

Credit Mining: An Incentive and Boosting System in P2P File-sharing Network

Bohao Zhang



Delft University of Technology

Credit Mining: An Incentive and Boosting System in P2P File-sharing Network

Master's Thesis in Embedded Systems

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

Bohao Zhang

3rd August 2018

Author

Bohao Zhang

Title

Credit Mining: An Incentive and Boosting System in P2P File-sharing Network

MSc presentation

30th June 2018

Graduation Committee

Dr. Ir. Johan Pouwelse

Delft University of Technology

TODO GRADUATION COMMITTEE

Delft University of Technology

TODO GRADUATION COMMITTEE

Delft University of Technology

Abstract

From the dawn of BitTorrent technology, free-riding has always been a critical issue restricting the performance and availability of the BitTorrent network. To solve this problem, nearly every private tracker implements some kind of credit system to eliminate free-riders and award the good-behaving users. However, due to these factors, the community size of private trackers is not even close to that of famous public trackers[11]. The average users have to put a lot of effort into maintaining a good credit record, thus make the experience less enjoyable. Moreover, there exists a majority group of light users who do not bother, do not have the capable knowledge or are not aware of the importance to seed for the community. Even worse, the hardcore seeders still need to manually download a lot of contents and waste considerable resources on over-seeded torrents[17].

In this thesis, based on the prototype first proposed by Mihai Capotă[10], we improve, implement and evaluate a boosting subsystem namely Credit Mining inside Tribler[6]. Credit Mining involves a private tracker-like incentive mechanism while remaining good accessibility for every user. This thesis is a piece of the puzzle towards the long-term goal of Tribler, "a trustful blockchain-based token economy to prevent bandwidth free-riding"[5].

Preface

TODO ACKNOWLEDGEMENTS

Bohao Zhang

Delft, The Netherlands
3rd August 2018

Contents

Preface	v
1 Introduction	1
2 PROBLEM DESCRIPTION	3
2.1 Related Works	3
2.1.1 BitTorrent	3
2.1.2 Tribler	5
2.1.3 Trustworthy Micro-economy for Bandwidth Tokens	6
2.2 Free Riding Phenomenon	7
2.2.1 Choking of BitTorrent	8
2.2.2 Private Trackers	9
2.3 Design Goal	10
3 SYSTEM DESIGN	11
3.1 Credit Mining Revisit	11
3.1.1 Data structure	12
3.1.2 Pre-download	12
3.2 Work Flow of the System	13
3.3 Supply of the Swarms	14
3.4 Selection of the Swarms	14
3.4.1 Vitality Policy	15
3.4.2 Switching between Normal Download and Credit Mining	15
4 IMPLEMENTATION	17
4.1 Software Engineering Work	17
4.2 Graphical User Interface	18
4.3 "Hot Swapping" and Garbage Collection	18
5 Experiments and Performance Analysis	21
5.1 Setup of the Test Environment	21
5.2 Functional Validation	22
5.3 Controlled Environment Validation	22
5.4 Real World Experiment	25

6	Advanced experiment and evaluation	29
6.1	One-shot experiment with recent swarms	29
6.2	Three-level promotion policy	30
6.3	Multi-level promotion policy	32
7	Conclusions and Future Work	33
7.1	Conclusions	33
7.2	Future Work	33

Chapter 1

Introduction

Prior to P2P file sharing systems, most contents on the Internet were distributed through centralized servers. Centralized systems suffer a great scalability problem. Since all clients leech data directly from a same server/cluster, requirement of bandwidth, processing power and electricity will increase linearly with the number of clients. Centralized servers are not cost efficient especially when the data-flow to the server is not so constant. The server needs either to prepare enough hardware for peak period, which will cost much but become idle and waste when off-peak, or to lower the service quality during peak period. This did not only deny the possibility of independent content creators to provide high-quality download service, and also made it not very cost efficient for huge companies to do so. It used to be a common sense that more popular the content was, the slower it was to download it.

The popularity of P2P file sharing systems, such as BitTorrent has greatly changed the way contents are distributed across the Internet. In the BitTorrent protocol, centralized servers are only used as trackers to provide peer information to every client. It could also work in a "trackerless" mode, where no centralized server is needed and distributed hash table (DHT) is used instead to provide peer information. After nearly 20 years' development, a large number of individuals and organizations are using BitTorrent to distribute their contents. For example, gaming industry is widely using BitTorrent protocol in game upgraders, such as *Battle.net* by Blizzard Entertainment, *Wargaming.net* by Wargaming, *Eve Online* by CCP Games. Many major open source and free software projects, like *Ubuntu*, provide BitTorrent as an option, to improve availability and reduce load on their own servers. Facebook and Twitter even use BitTorrent to distribute updates onto their their own servers.

However, BitTorrent is a typical tragedy of the commons. To achieve collective optimization, each peer is expected to contribute as much upload bandwidth as possible. This is against the self-interest of the peer itself. Assuming all other peers are acting honestly, one can archive more interest by limiting upload time/bandwidth or only upload to limit amount of other peers which it has interest in. There is no universal credit record in the community. Each peer keeps a credit record of all other peers it is acknowledged locally. "Swarms" are established per content.

Credit can only be tracked within each swarm but not across the whole BitTorrent community. As a result, selfish peers can always archive more interest than honest peers without any negative consequence, before everyone becomes selfish and the whole community corrupts.

Currently, the most efficient way to overcome this problem is by using "private trackers". A private tracker is a membership-based centralized server which is responsible for keeping a credit record for all registered users, and only allows the users with a good credit record to access its tracker. To avoid selfish users registering new accounts to void the negative credit record, private trackers always come with a lot of rules, regulations, and difficulties, which greatly limit the user base of their services.

In this thesis, we are to provide a solution to this problem. Our solution is based on an existing P2P system named Tribler. By enabling the ability to automatically join swarms to gain more credit, our system can provide a private-tracker-like credit system publicly to all Tribler users with convenient user experience and low requirement.

We begin with the problem description in Chapter 2. In this chapter we introduce the related works, the problems we are facing, as well as our design purpose. In Chapter 3, we present the design of Credit Mining in detail. Next, we explain the noticeable details during our implementation in Chapter 4. The experiment and evaluation are discussed in Chapter 5. Finally, we conclude in Chapter 7 and propose the future work of this project.

Chapter 2

PROBLEM DESCRIPTION

Ever since the popularization of P2P file sharing systems, people have always been fighting a war against free-riders. In this chapter we will introduce BitTorrent and Tribler, the two core technologies this thesis is related to, in Section 2.1. Then in Section 2.2 we will introduce what is free-riding, several existing mechanisms against free-riding, as well as their limitations. Then we will propose the design requirement in Section 2.3

2.1 Related Works

2.1.1 BitTorrent

Peer-to-peer(P2P) networks has been used in many application domains since the dawn of Internet. ARPANET, the technical foundation of Internet used a P2P structure itself. P2P finally became popularized among the public since 1999, with the release of Napster, a P2P file sharing Internet service that emphasized sharing digital audio files.

BitTorrent was first released in 2001 by Bram Cohen and soon started to dominate P2P file sharing till today. Unlike some of its precursors like Napster or Gnutella, Bittorrent protocol does not limit to a specified type of service, but provides a universal solution to P2P file sharing.

BitTorrent was not invented as a "fully" distributed system. There are 2 different types of peers in the BitTorrent networks, downloaders and trackers. Trackers centralized servers responsible for helping[12] downloaders find each other. To start a BitTorrent deployment, a static file with the extension *.torrent* containing information about the file, its length, name, and hashing information, and the URL of a tracker, needs to be created by the content provider and disturbed among other downloaders. Since these *.torrent* files usually need to be published on centralized websites, this further lowers the degree of decentralization of BitTorrent networks.

Later on May 2, 2005, Azureus version 2.3.0 was released and added another decentralized layer upon the BitTorrent protocol[1]. This layer uses Distributed

Hash Tables (DHT) to support trackerless peer discovery. This feature was later adopted by most of other BitTorrent clients and improved the decentralization of BitTorrent Networks.

BitTorrent has been widely used and dominated a significant share of the global Internet traffic. Though BitTorrent has suffered a decline in traffic share from the uprising of file storage applications like Google Drive and video services from Netflix and Amazon in recent years, it still holds the largest bandwidth share in uploading in most part of the world, and a noticeable share in downloading. According to research done by SANDVINE, shown in Figure 2.1, 2.2, 2.3 and 2.4, BitTorrent has already been a donating the upstream bandwidth, and has a noticeable share in downstream.

Rank	Upstream		Downstream		Aggregate	
	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%




Figure 2.1: Top 10 applications in Internet usage through fixed access in Asia Pacific in 2015

Rank	Upstream		Downstream		Aggregate	
	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%




Figure 2.2: Top 10 applications in Internet usage through fixed access in Europe in 2015

Rank	Upstream	2016	Downstream		Aggregate	Share
1	Facebook	14.85%	YouTube	20.87%	YouTube	19.16%
2	SSL - OTHER	14.02%	Facebook	13.97%	Facebook	14.07%
3	Google Cloud	9.28%	HTTP - OTHER	9.36%	HTTP - OTHER	9.32%
4	HTTP - OTHER	8.92%	SSL - OTHER	6.85%	SSL - OTHER	7.62%
5	YouTube	5.01%	Instagram	6.66%	Instagram	6.31%
6	Snapchat	4.36%	Snapchat	5.17%	Snapchat	5.09%
7	Instagram	3.35%	Netflix	3.72%	Google Cloud	3.56%
8	BitTorrent	2.16%	iTunes	3.02%	Netflix	3.41%
9	FaceTime	1.97%	Google Cloud	2.87%	iTunes	2.86%
10	iCloud	1.82%	MPEG - OTHER	2.37%	MPEG - OTHER	2.17%
		65.76%		74.87%		73.57%




Figure 2.3: Top 10 applications in Internet usage through fixed access in North America in 2016

Upstream		Downstream		Aggregate	
BitTorrent	30.03%	YouTube	28.48%	YouTube	25.91%
YouTube	9.30%	HTTP - OTHER	11.66%	HTTP - OTHER	11.12%
HTTP - OTHER	7.59%	SSL - OTHER	9.76%	BitTorrent	10.06%
Facebook	6.72%	Netflix	8.31%	SSL - OTHER	9.28%
SSL - OTHER	6.19%	BitTorrent	6.96%	Netflix	7.45%
Ares	5.27%	Facebook	5.10%	Facebook	5.32%
Skype	2.53%	MPEG - OTHER	2.28%	MPEG - OTHER	2.10%
Netflix	1.97%	RTMP	1.79%	RTMP	1.66%
Dropbox	1.16%	Google Market	1.69%	Google Market	1.52%
MPEG - OTHER	0.92%	Flash Video	1.60%	Flash Video	1.46%
	71.69%		77.63%		75.87%




Figure 2.4: Top 10 applications in Internet usage through fixed access in Latin America in 2016

2.1.2 Tribler

Tribler is a decentralized alternative to Youtube based on Bittorrent protocol, an overlay for P2P communication across NAT/firewalls. It is designed to protect user's privacy, be attack-resilient, as well as reward content creators and bandwidth donors directly. The current goal of Tribler is to build a programmable micro-economy without banks, government or any centralized agencies.[2][5][19]

In Tribler, each individual user, differed by its own private key, can create his own channel. The channel can be uploaded with the metadata of Bittorrent swarms, known as torrents. Data is transferred within communities. A Tribler community is similar to a network overlay: users can generate/join/leave a community and

exchange messages over it. For each community, the type of messages that can be sent over it can be defined. Currently there are 6 main types of communities and another noticeable community under development. The roles of these communities are described in Table 2.1.

Channel Name	Description
AllChannel	The communities which all Tribler users automatically join. As the name suggests, metadata of all the channels are transferred through this community.
Channel	The community that represents a single channel. A users will join this community when he subscribes to the corresponding channel. The content of this this channel is transferred through this community.
Market	The community for converting Tribler bandwidth tokens to other Cryptocurrencies.
Search	The community for remote keyword search and torrent collection.
TriblerChain	Community for tracking the reputation of peers using TrustChain, an utilization of Blockchain Technology.
TriblerTunnel	Community that enables anonymity when downloading content from the Tribler network by routing through other Tribler peers using a Tor-like protocol.
SwarmSize	Still under development. It is planned to be used for peers to share the status of Bittorrent swarms in each Channel and estimate the content popularity.

Table 2.1: Function of major channels in Tribler

2.1.3 Trustworthy Micro-economy for Bandwidth Tokens

"A blockchain-based token economy to prevent bandwidth free-riding" is the ongoing high-priority long-term project in Tribler. The basic idea is to create a micro-economy within the Tribler platform for earning, spending and trading bandwidth tokens. This brings together various research topics, including blockchain-powered decentralized market, anonymous downloading and hidden seeding. Trustworthy behavior and participation should be rewarded while cheating should be punished. A basic policy should prevent users from selfishly consuming bandwidth without making any contribution. This directly addresses the tragedy-of-the-commons phenomena.

The initial release is to provide basic primitives to earn, trade and spend tokens. It could be extended with more sophisticated techniques like TrustChain record mixing, multiple identities, a robust reputation mechanism for tunnel selection, global consensus and verifiable public proofs (proof-of-bandwidth/proof-of-relay).[7]

The high-level overview and architecture diagram are shown in figure 2.5 and 2.6. This thesis in position as "token miner" in both diagrams.

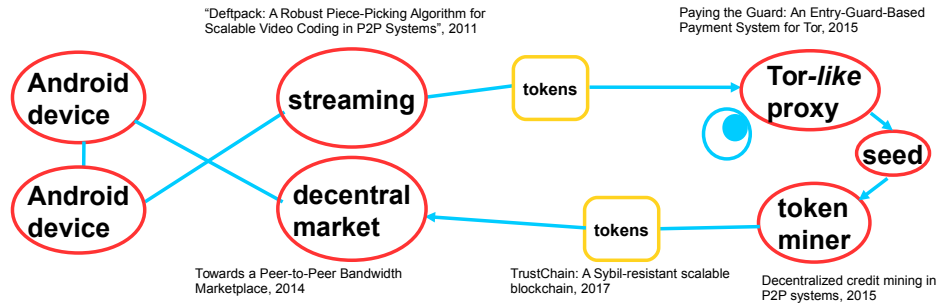


Figure 2.5: High-level overview of trustworthy micro-economy for bandwidth tokens

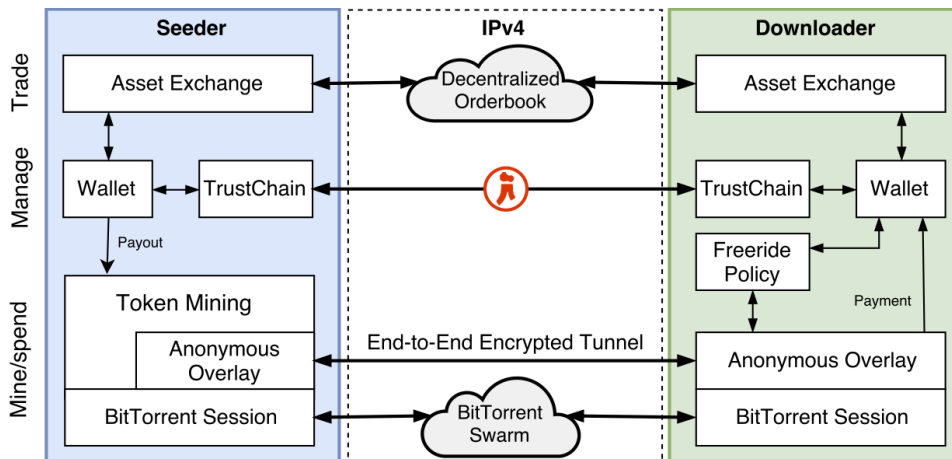


Figure 2.6: Initial architecture diagram of trustworthy micro-economy for bandwidth tokens

2.2 Free Riding Phenomenon

Free riding is defined as the user behave only on its own interest, consuming resources without contribute any or sufficient amount to the community. With only few uploaders available, a peer-to-peer network will become more and more centralized, degrade the performance, reliability and robustness of the network.

The free riding phenomenon is a common challenge among nearly all peer-to-peer file-sharing system. Even before BitTorrent came to being, Gnutella, one of

the most popular P2P applications at the time had already suffered critical free riding problem. In 2000, a group of researchers found out that the that nearly 70% of Gnutella users shared no files, and nearly 50% of all responses were returned by the top 1% of sharing hosts. They argued that free riding led to degradation of the system performance and added vulnerability to the system. These researchers suggested free-riding was even more critical to the system comparing to copyright issue[9]. What is more, in 2005, another group of researchers indicated that 85 percent of peers share no files and that 86 percent share 10 or fewer files, which was significantly worse since then[15].

There are two major methods taken by BitTorrent protocol. Their principle and limitation are described in detail in Section 2.2.1 and 2.2.2.

2.2.1 Choking of BitTorrent

Fully awarded of the free riding phenomenon on Gnutella, BitTorrent file distribution system was implemented with tit-for-tat to pursue Pareto efficiency.

BitTorrent protocol does not do central resource allocation. Trackers are not responsible for allocating resource, but only provide information of peers. Each peer is responsible to archive its own maximum download rate by downloading from whoever they can and upload to peers decided by a tit-for-tat algorithm.

New connections are by default labeled as "choked" and do not get uploaded to. Each BitTorrent peer always unchokes a fixed number, by default 4, of other peers. This decision of which peers to unchoke is based strictly on current download rate. The connections where this client get most benefit get unchoked. The unchoked peers are changed every 20 seconds[12] at Bittorrent's initial launch and later changed to every 10 seconds[8]. There is also a place for "optimistic unchoking". At any time, there is one connection unchoked regardless its upload rate. The optimistically unchoked peer is rotated every 30 seconds. New connections are three times more likely to be selected as the current optimistic unchoke as any other peer in the rotation.

If the client does not receive anything from a certain connection, this connection will be labeled as "snubbed". The client will not upload anything to the "snubbed" peer, unless it is selected as the optimistic unchoke.[8][12]

However, the tit-for-tat algorithm is far from enough for BitTorrent to survive from the free-riders. The Achilles' heel of BitTorrent is that it does not provide a traceable credit record relating to a certain user. A selfish user in one swarm can not be identified in another swarm. Moreover, it is too easy to get a new identity in BitTorrent network and replace the infamous one, making credit less useful even in a single swarm.

There are a lot of ways that one can act dishonestly in BitTorrent network without getting appropriate punishment. Some examples are listed below:

1. A selfish user can immediately turn off the client or remove/stop the download once it is finished to avoid further upload to other peer.

2. Similar to the previous trick, some BitTorrent clients provide the function to remove/stop the download or shutdown the computer automatically after the download is finished to save bandwidth.
3. A selfish user can upload only to the peers he is "interested" in, to gain more credit only from them.
4. A selfish user can upload nothing or little while downloading, and then rejoin the swarm with new identity after choked by other peers.
5. Similar to the previous trick, some BitTorrent clients provide the ability to automatically change identity after being choked. Some of them can create multiple identity to cheat for more bandwidth. This policy would archive local optimization, with a cost of harming the community as a whole.
6. Some clients can create a separate swarm specially for peers with the same client, and only upload to these peers. This will archive local optimization for the users in this client, harming the rest of the community. This can cause "Bad client drives out the good". Users of the "cheating" client will always outperform the "honest" clients, thus attract more users, making the "honest" clients harder and harder to survive.

2.2.2 Private Trackers

Private tracker is an approach to avoid free riding by involving membership. Typically, a "private tracker" includes both an torrent-discovery website and a affiliated BitTorrent tracker. Membership is required to use either of the services. Registration is usually not open. New member needs the "invitation" from an existing member. Usually invitation can only be provided by veteran members. Sometimes it also require a huge amount of credit to purchase such an invitation.

Private trackers usually use strict rules to maintain the good behavior of members. Users gain credit from uploading, and lose credit when downloading. There is also Sharing Ratio Enforcement(SRE), which is a rule for maintaining a certain amount of share ratio. There can be periodic check on SRE. Users who run out of token or do not meet the SRE requirement will be blocked from most, if not all, of the contents.

Due to the pressure of SRE, most users need to seed for long time to improve their share ratio. In research [11], this is called "uploading starvation". Every peer is forced to upload more to maintain a good reputation in the communities. Since the demand of contents is not infinite, the more uploaders there are, the less time/energy efficient for them to get real upload.

New members normally have huge limitation in downloading. This can further avoid selfish users camouflage themselves as new member. However, this also create a huge barrier for the new comers. A fresh member first need to find an innovation from a insider. Then he would need to learn the rules of the private

tracker and how to configure the BitTorrent client. Then since he starts as a new member with no credit, he needs to download and seed some "free"(do not require credit for downloading) content to gain some credits. After gathering enough credit, he can finally start to download the content he originally targeted for.

In conclusion, a private tracker is an closed elite community. It providing high-quality service to law-abiding members. As the cost, the rules are strict and it requires much effort to maintain the credit in the community. Thus the size of the community is limited. Researches [11] have show that private tracker are not comparable to public trackers in number of either torrents or registered users.

2.3 Design Goal

There are two goals driving this thesis. One goal is to provide a efficient way for Tribler users to mine as many "bandwidth tokens" as possible by donating as much bandwidth to the community as possible. The other is to solve the free riding problem in Tribler network.

By combining these two, our design goal of the subsystem in Tribler namely Credit Mining is as follows:

1. Users can archive bandwidth tokens as fast as possible by enabling Credit Mining.
2. Archive Pareto efficiency in improving the health of Tribler swarms while enabled.
3. Require minimal interaction from the user.
4. Have the potential to handle huge amount of swarms in Tribler network.
5. Maintainable for further changes that will occur in Tribler.

Chapter 3

SYSTEM DESIGN

In this thesis, we are introducing a framework namely "Credit Mining", which is an automatic framework to investigate multiple swarms and dynamically join or quit swarms according to their profit potentials. By joining the swarms, a user can contribute his bandwidth to maintain the availability and improve the performance of these swarms, while getting bandwidth token in reward. Credit Mining aims to maximize the tokens one user can earn. With a properly designed economy system, this could also mean maximizing the contribution this user could offer to the whole community. Credit Mining also needs to provide a friendly enough user experience to avoid the high barrier for new users similar to private tracker discussed in Section 2.2.2.

Only in this thesis, since the micro-economy system is yet to be constructed. We simplify the model from the following dimensions:

1. The number of tokens awarded to the user is linearly related to the total amount of his upload to the community.
2. Downloading does not require any token within Credit Mining.
3. There are far fewer miners, aka peers with Credit Mining system in the swarm than the normal peers, thus the competition with other miners are not considered.

Firstly, the decisions and technical debts of Mihai's[10] and Ardhi's[14] works since 2013 are revisited in Section 3.1. Then the design of this system will be introduced in Section 3.2, Section 3.3 and Section 3.4.

3.1 Credit Mining Revisit

According to Github Wiki of Tribler[6], Credit Mining first entered early Beta in 2013. It was later published in 2015 by Mihai[10] and improved by Ardhi in 2017[14]. However, due to this discontinuous develop process, there are noticeable

number of less optimized development decisions and technical debts. In this section they are to be discussed in detail.

3.1.1 Data structure

Data structure is not mentioned in any of previous Credit Mining related works[10][14]. We recovered the design decisions by analyzing the source code and unit test code[3].

The complete list of swarms is stored in a hash table. The key is the info hash of the corresponding swarm and the value is another a extremely messy hash table storing different type of data related to this swarm, including the reference to download handle object, reference to swarm basic definition object, reference to the swarm current state information, another hash table of extracted state information, another hash table of the extracted information of all the known peers in the swarm and the flags of its states in Credit Mining. Moreover, the whole hash table is poorly maintained. There are hash table nested in another hash table which is also nested in another. And no comment or document introduce how this complex hash table is structured. Elements are not initialized unifiedly but added to the table when they are generated.

The state of a swarm in Credit Mining was stored in one of the sub-hash-table mentioned above. When querying for subset of the swarms, such as all the enabled swarm, Credit Mining would traverse through all the elements in the swarm pool. The resource and time consumption is linearly related to the number of swarms, which is not efficient in real life scenario where the swarm pool is huge.

3.1.2 Pre-download

Pre-download, or speculative download, used to be a core feature of Credit Mining. It is the sub-module assists predicting the potential of the swarms. The prediction methods discussed in previous works[10][14] are all based on the characteristics of the swarm that are available to all peers. Since this data is only provided after the client establish the download, this sub-module is designed to download a minimal amount of content from the swarm and fetch the swarm size information.

However, this sub-module does not work well in reality. Firstly, it consumes more than acceptable amount of resource to work. To quick iterate through the whole swarm pool, it needs to rapidly create download handle, start download, periodically check the download progress with a short interval. Once the progress hit a certain threshold, characteristics of this swarm will be recorded and download will be removed. It was implemented without good scheduling. All the swarms are rushed to be added to the this sub-module. It immediately caused lagging and maximum CPU fan noise upon starting Credit Mining experiment on our high-performance test machine.

In Ardhi's improvement[14], there is also a mechanism to download the rarest pieces of the content first to get better acknowledged of the swarm. This operation

consumes much more resources by periodically traverse every piece of the content in every swarm. The complexity is $O(m * n)$, where m and n are the number of pieces in one swarm and the amount of swarm. Considering the number of swarms Credit Mining is expected to handle in real world, this is not effective. Moreover, in libtorrent manual[18], the piece picker is already described as "optimized for quickly finding the rarest pieces". So this part of resource is wasted for no reason. It basically not only repeats what libtorrent is already doing, but also in a inefficient way.

Scheduling the pre-download tasks and limit the maximum swarms being speculated in parallel can lower the CPU usage effectively. However, by observing the time it takes in average to get the characteristics of a swarm is relatively high and highly unpredictable. The evaluation of previous works[10][14] was done on very limited amount of swarms. It will take considerable long time to traverse all the swarms in even a single channel in real world. For example, the most popular channel on Tribler has more than 10 thousand swarms. Even if all the swarms can be eventually speculated after long time, the characteristics of the swarms speculated earlier will not be reliable any more due to the large time difference.

When considering DHT, where Tribler is now moving on, this problem is even more critical. There is no longer a centralized tracker who knows the complete picture of this swarm. This will increase the time to speculate a swarm and make it become even more unpredictable.

According to all the reasons above, We finally decide to drop this feature completely since it is unpractical to be implemented at this stage. There is no easy way to efficiently determine content popularity in community with large number of swarms.

In addition, there is another sub-project on-going in Tribler project trying to solve this problem[4] by disturbing the task to all the peers in the community, then each peer will only need to investigate a limited number of peers and get the rest gossiped by other peers. This could be a powerful aid to Credit Mining after its implementation.

3.2 Work Flow of the System

Credit Mining is fed on the swarms provided by Tribler channels. There are two main loops in the system. One loop periodically checks the selected channels to maintain the swarm pool accordingly, adding or removing swarms. From this swarm pool, the other loop periodically investigates and determines a subset of swarms with the best potential to earn the most tokens for its user. Then Credit Mining examine the difference of this subset and the subset which are currently being mined, replace the swarms currently being mined but having worse actual performance with swarms not being mined but having better potential performance.

The high-level workflow of Credit Mining with Vitality Policy which will be introduced in Section 3.4.1 is as shown in figure 3.1.

3.3 Supply of the Swarms

As a built-in module of Tribler, swarms in Credit Mining is supplied by Tribler channel communities. In Tribler, swarms are organized in channels. All channels can be found in Tribler *AllChannel* community. A channel is identified by a 40-digit hexadecimal string. This digit is also unique to each Tribler user's public key. Every Tribler can create one and only one channel of their own. User can maintain a torrent list in his own channel. The channel is automatically broadcast to other peers in Tribler network along with the all the metadata of the torrents. If a user subscribes to a channel, he will be notified when the channel is edited. Moreover, all the matadata of his subscribed channels will be automatically downloaded and saved into local Tribler database.

By specifying channel identifiers, it is possible for the users to enable a blacklist or whitelist. Credit Mining will automatically continuously try to find and join channels in Tribler *AllChannel* community. Once a channel is joined, Tribler will start to download the metadata of torrents in this channel. Credit Mining gets notified once a piece of metadata is downloaded to the local database. Then a mining handle for this swarm will be built and added to the Credit Mining mining pool.

3.4 Selection of the Swarms

The section mechanism in Credit Mining is called policy. Credit Mining provides a standardized interface to make the system more modular. Limited by the information Credit Mining currently has access to, the policy we implemented might not be the most optimized policy in the future. It is easy to embed other policies in Credit Mining in the future iterations.

Policy is defined as a filter that is given a full table of candidate swarms and swarms that are already being mined on, and returns a set of swarms that should be joined and and a set of swarms should be left. Policy is periodically applied once Credit Mining is enabled.

In previous works, multiple policies are introduced and evaluated in detail. The policy that always select the torrents with higher leecher/seedler ratio is proved to be effective. However as explained before in Section 3.1.2, this policy only works fine in lab environment when the size of swarm pool is not too large to traverse and getting the information of every single swarm is realistic. But as the size of swarm pool grows in real world, the policies fulling relying on the knowledge of the swarms are no longer practical due the reason that Tribler is leak of the function to effectively fetch the content popularity of its swarms. This situation might be change after another project called "swarm size" is completed[4]. The knowledge-based polities might be brought back at that moment.

3.4.1 Vitality Policy

In this thesis, we propose a simplified policy to filter huge number of swarms called Vitality Policy. This policy is inspired by the "optimistic unchoking" mechanism in BitTorrent as described in Section 2.1.1. Likewise, we select a certain amount of swarms to investigate according to their prior performance and several other swarms randomly to constantly keep the vitality of Credit Mining.

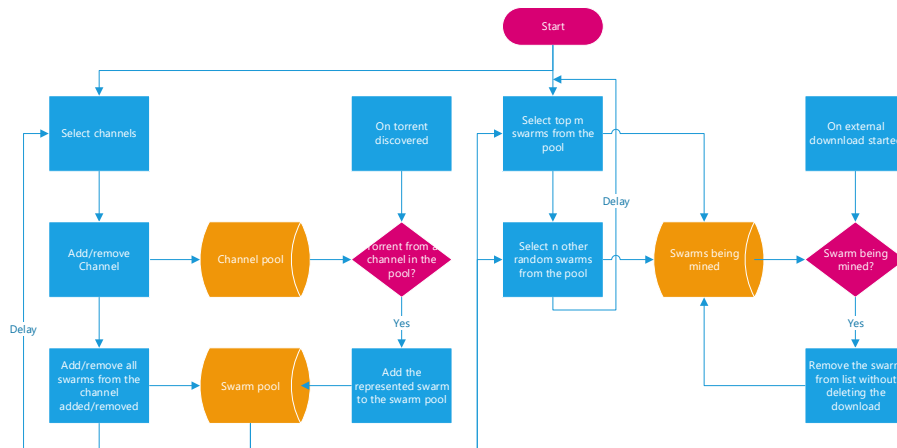


Figure 3.1: Workflow of Credit Mining under Vitality Policy

The workflow of Credit Mining under Vitality Policy is as described in figure 3.1. When the policy is applied for the first time, it randomly pick a number of swarms that equals to the maximum swarms can be mined in parallel in configuration (configured by the user, related to the bandwidth he is willing to share). From then on, every time the policy is applied, it first sort all the swarms according to the upload amount during last interval. A fixed percentage of the least performing swarms will then be put into the "to stop" list. Since there is a high percentage of dead swarms in real world, the swarms whose upload is below a certain threshold are regarded as "dead" and also put into the "to stop" list. The Policy then randomly select the same amount of swarms which are not enabled in this moment to fill in the slots left by the swarms to be stopped. These swarms are put to a "to start" list and return to Credit Mining along with "to stop" list.

3.4.2 Switching between Normal Download and Credit Mining

Since Credit Mining was never formally released in [10][14], two phenomenons was never considered. The user could happen to have interest in the content in a swarm which is currently being mining by Credit Mining. Or Credit Mining happens to select a torrent which is being downloaded by the user. In these cases a *DuplicateDownloadException* will be raised.

To solve these phenomenons, we add extra logic both when selecting swarms in Credit Mining and before starting a download outside Credit Mining in Tribler. When selecting swarms in Credit Mining, instead of select from all the all swarms we filter out all the infohash which are already being downloaded outside Credit Mining in Tribler. Before start a download in Tribler, if Credit Mining is enabled, Tribler will check whether the swarm associated with this infohash is already being mined on in Credit Mining. If so, Tribler will remove its record and configuration from Credit Mining, without stopping it. Then it will be regarded as a normal download task in Tribler. The empty slot left in Credit Mining will be filled with another random swarm in the next periodical investment.

Chapter 4

IMPLEMENTATION

Our implementation started from the branch containing the latest code of Credit Mining. However, due to the lack of documentation, overcomplicated callbacks, massive looping calls and all existing technical debts mentioned in Chapter 3.1, we decide to redo the implementation from scratch.

In this chapter, we introduce some key features of the implementation of Credit Mining in Tribler, python torrent client that was built at the Delft University of Technology.

4.1 Software Engineering Work

Our work started from recovering the code left from previous works. Similar to the technical debts in other subsystems in Tribler mentioned in de Vos, M.A.'s research[13], Credit Mining prototype also contains countless architectural impurity and incomplete testing framework problems. Moreover, there leaks proper documentation or comment of the implementation details, making it hard for maintenance.

When we first took over the code, neither Credit Mining nor its so-called "unit-test" cases was functional. It then became an "pulling oneself up by the bootstraps" problem. Since neither the code nor the test were trust-worthy, when an error was raised, there was no way to determine what is the cause of the error. It could either be caused by the code, the test case, or both.

Since the last work related to Credit Mining[14] gave quite detailed measurement of Credit Mining, we expected to only do minor maintenance to the code. This was later proved to be over-optimistic and we should have rebuild Credit Mining from scratch.

We started from fixing "unit-test" cases. Many of the test cases were not technically unit-tests since they use real torrent files and contact external servers within the test. This make the test result unreliable. Bad connection, timing out, or the availability problem instead of real implementation mistakes could also raise errors in the tests. What is worse, the initial code coverage was less than 50%. A lot of

code are implemented without test cases or even wrong test case.

We rewrote the unit test cases nearly from beginning. We wrote test cases for function by function while recovering, guessing and testing the use of those function. Eventually we reached 100% code coverage rate. All communications to the Internet are patched with mock object to eliminate noise from real world.

Credit Mining code is also debugged alternately at the same time. During this procedure, a great number of mistakes are detected. The mistakes did not only come from architectural impurity. There are several hidden logical mistake that would not stop Credit Mining from working but make it work with wrong logic. This could be the reason that they remained unfixed when we took over.

After both the code and test cases were fixed and working as expected. We detected serious architectural impurity as mentioned in Chapter 3.1 which would take even more time than starting over. We implement the whole module again from scratch and used the code we fixed as a reference.

The following chapters introduce the two other noticeable features we changed during the re-implementation.

4.2 Graphical User Interface

In Ardhi's thesis, a Graphical User Interface(GUI) was implement for Tribler 6.x with wxPython GUI framework. However Tribler has fully moved onward to 7.x version, in which wxPython has been totally abandoned and PyQt 5 is used instead.

Since our target is to simplify user operations, we decide to remove the indicator or channel selection panel existing in previous Credit Mining implementation shown in Figure 4.1. Credit Mining should be selecting channels for the users in the background, and is moving towards the final destination that all channels and their swarms are handled as a whole. Less options is better for users to hand on the system.

In our implementation, we only provide a check box to enable/disable Credit Mining and another check box to show/hide Credit Mining downloads in Tribler setting page, as shown in Figure 4.2.

4.3 "Hot Swapping" and Garbage Collection

Though never mentioned in the previous reports[10][14], Credit Mining was never able to be dynamically turned on and off, or "hot swapped". This is considered as a part of technical debt. According to our observation that Credit Mining is leak of proper unit test and was out-dated seriously from the main branch of Tribler, we speculate that Credit Mining was rushed during the finalization phase. Leak of hot swapping feature is also the result of the rush.

Credit Mining was not hot swapping capable since it is only initialized during the startup of Tribler. When enabling or disabling Credit Mining in setting pages, only configuration is recorded, Credit Mining is not actually initialized or destroyed.

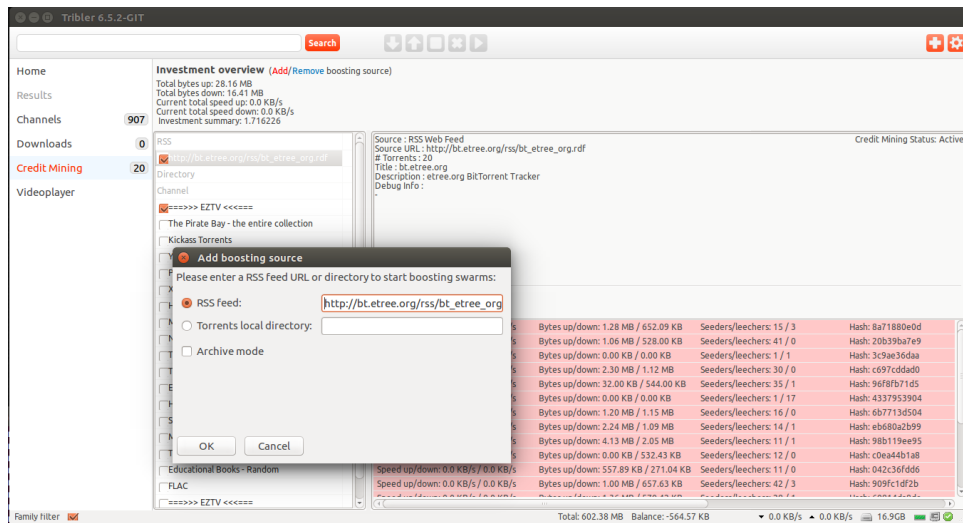


Figure 4.1: Credit Mining GUI in previous implementation[14]

Thus a full restart of the whole Tribler is always required. We modified the Tribler setting page and have it executed extra code when Credit Mining configuration is changed. If Credit Mining is toggled to "enabled", Credit Mining will immediately start to initialize, like the way it is initialized during the startup of Tribler.

If Credit Mining is disabled, the function to shutdown Credit Mining before Tribler is called. However, this function has memory leak problem. Since Credit Mining only needed to be destroyed before shutting down Tribler, it will not cause any trouble if garbage is not recycled completely. But if Credit Mining can be toggled multiple times, the amount of non-collectible garbage will increase and lead to memory leak. Python only deallocate an object when its reference count becomes zero, thus it cannot handle reference cycles[16]. We clean up all the reference in Credit Mining manager, channels and swarms. All reference to Credit Mining objects or from Credit Mining objects are cleared before Credit Mining reference is finally removed from Tribler.

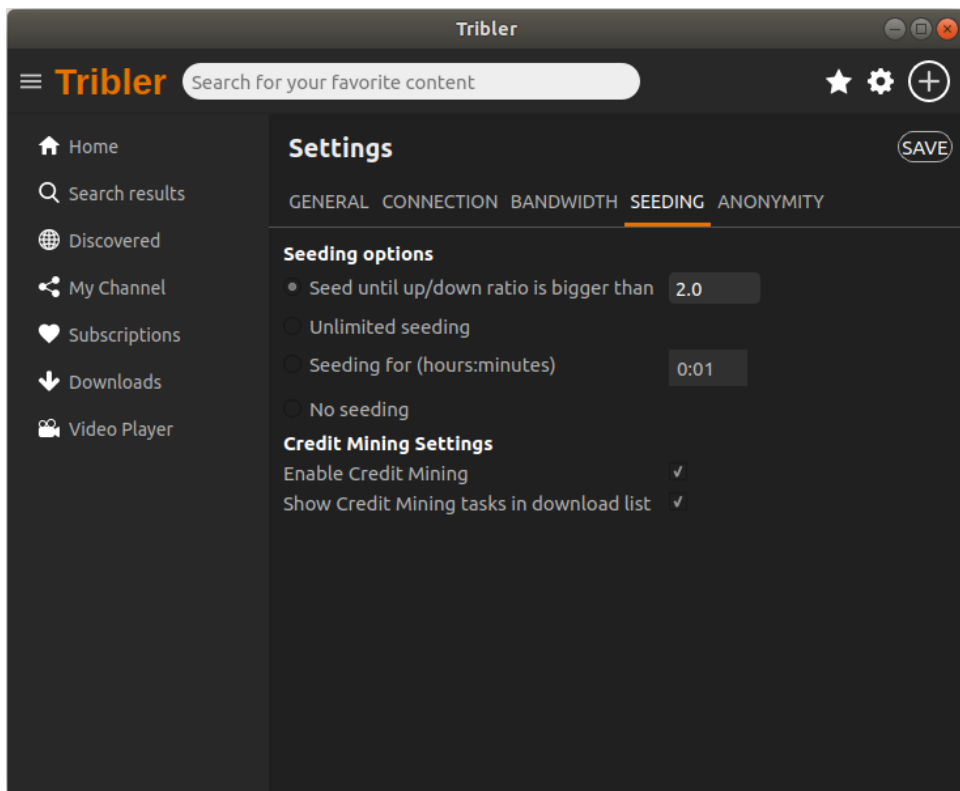


Figure 4.2: Credit Mining GUI in our implementation

Chapter 5

Experiments and Performance Analysis

This chapter will present the experiments and performance analysis of Credit Mining embedded in Tribler. The experiments are implemented step by step, from the basic functional validation in Chapter 5.2, to the validation in a small controllable test environment in Chapter 5.3, to the evaluation on real world torrents in Chapter 5.4.

Tribler uses a fully P2P database to store all the channel information and torrent information, thus makes it not possible to determine the time to fetch certain information from other peers. To eliminate the influence from this problem, instead of testing on remote existing channel, we create a test channel on the test device dumped with all the candidate torrents and run Credit Mining on this channel. From the perspective of Tribler, there is no difference in fetching data from a channel, regardless whether the user has the ownership of this channel. So mining on one's own channel is functionally equivalent to mining on other channels.

5.1 Setup of the Test Environment

In this chapter, tests are done on 2 different devices according to the need. The first and second experiment are done on the local machine with the following specifications:

- CPU: Intel Core i7-7820HK Processor(8M Cache, 2.9GHz to 3.90 GHz)
- RAM: 32GB, DDR4 2400MHz
- SSD: Crucial MX200 M.2 80mm 500GB
- OS: Ubuntu 17.10

The third experiment is done on a remote which with following specifications:

- CPU: Intel Xeon Processor E3-1230 v2 (8M Cache, 3.30 GHz to 3.70 GHz)

- RAM: 16GB
- SSD: 120GB
- Kernel Version: Linux 4.4.83-1-pve #1 SMP PVE 4.4.83-96

5.2 Functional Validation

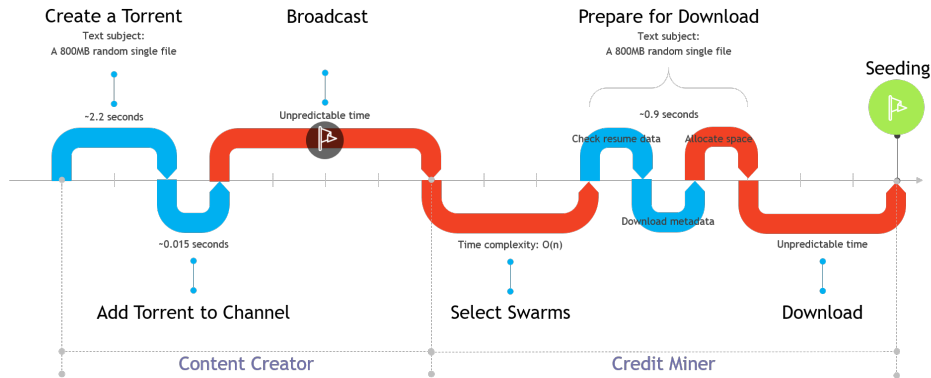


Figure 5.1: Timeline of Credit Mining

The first experiment aims to verify the system is functional. We add single torrent of a 800MB file with is randomly generated to our channel and use another BitTorrent client instance to seed it. This experiment is done several times to get more accurate result.

We generate a 800MB random file to mimic the content distributed through Tribler in real world. We monitor the time consumption from the creating a torrent from the test file from the content creator’s end, to start seeding content and mining bandwidth token on the Credit Miner’s end. The simplified timeline is shown in figure 5.1. We notice that the most time consuming processes are torrent generation. Swarm selection can also consume much time if the size of swarm pool is large enough since it is $O(n)$ time complexity where n is the size of the swarm pool. Since this is a local test, the time for broadcasting is almost 0. But since the torrents are distributed in the normal way though Tribler Channel, the time it reaches the Credit Miner is not predictable.

5.3 Controlled Environment Validation

The second experiment targets to validate the policy mechanism of Credit Mining. The second experiment is taken in a test Tribler channel created locally on the test machine. The channel contains 10 different swarms with various setups. The content of each swarm is a 800MB randomly generated binary file. Peers are are

represented by 6 individual libtorrent sessions running locally. Each session take a different port from 6881 to 6886, and is aware of the existence of other peers. The sessions are configured to communicate to localhost, all external IP addresses are filtered. *allow_multiple_connections_per_ip* option in libtorrent session is switched on to make it possible for the sessions to communicate to each other.

The upload and download speed limit of each torrent in each session are set to 100KB/s and 200KB/s respectively, to counter the problem that local test sessions do not suffer the same bottle neck as real world session. The upload and download speed is almost the read and write speed of hard disk. If no limit is configured, the whole test will be done in seconds, and the result will be fully random, since the "lucky" session which first established connections will consume most bandwidth of CPU and hard disk, choking the rest session and make the test result unreliable. What's more, without speed limitation, as long as there is uploader, the download speed will always be maximum, choking only by the computation capability but not the network, making the number of sessions in a swarm hard has any impact on the test result.

The swarms are assigned with different numbers mock seeders and leechers to reflect different swarm pattern in real world. Swarm 1 and swarm 2 represent the balanced swarms, with 3 seeders and 3 leechers each. Swarm 3 to swarm 5 represents the swarms which are over-seeded, with 3 seeders and 1 leecher each. Swarm 6 to swarm 8 represents swarms under-seeded, and have 1 seeders and 3 leechers each. Swarm 9 and swarm 10 have 1 leecher each, and no seeder, representing dead swarms on the Internet.

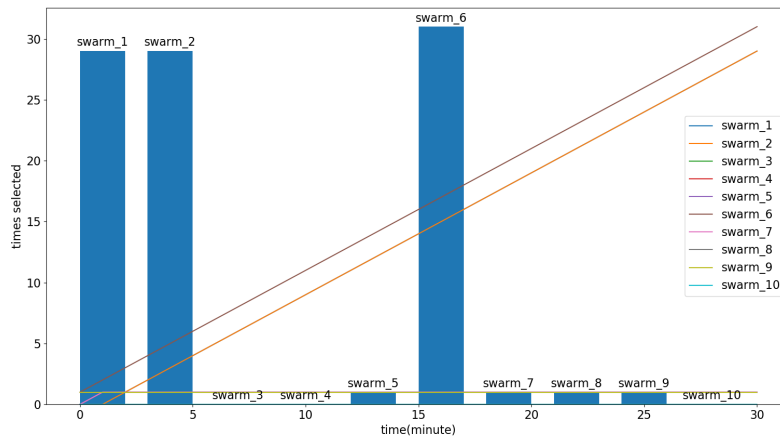


Figure 5.2: Total time being mined of each swarm

Figure 5.2 shows the how Credit Mining select swarms during the 30-minute experiment, exclude the swarms randomly selected, and Figure 5.3 show how it joins swarms, include the swarms randomly selected. Note that this experiment

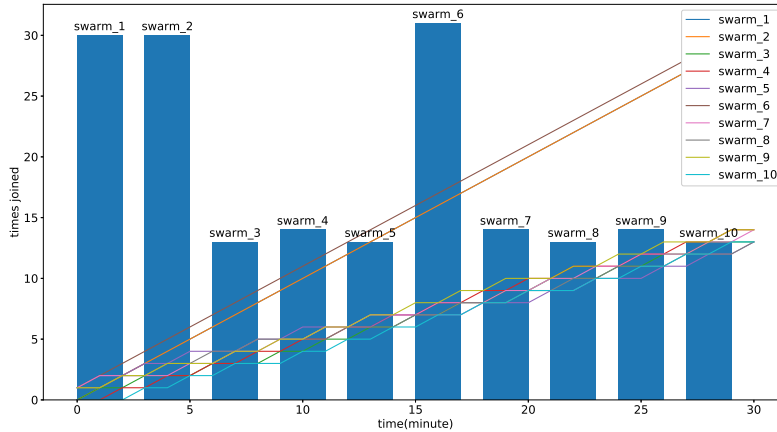


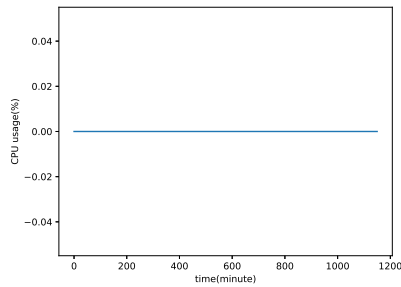
Figure 5.3: Total time being mined of each swarm

is to verify whether the system can differ the performance of each swarm. Since size of swarm pool is relatively small compare to the active swarms within each iteration, Figure 5.3 can not properly show the difference in the joined times for each swarm. With the size of swarm pool enlarges, the proportion of joined times between the swarms with good and bad potential will also be enlarged.

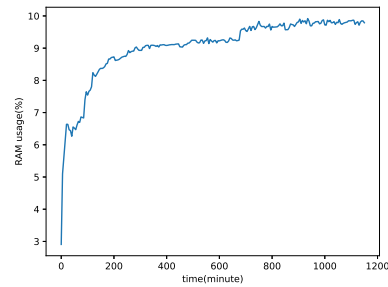
By comparing Figure 5.2 and Figure 5.3, we observe that swarm 5, 6, 7, 8 and 9 are randomly selected in the first iteration of Credit Mining. After this iteration, swarm 8 and 9 was dropped due to their bad performance. Swarm 1, 2 and 3 was then randomly selected to replace them in the second iteration. After this iteration, swarm 1, 2 and 6 are selected to carry on and swarm 3, 5 and 7 are dropped. From then on, swarm 1, 2 and 6 are always selected in every iteration till the end, and other swarms are randomly picked to compare but dropped at the end of iteration since they have lower performance than swarm 1, 2 and 6.

The result in Figure 5.2 was against our assumption. In Mihai's work[10], seeder/leechers ratio is proved to be an efficient approach to rate a swarm. Basing on this conclusion, we were expecting Credit Mining would constantly select swarm 6, 7 and 8 after few intervals. The result turned out to be that Credit Mining constantly select swarm 1 and 2 instead. In this experiment, it also select swarm 6, which is an interesting phenomenon. We assume that swarm 1 and 2 have the best performance and swarm 6, 7 and 8 are next to them. The selection of swarm 6 is actually coincident. Due to the fact that we have very limited amount of seeder and leechers in total and swarm 6, 7 and 8 are underseeded. If Credit Mining cannot join the swarm before all other leecher peer establish the connection with the seeder, it cannot have enough share of bandwidth from the seeder, thus makes it have less content to upload.

To verify the assumption, We then repeated this experiment several times. The



(a) CPU usage during the experiment



(b) RAM usage during the experiment

Figure 5.4: System resource usage during the experiment

result was similar to the experiment shown in Figure 5.2 and Figure 5.3. Swarm 1 and swarm 2 are always be the major choice of Credit Mining once investigated. There is always a third swarm being selected just like swarm 6 in Figure 5.2. Through our experiments, this swarm could be either swarm 6, swarm 7 or swarm 8.

This basically proved our assumption that swarm 1 and 2 actually have better performance than swarm 6, 7 and 8, which is against Mihai's conclusion.

5.4 Real World Experiment

Real World Experiment focus to analysis how much resources Credit Mining will use in real world scenario and its performance. This experiment is done on the test channel with all 25 torrents from a real tracker from the Internet.

The experiment was done for roughly 20 hours. The system resources usage is displayed in Figure 5.4. Note that Since the since the CPU usage rounds to 0.1%, it appears like Credit Mining along with Tribler does not consume much CPU generally. Memory usage is always under 100MB. However, We observed that the There are several unusual slight stepping in RAM usage during the experiment. This cannot be explained from Credit Mining's mechanism or the experimenter's log file. It might be caused by some undetected memory leaking problem in Tribler core.

Performance-wise, the total payload upload and total download curve is shown in figure 5.5. Credit Mining seeded 300GB during the whole experiment. The upload speed keeps increasing during the first hours Credit Mining is turned on. This is due to the feature that Credit Mining start with random swarm, and then replace them with better performing ones. From figure 5.6 speed slowed down after seeding for 7 hours. We assume the reason is that the demand from these swarms are mostly fulfilled already. If Credit Mining is running with a larger and dynamic mining pool, the speed will increase after Credit Mining find the better performing ones to replace the mostly fulfilled ones.

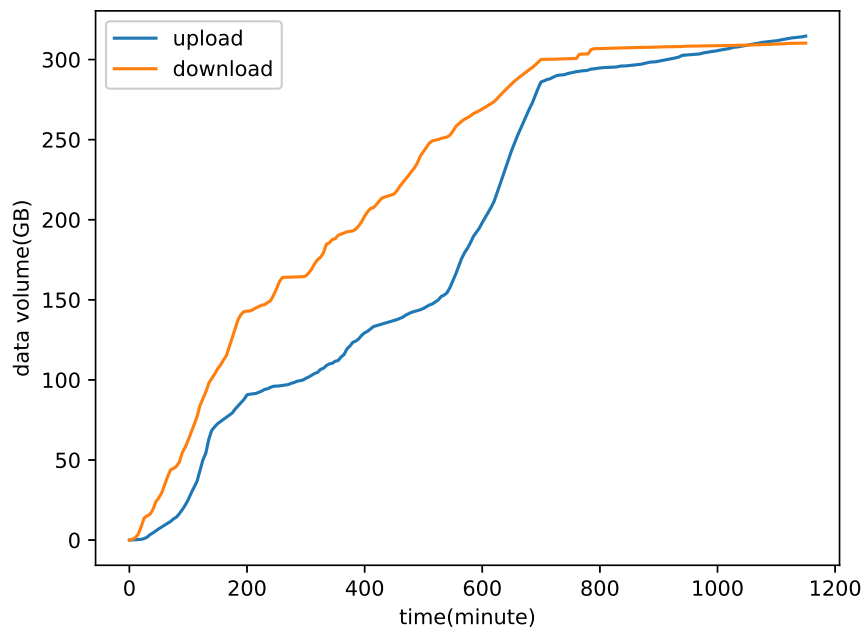


Figure 5.5: The payload upload and download performance with real world swarms

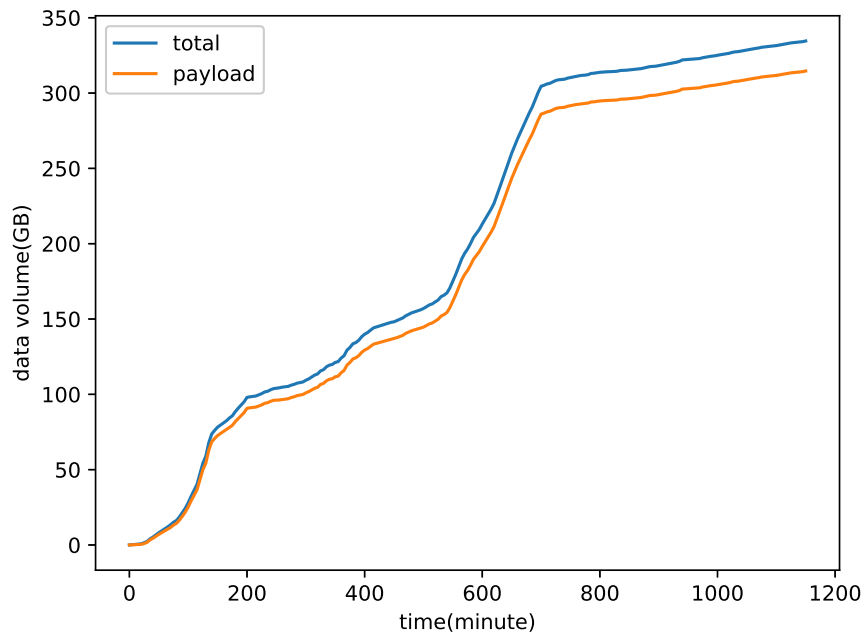


Figure 5.6: The payload upload performance comparing to total upload with real world swarms

Chapter 6

Advanced experiment and evaluation

From the various experiments, we are sadly aware that without any prior knowledge to the swarms, the policy we implemented do not provide very positive yield. The policy can distinguish the most profitable swarms but with a high cost of total download. In this chapter we are applying the knowledge and experience we learn from these initial experiments and modifying the policy to further improve the performance.

The key insights and experimental results are presented in the following sections, each improved from another and growing more complex and efficient.

6.1 One-shot experiment with recent swarms

This experiment bears the concern that the popular torrents are mostly over-seeded and the rapid random investment is one of the reasons that causes the negative yield. We change our swarm candidate from popular swarms to 30 most recent swarms. We also remove the periodical random investment feature and use a one-shot investment instead.

To further avoid wasting download stream on the over-seeded or dead swarms, we assign an initial limit of 25 Megabytes to every swarm. Once the limit is reached, the download task will be switch to upload mode. After 3 minutes we rank all the swarms by their total upload till then. The download limit of the top 3 swarms are removed for further investment.

The result is shown in Figure 6.1 and Figure 6.2. Figure 6.1 shows the Number of torrents in each category over time. "Torrents added" counts the torrents which are existing in Libtorrent. "Torrents active" counts the torrent which are has non-0 total download. "Torrent investigated" counts the torrents with no less than 25 Megabytes download, which then become candidates to be unlimited. From the figure we can observe that there is a slightly difference in "Torrents added" and "Torrents active". The torrents we added are the most recent ones by magnet link,

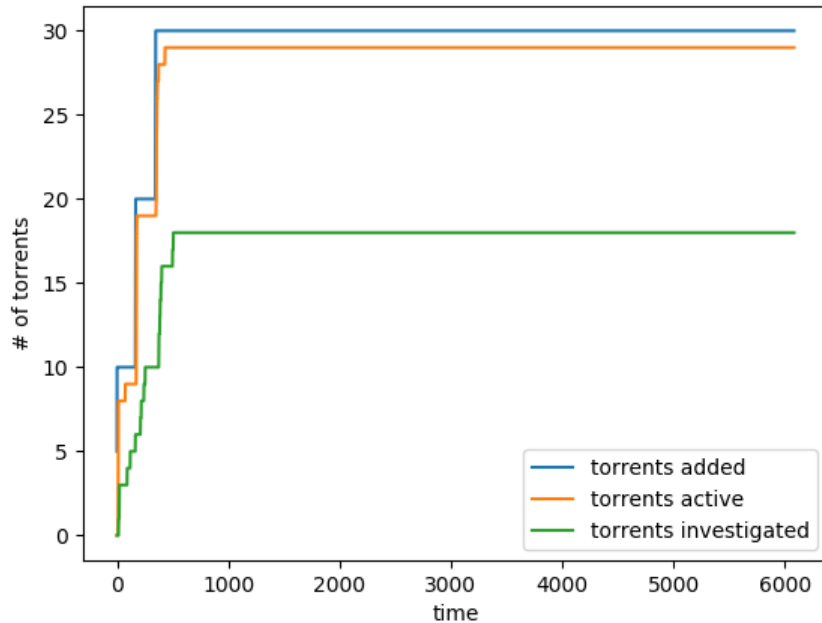


Figure 6.1: Number of torrents in each category over time

which also means they are fairly random. There could be torrents that are not reachable or torrents of 0 health. We also observe that the percentage of "torrents investigated", which are torrents with no less than 25 Megabytes download, is also quite low. This shows that there is a high percentage of torrents on the Internet are dead. These torrents are not likely to benefit anything to Credit Mining.

Figure 6.2 shows the total upload and download over time. There is a clear pattern that the download speed is higher than upload speed before all downloads either complete or reach their limit. After that the upload can soon catch up with and exceed the download.

In this policy we only give each swarm a single chance to be unlimited. It might fail to bring out the profitable swarms if they perform not so good in the first minutes. Also if a torrent is outstanding in the first minutes, it will be fully unlimited, regardless its later performance. In our repeated experiences, the unlimited swarms are not always providing positive yield.

To reduce these problems, we proposed a "three-level promotion" policy.

6.2 Three-level promotion policy

In this policy, instead of directly remove all the limit, we "promote" a torrent to the next limit if it outperforms others in a certain period of time. The limits we

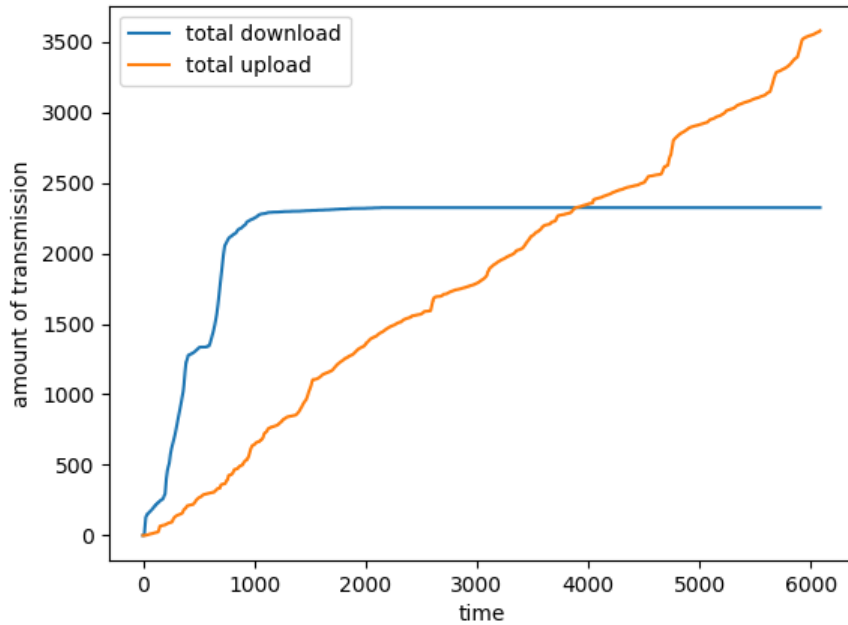


Figure 6.2: Total data transmission over time

set in this experiment is 25 Megabytes, 250 Megabytes and 1 Gigabyte. Torrents with these download limits are namely non-promotions, promotions and super-promotions. Considering the limited disk space, we set the maximum number of all torrents to 1000, torrents as least promoted to 100 and torrents super-promoted to 25. Torrents from non-promotion and promotion category are ranked respectively by upload very five minutes. The top one from each category is promoted to the next phase. When ranking, only the torrents with more than 1 Megabyte is taken into consideration to avoid wasting resource when all the torrents available at that time is dead or over-seeded. We also add torrent periodically instead of entirely to maintain the freshness of the torrents.

The results are shown in Figure !!!!!. It is visible that the download and upload curves show a very similar pattern to the result from last experiment. The download curve is steeper than upload until the downloads are mostly completed or limited. Compared to the experiment result in the last section, the total upload and download are great increased both in speed and in total amount. Though there is no dramatic increase in upload/download ratio, the net upload gain(upload-download) is increased dramatically, which is exactly what we expect to archive when tokens we gain are measured in the total bandwidth contribution rather than the ratio.

6.3 Multi-level promotion policy

To further explore the possibility of boosting the efficiency of every byte we download and store in the disk, we attempt to further subdivide the levels and set more aggressive requirement to promotion. In this experiment, consider the relatively high percentage of dead and over-seeded torrents, the minimal limit is reduce to 5 Megabytes. Download limit for torrents in each phase are respectively 5, 8, 12, 18, 27, 40, 60, 90, 135 and 200 Megabytes, 11 layers in total counter the torrents failed to download the initial 5 Megabytes. To further avoid the promotion of a unprofitable torrent, the basic requirement to promote a torrent is reaching 1.0 upload/download ratio. In another word, one torrent must earn the tokens spent on investigating itself back until it is qualified to consume more resource and be further investigated.

The

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis is to design and implement a module in Tribler to improve the availability of the contents in the Tribler community, and provide a solution to token mining in Tribler.

7.2 Future Work

Currently, the most optimized policy is yet to be find. The current policy is theoretically guaranteed to find the torrents with the best potentials. But the cost is too high. It must be running in the community where there are a reasonable percentage of active under-seeded swarms. Otherwise, the current policy will waste too much time traversing across the swarms before it finds the demanded one. Also, if CMS is restarted, it will also be time consuming to get it back to the best working condition.

One possible solution is by developing a model based on the accessible swarm information. Our plan was to get a database of download performance and swarm information, and train the model on this database. However, there is no such database known to be existing, and we do not have enough resource and time to construct such a database in this thesis. Moreover, even after we got a model for estimation, we still need a decentralized swarm information sharing system[4] to make it work. Thus we decide to leave this to the future, after the completion of this sub-system.

Another problem is that CMS currently regards all other peers in every swarm as normal peers. There is situation that there are multiple users in CMS investigating the same swarm. In that case, transferring data between the miners are not as beneficial to the swarm. Also if too many miners investigating the same swarm, it might have a possibility to become a trap for new miners. This problem needs to be further researched. The solution might be an approach for the miners to announce themselves to other miners and cooperate based on the swarm information.

The current Tribler system does not provide the resilience against Sybil attack. Selfish users can always create as many fake account as needed to clear up any negative credit record. In this way they can still escape the punishment of free-riding. Our plan is to establish a naive proof-of-contribution mechanism only targeting the new users. Upon a new user with limited credit record join the community, he will be given very limited access speed compared to a honest user. Only after a certain amount of contribution is done to the community, the new user can be issued a normal. This will make is less efficient to selfishly fake accounts to leech the community than to honestly mine in the community to earn more tokens.

The last problem is that current policy only focuses on the total upload amount of the swarms. The miners should be also be rewarded for keeping the availability of the dying swarms. This can be archived by changing the sorting key function once such rewarding function is specified.

Bibliography

- [1] Azureus2 changelog - vuzewiki. http://wiki.vuze.com/w/Azureus2_changelog#2.3.0.0_-_May_2.2C_2005. (Accessed on 05/13/2018).
- [2] blockchain-regulated markets. <https://github.com/Tribler/tribler/issues/2559>. (Accessed on 05/07/2018).
- [3] Github - tribler/tribler: Privacy enhanced bittorrent client with p2p content discovery. <https://github.com/Tribler/tribler>. (Accessed on 02/25/2018).
- [4] Swarm size community: content popularity issue #2783 tribler/tribler. <https://github.com/Tribler/tribler/issues/2783>. (Accessed on 02/24/2018).
- [5] Towards global consensus on trust. <https://github.com/Tribler/tribler/issues/3357>. (Accessed on 05/07/2018).
- [6] Tribler: an attack-resilient micro-economy for media. <https://github.com/Tribler/tribler/wiki>. (Accessed on 05/07/2018).
- [7] A trustful blockchain-based token economy to prevent bandwidth free-riding. <https://github.com/Tribler/tribler/issues/3337>. (Accessed on 06/09/2018).
- [8] The bittorrent protocol specification. http://www.bittorrent.org/beps/bep_0003.html, February 2017. (Accessed on 05/16/2018).
- [9] Eytan Adar and Bernardo Huberman. Free riding on gnutella. 5, 04 2001.
- [10] M. Capot , J. Pouwelse, and D. Epema. Decentralized credit mining in p2p systems. In *2015 IFIP Networking Conference (IFIP Networking)*, pages 1–9, May 2015.
- [11] X. Chen, Y. Jiang, and X. Chu. Measurements, analysis and modeling of private trackers. In *2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P)*, pages 1–10, Aug 2010.
- [12] Bram Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72, 2003.
- [13] M.A. de Vos. Identifying and managing technical debt in complex distributed systems. MSc thesis, Delft University of Technology, <https://repository.tudelft.nl/islandora/object/uuid:e5a817a4-ce0a-4dd3-afd4-d70660b63d16?collection=education>, August 2016.
- [14] Ardhi Putra Pratama Hartono. Credits in bittorrent: designing prospecting and investment functions. MSc thesis, Delft University of Technology, <https://repository.tudelft.nl/islandora/object/uuid:809eaec7-883c-47b0-9d57-8e605eaaed1/datastream/OBJ/download>, March 2017.
- [15] D. Hughes, G. Coulson, and J. Walkerdine. Free riding on gnutella revisited: the bell tolls? *IEEE Distributed Systems Online*, 6(6), June 2005.
- [16] Digi International Inc. Python garbage collection. https://www.digi.com/resources/documentation/digidocs/90001537/references/r_python_garbage_coll.htm, September 2017. (Accessed on 03/06/2018).
- [17] Adele Lu Jia, Xiaowei Chen, Xiaowen Chu, Johan A. Pouwelse, and Dick H. J. Epema. How to survive and thrive in a private bittorrent community. In Davide Frey,

Michel Raynal, Saswati Sarkar, Rudrapatna K. Shyamasundar, and Prasad Sinha, editors, *Distributed Computing and Networking*, pages 270–284, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

- [18] Arvid Norberg. libtorrent manual. <https://www.libtorrent.org/features.html>. (Accessed on 02/24/2018).
- [19] Dr. J.A. Pouwelse. Delft blockchain lab: Roadmap 2030. https://d1rkab7tlqy5f1.cloudfront.net/EWI/Delft%20Blockchain%20Lab/Presentaties%20Kickoff/PDF/johan_goed.pdf. (Accessed on 05/07/2018).