, it confirms that credit mining system can be deployed in all system that supported by Tribler.

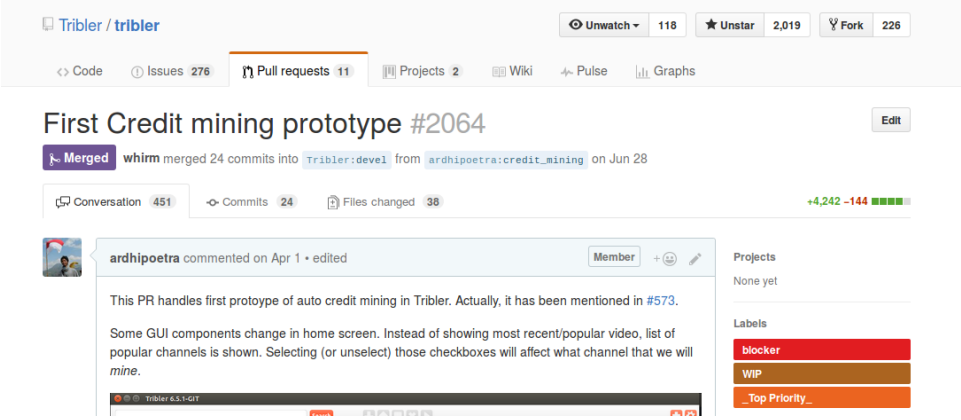



Figure 5.1: Merged pull request on credit mining prototype[3].

To ensure the quality of the main branch, code submitted through pull request is tested by a unit test mechanism. There are two categories in the credit mining unit tests. First one is the tests that check 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 credit mining system will react. The second test is more complex because it emulates the whole

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

credit mining flow for each mining source type. For RSS source, the test deployed local server acting as RSS feeder. As for *channel* source, the test suite prepare the environment by creating both local channel and torrent, inserting torrent metadata into channel, and pushing created channel to *AllChannelCommunity*.

### 5.1.2 Graphical user interface revampment

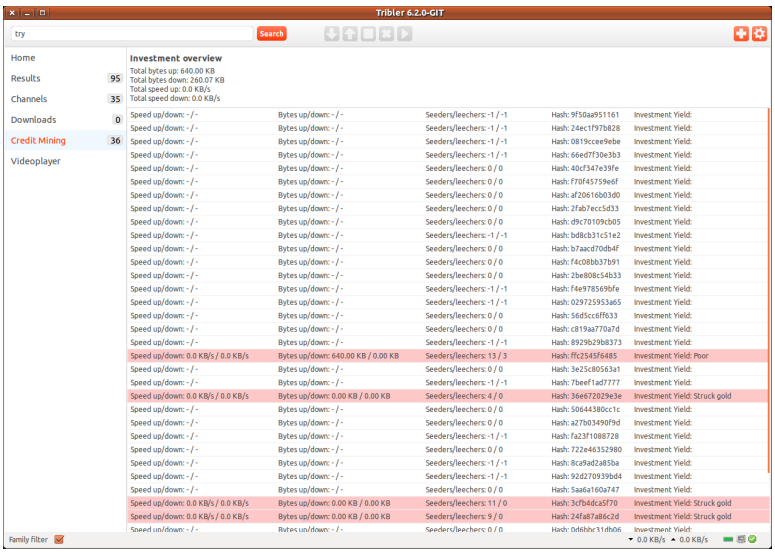
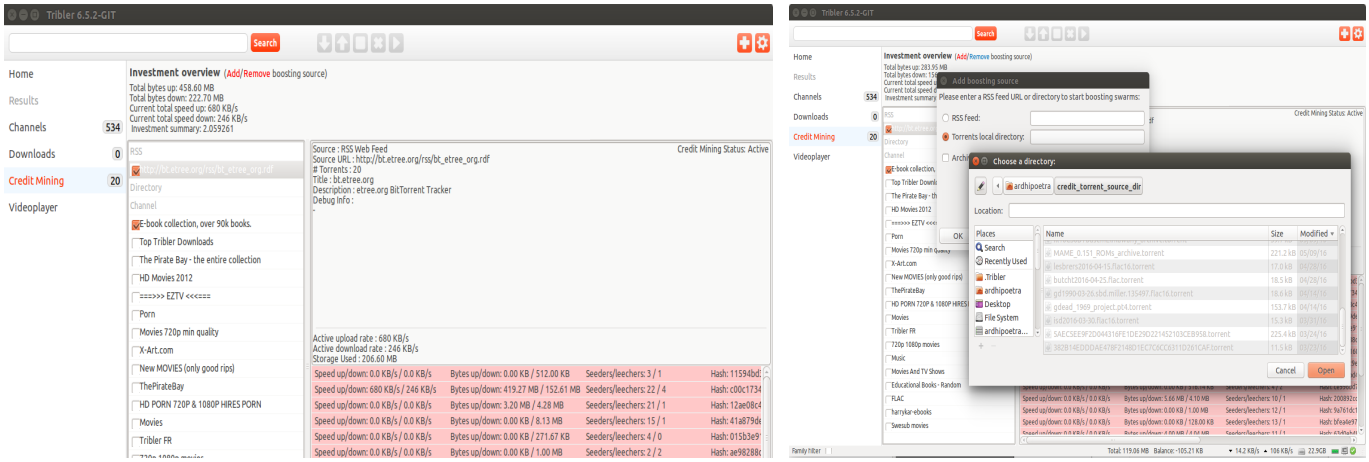



Figure 5.2: The GUI for showing information from prior work [26].

In prior version, it was not possible to add the mining sources except changing from 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, credit mining main screen shown in 5.3a. 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 source can be done by clicking the upper left option. Adding *channel* as a source can be done by put the mark in the check boxes in the channel list. Figure 5.3b shows the example of adding directory source.

As experimental feature, credit mining system is disabled by default in Tribler. Activating credit mining module made the home screen of Tribler changed. We put several channels sorted by its popularity at the home screen as shown in Figure 5.4. The purpose is to encourage user to altruistically mine. To show the channel information, we provide two details. First is the popularity, which shown by number of stars. Second is random swarms that resides within a particular channel. User can simply click which channel he want to mine either in this screen or in the credit mining main screen. Both actions will also be reflected in the other screen.



(a) The investment overview.

(b) The interface of adding mining source.

Figure 5.3: Credit mining main window.

## 5.2 Gumby

*Gumby*[4] is an experiment runner framework for both Tribler and Dispersy. Gumby can run in both local computer and cluster computer to emulate the experiment. In this case, we are able to run gumby in DAS4 as part of our experiment. Gumby run on different scenarios, which consists of many commands, for each experiment. It also uses configuration file to define all the settings needed for running the experiment. Developer can easily specify the number of peer needed, the post process script after running the experiment, the value that needed to be distributed to all of the peers, and many others. The most important part is to code the *client* that written in `python`. In *client* file, one must define how the experiment will run and behave, including the commands interpretation.

Gumby run in 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 and synchronize the *client* on multiple nodes, in case of running gumby in cluster computing. Next, the setup script will be executed. After that, gumby spawns Dispersy and experiment tracker to monitor the nodes in case of error occurred. All of the experiment nodes communicate with server using specified IP address and port. Finally, both local and remote processes are started in parallel. Upon finishing experiment, server will wait for all node instances to exit and disconnect gracefully. Then it will copy the data to predefined directory which can be processed using specified post-experiment script to generate items such as graphs and tables.

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

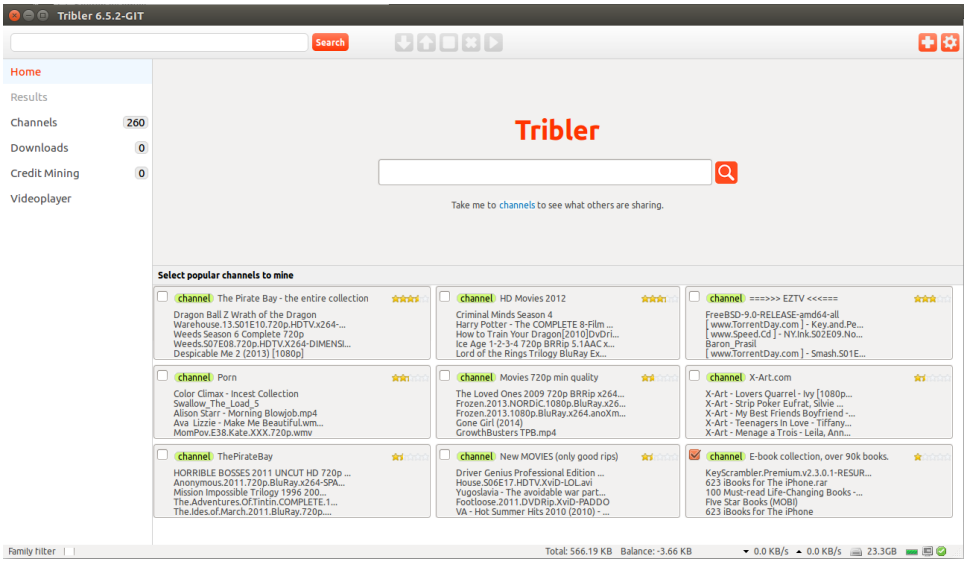



Figure 5.4: The home interface of Tribler with credit mining active.

### 5.2.1 Scenario and Configuration

In gumby, it is not possible to intervene the experiment on the fly. What the developer can do is by specifying commands in scenario file. The scenario notation consists the time of an action and the command itself. Optionally, which node that need 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 in seconds 36, gumby will run command `set_boost_settings` with `boosting.ini.1` as parameter on node number 3.

In contrast to scenario file, gumby configuration file only contains a number of variables that need to be filled. These variables can be accessed from inside the *client*. Figure 5.6 shows the example of configuration format in gumby. There are some necessary variables such as experiment name and `tracker_cmd`. Some of the variables 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 subvariables that recognized by `das4_` precedence. There are also variables that completely optional. In this case, specifying variable `scenario_file` is optional to point which scenario we want to run.

## 5.3 Experimental setup

We now focus on how the credit mining system will be evaluated. In general, there are two aspects we want to address. First one is how credit mining system can gain benefit to its user. This means a user is expected to get considerable amount

```
@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

of credit with small investment. The parameter for this experiment can be derived from how much user actually gain and how it compares to the investment they already had. Second perspective is to find out how the credit mining system can benefit the swarms as a whole. It can be done by monitoring the performance of each of the peers. If each of the peers performance are increasing, then the swarm itself have its capacity increased as well.

### 5.3.1 Experiment conditioning

The experiments were conducted in different scenario and architecture. We hand crafted the environment needed for all the experiments. In this thesis, all of the scenarios are presented in the appendix. Most of the experiment conducted in

closed environment using *channel* as dissemination method. Swarm with dummy files as a content is created. This swarm then inserted into a particular *Channel*. This *channel* can be accessed from all the nodes using Dispersy. In the end, user can only get the metadata of this swarm such as files information, infohash, and other thing typically found in `.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 mining source. The reason is because 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 new published swarm usually has sufficient supply and demand for testing.

We have two different site 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 has dual quad-core processor with 24 GB memory. The interconnection speed between nodes is 1Gbit/s. DAS site is used to run experiments that needs a lot of nodes. It has *libtorrent* version 1.1.1 installed. 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. 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 *channel* and dummy files, generates metadata, pushes it into the *channel*, and seed for the rest of the experiment. Another node can help become a *seeder* for the swarm if necessary. For other node, it can be either download or activate credit mining system. This *channel* can be added to 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 assumption and code modification for the experiment in closed environment. As we limit the download and upload rate, we assume the system know 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 to relatively equal and small. However, it is important to distinguish the features of the policy. The difference between multiplier should not significant such as twice as much as another. $M\_leech$ and $M\_avail$ show the performance shortage in swarms. $M\_leech$ is used in previous work so it has bigger multiplier than $M\_avail$. $M\_pratio$ is a tie-breaker, thus assigned the smallest multiplier.

We then altered the code in three occasions. First is the system will aggressively connect to each other. The IP address and port for each nodes are predetermined

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

prior to launch. We use this information to build full mesh connection topology. The second is, any peer information outside the predetermined range is rejected. The third only applied in *predownload* experiment. In this experiment, we increase the maximum swarm per source to one hundred and set the active swarm to zero. Moreover, after a swarm has been *predownloaded*, instead of send it to miners, we retrieve the information and then delete it afterwards. By this approach, the swarm per source slot will be freed faster and the experiment result still valid.

### Torrent crawler

For *predownload* experiment to succeed, the large number of swarms are needed. Although many swarms can be retrieved from anywhere including illegal source, we want to contribute to the society by providing support for the legal one. We implemented legal torrent crawler that can be accessed in `https://github.com/ardhipoetra/legal-torrent-crawler`. It uses *scrapy*[6] as a scraper for the torrent portal sites. The crawler will access those sites, find any link to `.torrent` file, then download and categorize it. So far, we has implemented crawler for 8 sites as shown in Table 5.1. The crawler is completely unrelated from credit mining system. It can be executed independently. The crawler is executed before *predownload* experiment started. The output of this crawler is `.torrent` files in a single directory which act as an input for the predownload 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 credit mining system implementation in Tribler. First, we will describe several metrics used to identify the results in Section 6.1. Next, we will validate the core components of credit mining system in Section 6.2. Next, we presented the gain comparison between this and prior work to show there is an improvement. After that, two supportive components will be specifically evaluated. First is the *predownload* mechanism and second is mining priority adjustment in case of user downloading activity. Lastly, we will discuss the effect that credit mining system may introduce to the community. All of the experiment and evaluation in this chapter comply to the specifications mentioned in Section 5.3.

## 6.1 Evaluation metrics

Throughout the experiments, we use several metrics to refer the credit mining system evaluation. In order to measure how many credit user already gain, *Net upload gain*[26] is used. Net upload gain is defined as a difference between uploaded and downloaded bytes. To complement this metric, *upload ratio* is also used. Upload ratio is the ratio between uploaded and downloaded bytes. While net upload gain shows the amount of credit a miner retrieved, upload ratio shows how efficient a miner can get the credit after put the investment.

In order to measure whether miner consume the resource 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 maximum upload rate for 70% of its lifetime is using the resource more efficient than another one that only consume the same amount for 50% of its lifetime. Higher number of these metrics means that less resources are wasted.

We use average download speed in a specific period to measure user experience. However, observed download speed may look unstable because of BitTorrent nature. Therefore, as long as the measurement difference is not significant, we con-

clude that user will not experience noticeable performance drops.

## 6.2 Validating the credit mining system

The following experiment is specifically designed to be simple and able to validate all 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 nodes. Our first validation experiment tests the basic credit mining swarm selection algorithm.

The experiment run for three hours. There are 10 swarms, each has the same content size with various number of seeders and fixed number of 5 downloaders. A single credit miner then starts prospecting these swarms and should select the most underseeded swarm. Each swarm's peers are isolated. The credit mining system actively choose at most 3 of those swarm to mine at a time. We set share mode target as 2 instead of the recommended 3 because this experiment only has few peers for each swarm. In this experiment, we use *peer translation* function to interpret the number of seeder and leecher in a swarm.

By its nature, credit mining system only able to control what swarm to choose and how long the mining in one session is. After a swarm has been chosen, *libtorrent* with its *share mode* will do the rest. This causes a slightly different results in each experiment. However, if the swarm selection constantly choose a particular swarm, we expect the trend to be similar.

### 6.2.1 What the policies choose

Figure 6.1 shows how the policy chooses swarm from time to time. At the beginning (timestep 0-2500), the lacks of information causes peer translation function to be less accurate. But 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 current swarm is saturated, the effect is inverted. That

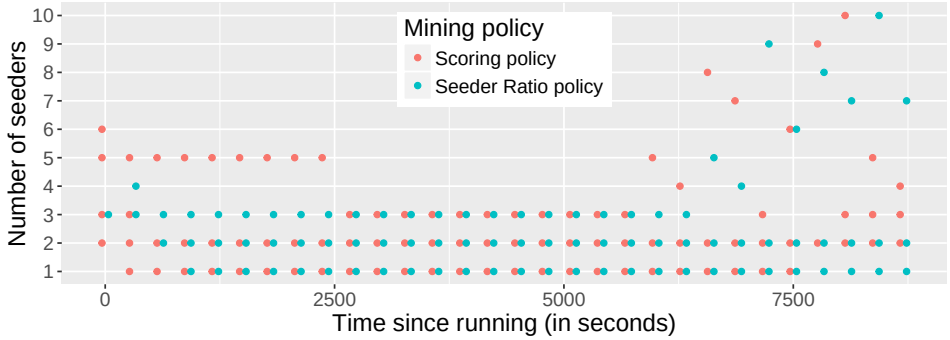
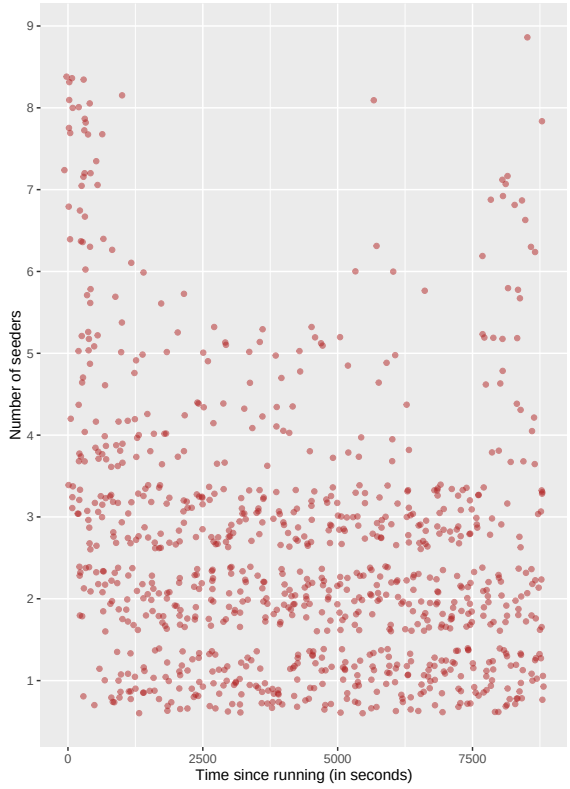



Figure 6.1: Seeder ratio and scoring policy swarm selection.




(a) Scoring ratio swarm selection with 10 miners.


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

Figure 6.2: Swarm selection result on extended experiment.

means, the information of other swarms become very outdated and cause both policy and peer translation function have inaccurate result. The policy have to rejoin other swarm to fetch the latest information. That explains the policy choice dispersion in the last 40 minute (2500 seconds) near the end.

The policy still valid when there are more than one miner in the community. To support this argument, we conducted similar experiment with 10 credit miners with scoring policy. Shown in Figure 6.2a, the miners still choose the first three lowest-seeded swarm for the majority of the experiment. Specifically, it chooses by 79% and 89% for scoring policy and seeder ratio policy, respectively.

To distinguish scoring and seeder ratio policy further, we conducted another experiment. In this setup, the number of seeder and leecher is equal for all the swarms. The only difference is the file size contained in a swarm. Figure 6.2b shows what the policies choose in this kind of community. For each policies, we launched 3 miners. From the result, scoring policy mainly chooses swarm with large file. On the contrary, in seeder ratio, it randomly chooses more than half of all the swarms in the community. A swarm that has large files will have slower

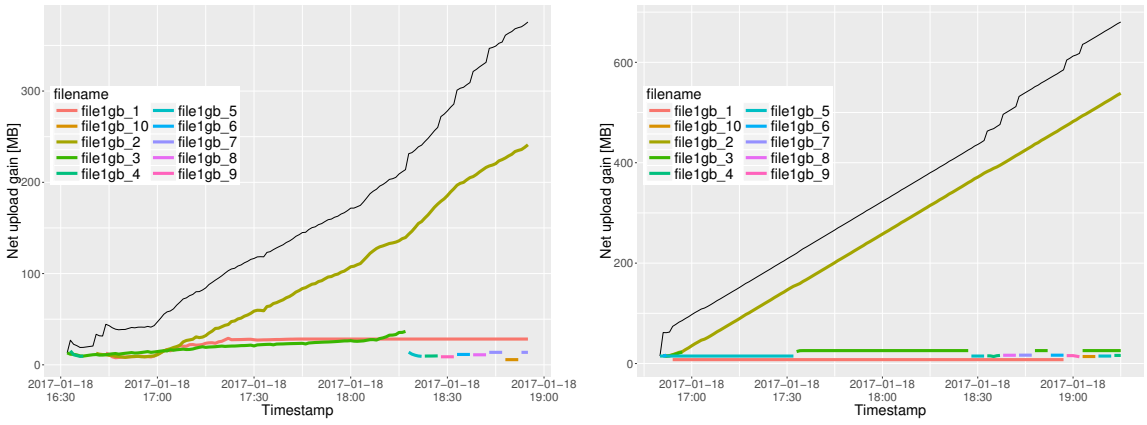
completion rate on its peers compared to one that has small files. Slower completion rate also means there are more rare pieces in a swarm, hence low piece availability. In this experiment, the target swarms are `file5gb`, `file3gb`, and `file2.5gb`. Scoring policy correctly detect the piece shortage problem and address it by choosing swarm with lowest piece availability. From the three targeted swarm, scoring policy chooses 82%. On the other hand, seeder ratio sees all the swarm as equal and the behavior is unpredictable. It chooses 67% of the three targeted swarms.

### 6.2.2 Obtained gain by the selection

The next experiment is conducted to know the actual gain obtained as a result of swarm selection presented in Figure 6.1.

The obtained gain of applying seeder ratio policy is shown in Figure 6.3a. A swarm with 2 seeders (`file1gb_2`) is dominating the result. The rest of the selected swarms are relatively constant most of 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 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 comparison, the average upload ratio is 2.650.

In Figure 6.3b, scoring policy is applied. Unsurprisingly, the trend is similar. This time, the gain is 538 MB, almost twice as of the seeder policy with the same swarm. Although `file1gb_2` returned the highest gain, its upload ratio is not the highest by only 2.99. The highest ratio is returned by `file1gb_7` by 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 is taking more than 80% of the maximum




(a) Seeder ratio policy gain.

(b) Scoring policy gain.

Figure 6.3: Net upload gain of both policies.

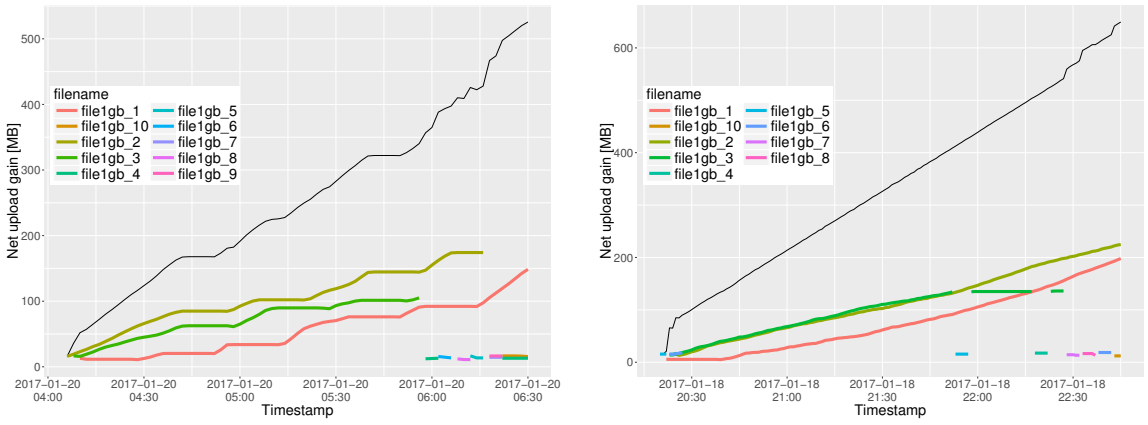
upload rate. By this results, it is clear that the factor that limits credit mining system obtained higher gain is the maximum upload rate.

After we observed those, we come with three conclusions. First, our hypothesis about the similar choice in this particular experiment on both of the policies is stand corrected. Although the gain is significantly different, it was not directly caused by the mining system. Instead, it is part of the BitTorrent protocol that build up the download and upload speed. Second, although the resource may be used to its full capacity like shown in Figure 6.3b, 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 are barely increasing for 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, swarm that gets the highest net upload gain is not necessarily has highest ratio. We believe it is caused by the predownload mechanism that downloads only few rarest pieces and then uploads those pieces to most of the peers. After downloading rarest pieces, some of the swarms are not chosen by policy. Therefore, the ratio keep high because there is not any downloading activity.

### The effect of stimulating swarms

With swarm stimulation, inactive swarms tend to gain more credits when mined. Figure 6.4a shows the case for seeder ratio policy and Figure 6.4b for scoring policy. We specifically focus on swarm `file1gb_1` and `file1gb_3` because they are the most affected swarm by stimulation mechanism. Table 6.1 shows the stimulation effect of those swarms.

Compared to previous results, inactive swarms have increasing performance. From the figure, we can see that there are many bumps that lead to the increasing




(a) Seeder ratio policy gain with stimulation enabled.

(b) Scoring policy gain with stimulation enabled.

Figure 6.4: 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 |

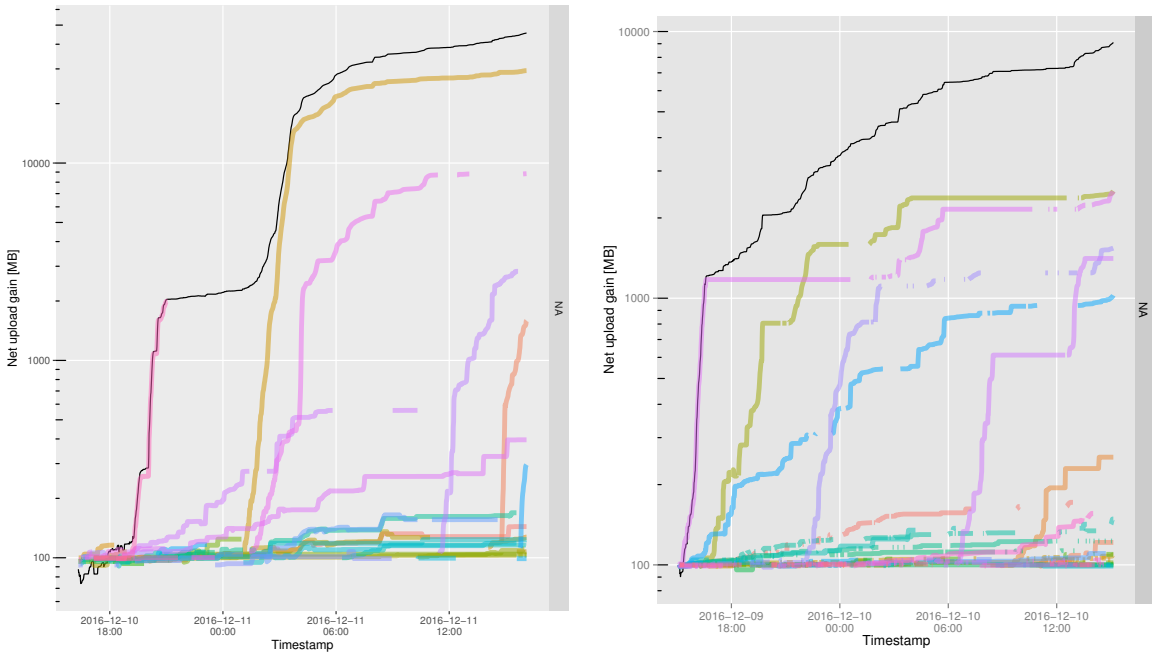
amount of gain. Total stimulant by each swarm is 20, 12, and 12 for file1gb_1, file1gb_2, and file1gb_3, respectively. Also, both upload ratio and net upload gain are significantly increased. On the contrary, in scoring policy, the effect of stimulation is not significant. Although upload gain is increasing, the ratio is similar than when stimulation was disabled. This happened because in previous experiment with scoring policy, most of the resource already in used. Therefore, we conclude that swarm stimulation works best when the resource in miner is not fully used and there is idle swarm in the community.

## 6.3 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 will be run for 24 hours. *Etree.org* will be used as mining source because prior version could neither handle other sources nor use the experiment framework in closed environment. The recommended parameter on prior work is *SeederRatio* as policy, target ratio is 3, and 5 minute swarm interval. The result will be shown in Figure 6.5b.

For the proposed system, we applied scoring policy with stimulation enabled. The other parameter and configuration will be kept identical with prior work's. The result then will be compared to the prior work. Figure 6.5a shows the result of 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 dominated by only few swarms. It means that not all the recent swarms are popular. Popular swarm means more leecher which impact the overall credit gain. For the main difference, we found two aspects that shows new credit mining system is better : policy stability and reducing idleness. In new system, thanks to scoring policy, the miners do not need to unnecessarily switch swarm. It shows that the lines tend to more continuous compared to one in prior work. When the line is invisible, it means that 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 proposed system's experiment, when net upload gained is more than 500 MB, none of the swarm is idle. On the contrary, some swarms in prior work are idle,




(a) 24 hour new experiment.

(b) 24 hour old experiment

Figure 6.5: New vs old experiments (run separately) result on 24 hour (*seederratio* policy)

even when theirs upload gain already high. It can be seen in three horizontal line above the 1000 MB gain.

## 6.4 Predownload hit experiment

This experiment will evaluate the swarms available in the net and its prospect for mining. *Predownloading* will stop when either 1 hour threshold is reached or all the necessary pieces has been downloaded. Swarm with small content size will be automatically discarded. The program uses local directory, where the crawler stored `.torrent` file, as a source. These `.torrent` file is collected through the crawler presented in Section 5.3.2.

In the figure 6.6, it is shown the portion of swarm that has been successfully *predownloaded*. In this experiment, we inserted 1 swarm for every 7 seconds until the maximum amount of active swarm is reached. The number of piece that need to be downloaded is 4. The number of maximum attempts to find rarest piece is 60 times with 30 seconds interval. We divide the failing result in five category : *timeout, zero peers, no information, no downloader*, and *insufficient pieces*.

*Timeout* is the condition where the threshold is reached and we could not finish the download. *Zero peers* happened when the system could not get any peers information. *No information* means that there is not any piece information from known

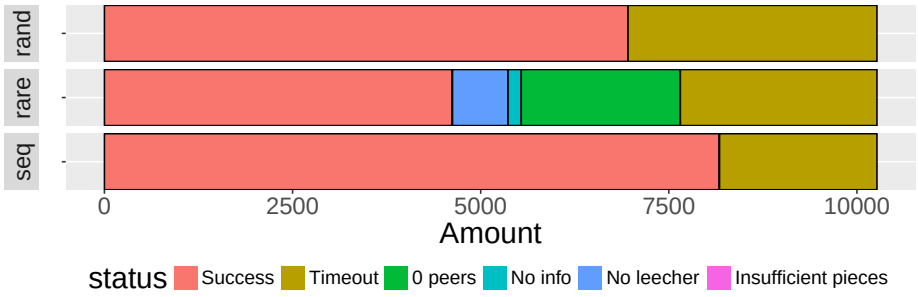



Figure 6.6: Predownload success percentage with three methods.

peers. In *No downloader*, we could not find prospective downloader so that made investment impractical. *Insufficient pieces* means we could not get the necessary number of piece to complete the download.

Out of 13956 observed swarms, 3690 swarm has very small file size. This left us with 10266 swarm with 45% finished, 25% timeout, and 20% with zero peers. This result can be extended in Figure 6.7. Figure 6.7 shows the distribution of time needed to download 4 of the rarest piece. We only consider successfully downloaded swarm, which totaled 4623 swarms. Most of the swarms can be downloaded in less than 2 minutes. This also includes the time needed to compute which the rarest pieces are. The average time is 514 seconds and 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 minute instead of 60 minutes.

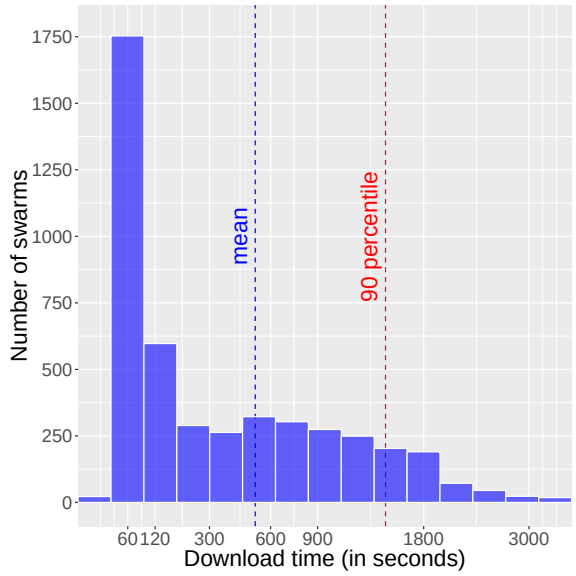



Figure 6.7: Predownload distributed time

We arrived in this conclusion because in less than 30 minute, 90% of successfully *predownloaded* swarm is finished.

In next experiment, we changed the way the system look for pieces in predownloading for comparison. Instead of looking for the rarest one, we implemented two approaches : sequential and random. In sequential mode, the system only will download first 4 pieces. As for random mode, it will look for 4 piece randomly. On contrast with finding the rarest one, these two approaches will immediately decide which pieces they intend to download after first piece has been downloaded. They do not need to collect both peer and swarm information extensively. The result can be seen in on top and bottom part of Figure 6.6 for random and sequential mode, respectively.

From the figure 6.6, it shows that the number of swarm that has successfully downloaded in both methods has significantly increased. Both approaches resulting 0 for attempted failure swarm. This clarify that most of the swarms in this experiment are alive. However, some of them do not have any leecher/downloader at the time of experiment, thus made them inactive. Rarest piece method can filter these swarm as they are not suitable for mining. This method can cut the number of prospected swarm for mining to half of total swarm which is beneficial in order to reduce the complexity for the next phase. Moreover, its behavior is similar to *share mode* in *libtorrent* by only downloading rarest piece that we will be able to upload in the future. In fact, just after predownload finished, the upload ratio may be very high because of the piece rarity as shown in previous experiment.

## 6.5 Sustaining user experience on downloading

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

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

The results in Figure 6.8 shows the download speed on selected peer with and without activating credit mining. The line which shows user download activity is colored as blue and green. Blue line is run on the experiment with credit mining,

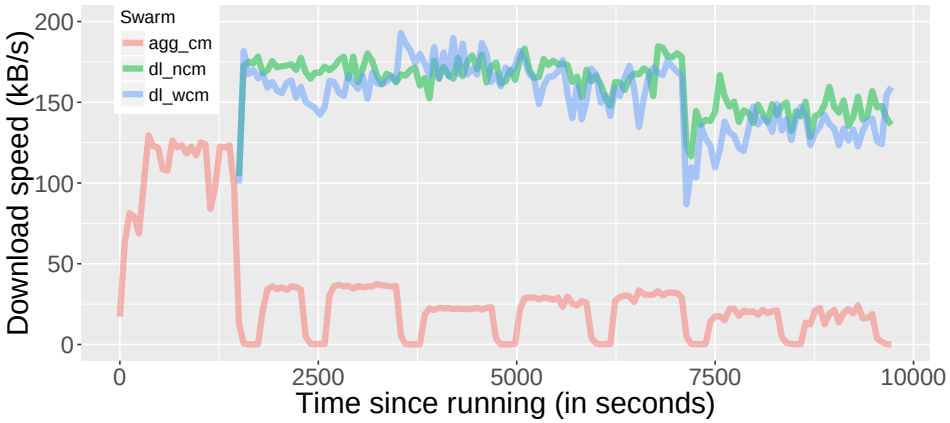



Figure 6.8: Download speed of user download activity.

while green line is on the experiment without it. We find that the results match with our expectation in terms of bandwidth distribution. Before new batch of peers arrived to download, both findings resulting similar speed constantly in 150-170 kB/s. After new batch of peers was coming, the speed can reach up to 150 kB/s for both cases. We conclude that credit mining system have a negligible effect on the overall user experience.

In Figure 6.8, red line is shown as the accumulative download speed for all the swarm mined. At first, the mining speed is quite high by reaching 125 kB/s. After user promptly download a particular swarm, the speed is adjusted to around 45 kB/s. Totaling both mining and downloading activity, it can reach more than 200 kB/s which is 80% of total download bandwidth. Sometime, the mining speed is 0. This is when the observation phase occurred as elaborated in Section 3.2.2. Although not really obvious, the general trend of user download and mining speed is contradictory. This can be observable in between time 2500s until 5000s. When the user download speed increase, credit mining system adjusted its mining speed by allocating lower bandwidth. This allocation method also valid on the opposite, which is when the download speed is decreasing. All of these results shows that credit mining system able to run with leftover bandwidth.

## 6.6 Swarm performance with credit mining

After we confidently get high return gain from the previous results, it is worth to find out what is the effects of credit mining implementation to the community. The experiment run with the same setting as from Section 6.2, except we add more miners instead of one. We also use scoring policy as default and enable both pre-download and stimulation. Other parameters are left default.

Figure 6.9 shows the average download speed of all the member in each of the swarm. In this experiment, no credit mining system are active. Some of the swarms

has reached its maximum download speed such as `file1gb_8`, `file1gb_9`, and `file1gb_10`. We will take this result as base figure for the next experiment.


Figure 6.9: Swarm performance without credit mining

Next, we will introduce credit mining system in the swarms. We start by spawning credit mining system half of the number of downloader, which is 25 node. Those are dedicated miners and all of them started simultaneously. Figure 6.10 shows the average download speed from all the downloaders for each of the swarm. We define the swarm is *covered* by credit mining if it significantly (more than 5%) either get performance increase or drop. In this experiment, the credit mining covers swarm from `1gb_2` to `1gb_5`. Compared to previous experiment without credit mining, the download speed is increased from 18.3% (`1gb_4`) to 29.8% (swarm `1gb_2`). For unaffected swarm, the speed can decrease up to 0.8%.

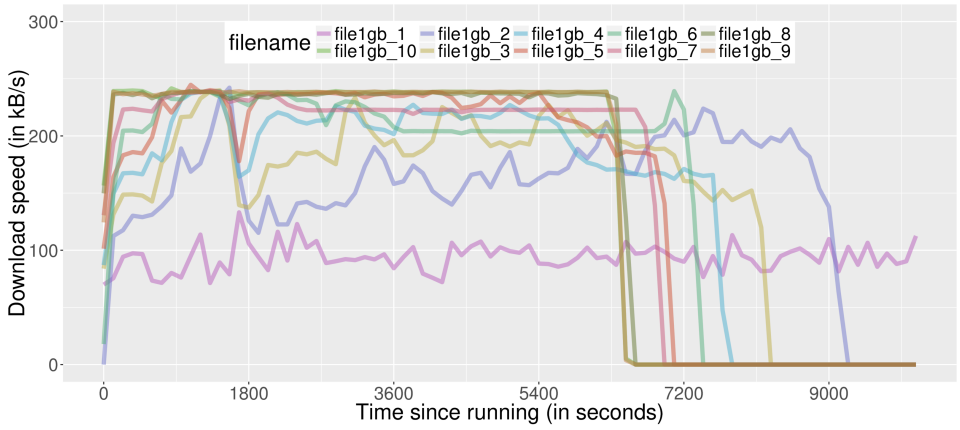



Figure 6.10: Swarm performance with credit mining in the swarm (25 peers)

There are two noticeable drawbacks when introducing credit mining system to the swarm, one of them is that the download speed has become unstable. This

exists because of two reasons. First, because of swarm stimulation, there are higher chance the miners become more active on downloading pieces. Second, when switching to new swarm, miners has incomplete information on rarest pieces. Therefore, they need to download those pieces to be able to mine. Both reasons have a purpose to maximize the upload gain as soon as possible. When miner downloads any piece, it took the seeder's bandwidth and then observed speed for other peers looks like decreasing. These factors combined explains the down and up in the Figure 6.10.

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

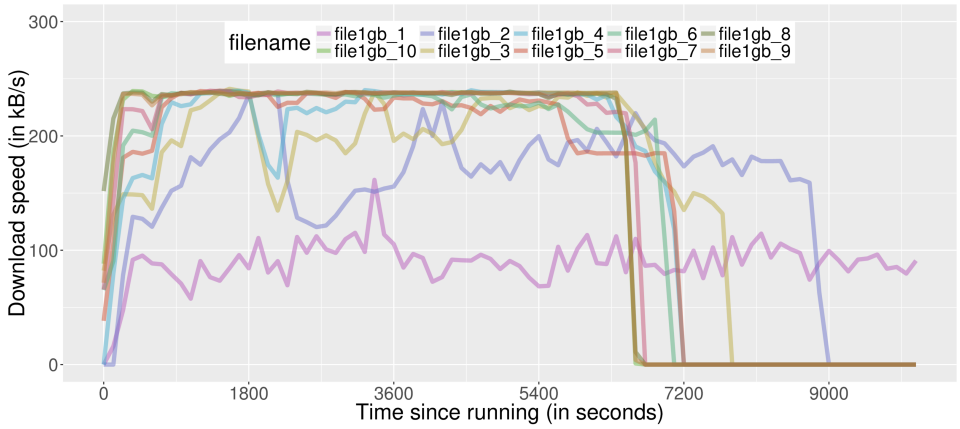
### 6.6.1 Varying the number of credit miners

In this experiment, we will change the number of credit miners in a swarm. First, we double it to 50 nodes. The average download speed observed from peers can be seen in Figure 6.11a. Although the download instability is still there, not only the speed is faster than previous experiment with 25 miners but also it overcomes the second drawback. Compared to previous experiment, this shows that the number of miners relate to the boosting coverage of swarms. With 50 miners, it is enough to cover all the possible swarm in this environment. The exceptional case on swarm with single seeder still hold. 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 covered swarm is 17.98%.

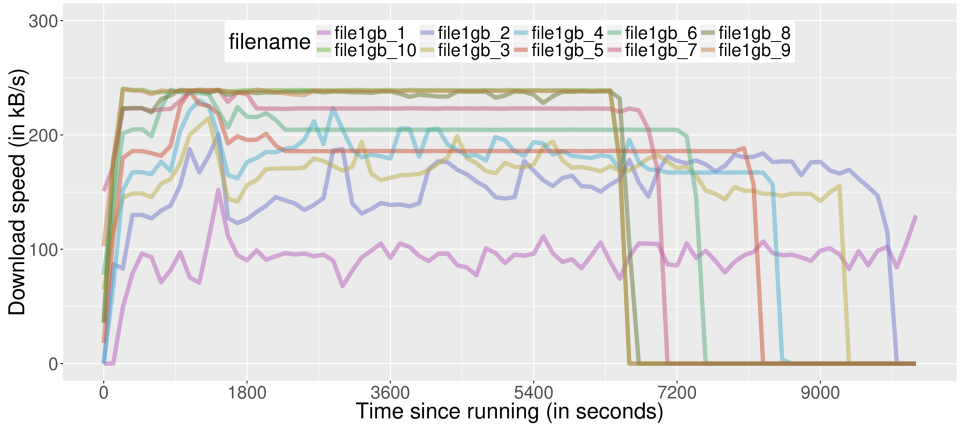
Second, we also conducted an experiment with less number of miners. In this case, the credit miners in the swarm is set to 10 nodes. From Figure 6.11b, it is shown that the coverage is lessened. Now, half of the swarms (from `1gb_5` to `1gb_10`) are not boosted by the miners. The average speed is slightly higher compared to base experiment but lower than 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%.

### 6.6.2 The effect of swarm stimulation

Our next experiment is to understand the effect of swarm stimulation to the community. As shown before (Section 6.2.2), swarm stimulation can increase credit gain




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




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

Figure 6.11: The effect of different number of miners on swarm.

for user. A notable difference from this experiment is that, the average speed instability is gone as can be seen in Figure 6.12. However, as a trade off, the average speed is slightly lower on boosted swarm. Compared to 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`) . However, it is still better than base experiment. Swarm `file1gb_4` improving as much as 28% and swarm `file1gb_2`, despite has the worst decreasing performance compared to one with 25-miners, still improved by 7.25%. Another disadvantages are on the lower-seeded swarm, namely `1gb_1` and `1gb_2`, it has negligible difference with base experiment, especially in latter half of the experiment. Therefore, the coverage of boosted swarm is also reduced.

When swarm stimulation is enabled, it will consume the bandwidth from both seeder and downloader if necessary. After it finished downloading rarest pieces, it immediately returns the bandwidth it consumed back to the community in equal

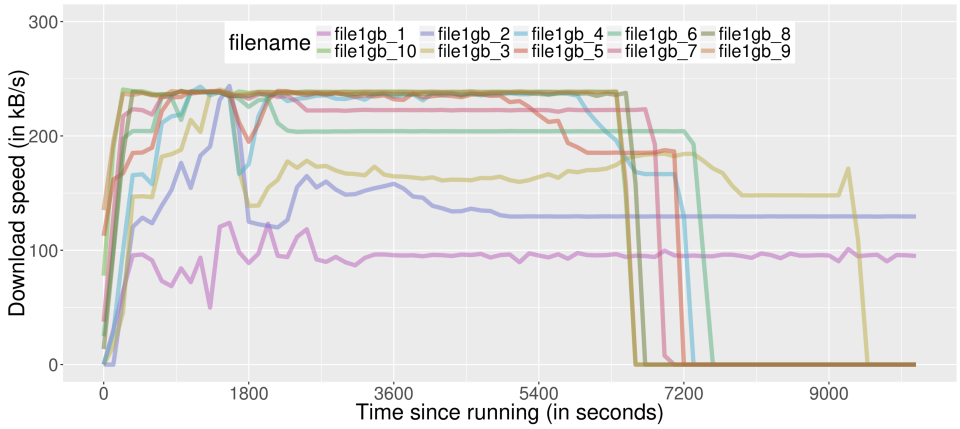



Figure 6.12: Swarm performance with credit mining in the swarm (25 peers) without stimulation.

or higher amount. That explains the bumps which represented instability of the downloaders' download speed. However, we argue that stimulate swarm with very few seeder is counterproductive. Although the stimulation can widely enabled on all the swarm which gives very little drawback, it seems is not suitable for some swarms.

# Chapter 7

# Conclusions and Future Work

## 7.1 Conclusions

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

We then proposed *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 traditional credit system that already widely implemented in both private and public communities. 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 focus on the prospecting and investment algorithm which is the core of credit mining. Two stage of prospecting were presented. Both stages are important. *Probing* stage filters a huge number of swarm while keep resilience by not depending on centralized tracker. *Mining* stage only selects best swarms that have high gain potential. If necessary, it also stops problematic or low potential swarms. We also proposed *scoring* policy, a highly customizable method to quantify swarms into score that can be compared to each other and reduces possible identical result.

Credit mining system now is fully integrated with Tribler. When enabled, it is tailored to not interfere with users activity on downloading. Provided with accessible GUI, user can easily interact with the system to start investing. Before the system is implemented, it has passed several tests and proved to be stable on cross-platform. All the components designed to not hinder user experience while credit mining system is active.

The performance of credit mining system is fulfilled our expectation. All the components doing their tasks properly. Scoring policy successfully selects the most undersupplied swarms and surpasses previous policy accuracy. Moreover, it also stops both saturated and low potential swarm. With stimulating enabled,

most of the time, the system use a large portion (80%) of its resources. The upload/download ratio can reach up to 4.91 with average 3.71 despite the target was only 2.0. After doing the comparison, current system can gain more credit than in prior work. We also showed that credit miners have a good impact to the community as a whole. With the number of miners is half of the peers, it can boost more than half of 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 sees other miners as normal peer. There might be a case where a miner seeds to another miner, which is unnecessary. The key problem of "Co-Investors" is to recognize and utilize the existence of other miner. With recognizing other miners, investment can be more selective. There is less need to boost a swarm if there are already miners in there, for example.

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

Other aspect that still unknown is whether using different policies for different swarms is beneficial. Currently, the policy can not be changed and all swarms need to comply with single policy. Moreover, as we shown in previous chapter, not all swarms suitable to be stimulated. The *partial mining* mechanism is a method to apply different policies and optimizations to different swarm in order to get 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.

# Appendix A

# Experiment scenarios

Appendix body