Credit Mining: An Incentive and Boosting System in a Peer-to-Peer File-sharing Network

Bohao Zhang

TUDelft

Delft University of Technology

Credit Mining: An Incentive and Boosting System in a Peer-to-Peer 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

31st August 2018

Author
  Bohao Zhang

Title
Credit Mining: An Incentive and Boosting System in Peer-to-peer File-sharing Networks

MSc presentation
  31st August 2018

Graduation Committee
  Dr. Ir. Johan Pouwelse   Delft University of Technology
  Dr. Przemyslaw (Przemek) Pawelczak   Delft University of Technology
  Dr. Claudia Hauff   Delft University of Technology

Abstract

Since 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, BitTorrent involves a tit-for-tat mechanism which does not function satisfactorily against free-riding. Private trackers implement credit systems to eliminate free-riders and award the good-behaving users. However, due to these factors, the community size of private trackers is limited and not even close to that of famous public trackers. Users have to put considerable efforts to maintain a good credit record, making 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 of seeding for the community. Even worse, the hardcore seeders still need to manually download much content and waste considerable resources on over-seeded torrents.

In this thesis, we design, implement and evaluate an incentive and boosting system namely Credit Mining inside Tribler, an open source Peer-to-Peer file sharing program. Credit Mining involves a private-tracker-like incentive mechanism while maintaining good accessibility for every user. Our results show that we have succeeded in creating a profitable swarm selection algorithm that works in the real world. 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".

Contents

Chapter 1

Introduction

Prior to peer-to-peer protocols, most contents on the Internet were distributed through centralized servers. Centralized systems often suffer scalability issues. Since all clients leech data directly from the same server/cluster, requirements of bandwidth, processing power and electricity will increase almost linearly with the number of clients. Centralized servers are not cost-efficient, especially when the data-flow to the server fluctuates heavily. The server either needs to prepare enough hardware for peak periods, which is expensive and becomes overkill when off-peak, or it has to sacrifice the service quality during the peak period. This did not only deny the possibility for independent content creators to provide high-quality service on a massive scale, but also made it cost-inefficient for big companies to do so.

The popularity of peer-to-peer file sharing systems, such as BitTorrent has dramatically changed the way contents are disturbed across the Internet. After the development of nearly 20 years, a large number of individuals and organizations are using BitTorrent to distribute their contents. For example, the gaming industry is widely using BitTorrent protocol in updaters, such as *Battle.net* by Blizzard Entertainment, *Wargaming.net* by Wargaming, and *Eve Online* by CCP Games. Many major open-source and free software projects, like *Ubuntu*, provide BitTorrent as an option, in order to improve availability and reduce the load on their servers. Facebook and Twitter also use BitTorrent to distribute updates onto their servers.

However, BitTorrent suffers from a tragedy of the commons. To achieve collective optimization, BitTorrent expects that every peer contributes as much upload bandwidth as possible. Unfortunately, this is against the self-interest of an individual peer. While BitTorrent's built-in tit-for-tat mechanism attempts to mitigate this, users can still download contents without contributing sufficient upload. In an attempt to force users to share contents, private trackers are keeping records of how much their users have uploaded and downloaded with a cost of user experience.

To provide users with a more attractive solution than the existing centralized private trackers, we have created an automated credit system on top of Tribler [8], a peer-to-peer file-sharing client. By enabling the ability to join swarms and gain more credit automatically, our system can provide a private-tracker-like credit

service to all Tribler users with a convenient user experience.

In this chapter, we provide an introduction to the free-riding problem, the Bit-Torrent protocol, and the Tribler file-sharing client. It is organized as follows: Section 1.1 discusses free-riding in peer-to-peer networks and existing mechanisms to solve this issue. Sections 1.2 introduces the BitTorrent protocol and its mechanism to prevent free-riding. Sections 1.3 discusses the Tribler file-sharing client and how it aims to provide an alternative to private trackers, followed by Section 1.4 which states the contributions of this thesis. Finally, Section 1.5 provides an overview of the remaining chapters.

1.1 Free-riding

Free-riding refers to the situation where users only act in their self-interest and consume resources without sufficient contribution to the community. With only a few uploaders available, a peer-to-peer network will become more and more centralized, degrading the performance, reliability, and robustness of the network.

The free-riding phenomenon is a common challenge among nearly all peer-to-peer file-sharing systems. Even before BitTorrent came to being, Gnutella, one of the most popular peer-to-peer applications at the time had already suffered critical free-riding problems. In 2000, a group of researchers found out 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 a degradation of the system performance and added vulnerability to the system. These researchers suggested that free-riding was even more critical to the system than copyright issues [13]. Furthermore, in 2005, another group of researchers indicated that 85% of peers share no files and 86% share 10 or fewer files [19].

1.2 BitTorrent

BitTorrent was first released in 2001 by Bram Cohen [16] and has been dominating peer-to-peer file-sharing protocols till today. Unlike some of its precursors like Napster or Gnutella, the BitTorrent protocol does not limit itself to a specific type of service but provides a universal solution to peer-to-peer file sharing. The typical workflow of BitTorrent is as follows:

- The user finds the link to a torrent file on a website describing the content he desires.

- The user adds this file to a BitTorrent client software. The client communicates with a centralized server namely tracker to find other computers running any BitTorrent compatible clients that have the download task of the same content described by the same torrent file. These computers can be either still downloading or with the complete content.

- The user's client tries to establish connections to the computers introduced by the trackers. Once connected, the client will try to download the segments they do not have from the other computers and upload what they already have to other computers.

- Every client keeps a periodical contact with the tracker in order to update the list of other computers as well as notify its existence.

From this workflow, we can see that BitTorrent is not yet a "fully" distributed system. There are two different types of nodes in the BitTorrent network, downloaders and trackers. Trackers are fewer in number but more crucial, making it vulnerable in the network. Moreover, to distribute contents on the BitTorrent network, the content provider need to generate a static file with the extension *.torrent* containing the metadata and distribute it among other downloaders. Downloaders specify the contents they need with these torrent files to the trackers. These torrent files are usually published on centralized websites. The existence of these torrent-discovery websites along and the trackers lowers the degree of decentralization of BitTorrent networks.

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 the 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 Internet traffic share from the uprising of file storage applications like Google Drive and video services provided by, for example, Netflix and Amazon in recent years, it still holds the largest upload bandwidth share in most of the world and a noticeable share in downloading, according to research done by SANDIVE, shown in Figure 1.1.

In attempts to fix the free-riding problem in BitTorrent, there are two major methods taken. Their principles and limitations are described in detail in Section 1.2.2 and 1.2.3. Before that, we first need to clarify the terms and definitions in Section 1.2.1.

1.2.1 Terms and definitions

BitTorrent has been a popular concept for more than a decade. Therefore many ambiguities have been created during its popularization and evolution. To avoid the potential ambiguities that might occur in this thesis, we clarify the main proper nouns in this section. The special usage in this thesis is also introduced.

Peer A peer is one instance of a compatible BitTorrent client running on a computer on the Internet, to which other clients connect and transfer data. Depending on context, a "peer" can refer to either such a client in general or a more specified client transferring the same content.

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%

sandvine

Figure 1.1: Top 10 applications in Internet usage through fixed access in Europe in 2015.

tracker A tracker is a server that keeps track of which peers are in the swarm. Clients report and receive information to the tracker periodically and receive information about other clients to which they can connect in exchange. The tracker is not directly involved in the data transfer and does not have a copy of the file.

Swarm Together, all peers transferring the same content are called a swarm. Note that "swarm" does not require all the peers in it acknowledge each other, nor even the topological connectivity of the whole graph. The tracker does not necessarily give the whole graph of the swarm to each peer.

Seed/seeder(noun) A seeder is any peer that has already downloaded 100% of the content but still remains in the swarm and uploads to peers. Note that the progress of a download is defined as $download/demand$. So, in special cases, a peer with part of the content, but only uploading and not downloading, is also regarded as a seeder. In this thesis, "seeder" is more often referring to these special cases.

Seed(verb) The action of uploading taken by seeders.

Overseeded "Overseeded" is the word to describe the situation where there are too many seeders compared to the downloaders, making it hard for seeders to find a downloader or to seed.

torrent A torrent is the file with .torrent extension name and contains the UTF-8 encoded metadata of files described by it. It also contains the address of a tracker that coordinates communication between the peers in the swarm.

1.2.2 Tit-for-tat

Fully aware of the free-riding phenomenon on Gnutella, the BitTorrent file distribution system was implemented with a tit-for-tat strategy to break this reiterated prisoner's dilemma among the peers.

The BitTorrent protocol has no central resource allocation. Trackers are not responsible for allocating resources. They only provide information about peers. Each peer aims to achieve its maximum download rate by downloading from whomever they can and uploading to peers hopefully cooperative to them, determined by a tit-for-tat algorithm.

BitTorrent clients label new peers as "choked", meaning that they do not upload to them. Each client has a fixed number of unchoked slots, by default four slots. The decision of which peers to unchoke is based strictly on current download rate. The peers where the client gets the most benefit get unchoked. The unchoked slots were updated every 20 seconds [16] at Bittorrent's initial launch. This period was later shortened to 10 seconds [11]. There is also one slot 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 particular connection, this connection will be labeled as "snubbed". The client will not upload anything to the "snubbed" peer unless it is selected for optimistic unchoking.[11][16]

However, the tit-for-tat algorithm does not entirely stop free-riding. The Achilles' heel of BitTorrent is that it does not provide a traceable credit record relating to a particular user. A selfish user in one swarm is not identified in another swarm. Moreover, it is too easy to get a new identity in the BitTorrent network and replace the infamous one, making this tit-for-tat algorithm less useful even in a single swarm.

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

1. Turning off the client or remove/stop the download once it is finished to avoid further upload to other peers.

2. A selfish user can upload only to the peers with the pieces he needs, to gain higher reputation only to them.

3. Uploading nothing or little while downloading, and then rejoin the swarm with a new identity after being choked by other peers.

4. Creating multiple identities to get more bandwidth from other peers.

5. Some clients can create a separate swarm for peers with this client, and only upload to these peers.

These cheating tricks can be utilized independently or jointly by either individual users or clients on a massive scale. These tricks will achieve local optimization for the selfish users, harming the rest of the community. This can cause "bad money drives out good money". Peers using cheating tricks or clients will always outperform the honest ones, turn more peers selfish and further penalize honest peers.

1.2.3   Private trackers

Private trackers is an approach to avoid free-riding by involving membership. Typically, a "private tracker" includes both a torrent-discovery website and an affiliated BitTorrent tracker. Membership is required to use either of the services. Registration is usually not open on these private trackers. New members need an "invitation" from an existing member. Usually, invitations can only be provided by the veteran members. Sometimes it also requires a considerable amount of credit to purchase such an invitation, forcing the inviter to consider the quality of the invitee seriously.

Private trackers usually use strict rules to maintain the quality of the members. Users gain credit from uploading and lose credit when downloading. There is also Sharing Ratio Enforcement (SRE), which requires all the members maintain a share ratio($upload/download$) above a certain threshold. Users who run out of credit 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 a long time to improve their share ratio [21]. In research [15], this is called "uploading starvation". Every peer is forced to upload more to maintain a good reputation in the community. Since the demand for contents is not infinite, the more uploaders there are, the less efficient for them to get any upload.

New members usually are significantly limited in their download capabilities in order to increase the cost of selfish users camouflaging themselves as new members. However, this also creates a huge barrier for the newcomers. A fresh member first needs to find an invitation from an insider. Then he would need to learn the rules of the private tracker and how to configure the BitTorrent client. Since he starts as a new member with no credit, he also needs to download and seed some contents either "free"(do not require credit for downloading) or obtained elsewhere to gain some credit. Only after gathering enough credit can he eventually start to download the contents he initially wanted.

In conclusion, a private tracker is a closed elite community. It provides high-quality service to law-abiding members. As the cost, the rules are strict. It requires much effort to maintain the credit in the community and blocks most new users from joining. Thus the size of the community is limited. Researchers [15] have show that private trackers are not comparable to public trackers in the number of both contents and users.

Community Name	Description
AllChannel	The community which all Tribler users automatically join. As the name suggests, metadata of all the channels is transferred through this community.
Channel	The community that represents a single channel. A user will join this community when he subscribes to the corresponding channel. The contents of this channel are 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	The community for tracking the reputation of peers using TrustChain, a utilization of Blockchain Technology.
TriblerTunnel	The community that enables anonymity when downloading contents from the Tribler network by routing through other Tribler peers using a Tor-like protocol.

Table 1.1: Function of communities in Tribler.

1.3   Tribler

To better eliminate free-riding while avoiding the major drawbacks, the researchers from Delft University of Technology have developed Tribler [26].

Tribler is a decentralized peer-to-peer file-sharing system based on the BitTorrent protocol. It is not only a compatible BitTorrent client, but it also implements an overlay for peer-to-peer communication across NAT/firewalls. It enables the decentralized transmission of data such as torrent information, swarm popularity, and user credit record. It also has a Tor-inspired onion routing to hide users' IP addresses which are not protected in BitTorrent protocol. Tribler is designed to protect the user's privacy, to be attack-resilient, as well as to reward content creators and bandwidth donors directly. The current goal of Tribler is to build a programmable micro-economy without banks, governments or any centralized agencies.[2][7][25]

In Tribler, each user, identified by its private key, can create a channel. The private key is automatically generated for the user, and the user does not need to go through a registration like in private trackers. Metadata of BitTorrent swarms, known as torrents can be uploaded to these channels. Data in Tribler is transferred within Tribler communities. A Tribler community is similar to a network overlay: users can generate/join/leave a community and exchange messages over it. Different types of messages are transferred through different communities. Currently, there are six main types of communities. The roles of these communities are described in Table 1.1.
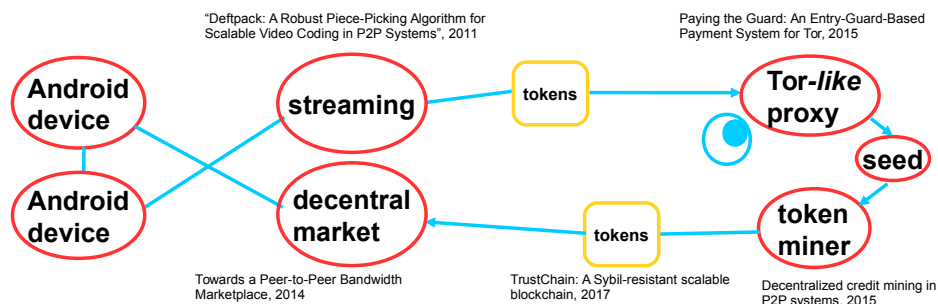
Figure 1.2: High-level overview of trustworthy micro-economy for bandwidth tokens.

Trustworthy Micro-economy for Bandwidth Tokens

"A blockchain-based token economy to prevent bandwidth free-riding" is an on-going long-term high-priority 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. There should also be a basic policy to 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).[9]

The high-level overview and architecture diagram are shown in figures 1.2 and 1.3. This thesis serves the role as the "token miner" in both diagrams.

In Tribler, tokens are issued to users when they contribute to the community and are spent when they get help from the community. To earn tokens, a Tribler user can donate his bandwidth to the community by seeding when idle [9], or relay Tor-like encrypted traffic.[5] A user, on the other hand, needs to pay tokens when they consume these services. There is also a decentralized market built in Tribler where users can trade tokens with Ethereum or Bitcoin.[2]

Within each exit node in Tribler, there are two different type of slots to provide service: competing slots and equal opportunity slots. Equal opportunity slots can serve any other Tribler user, but competing slots are only reserved for the request coming in from users with the highest token balance. An existing circuit in one of the competing slots might be destroyed if a request from a user with higher token balance comes in.[3] In this way, when downloading from the Tribler community,
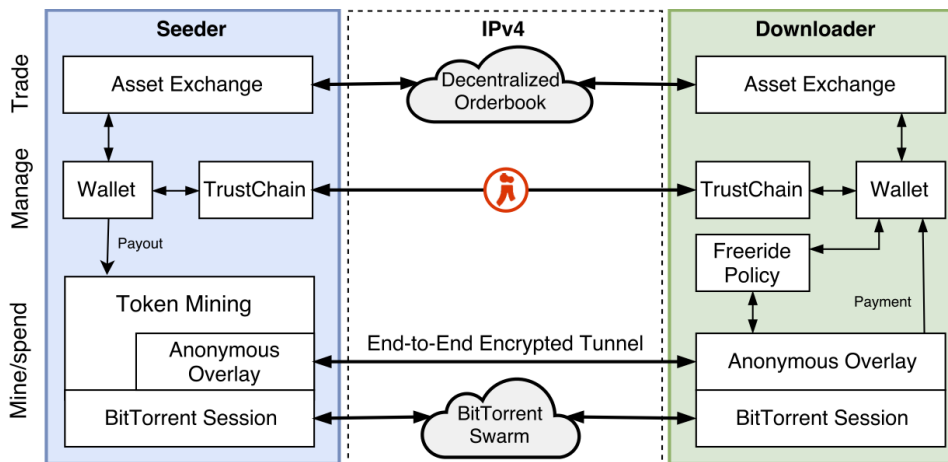
Figure 1.3: Initial architecture diagram of trustworthy micro-economy for bandwidth tokens.

the speed one user can get is positively correlated to his token balance. The contributors are rewarded, and the free-riders are punished.

1.4 Research questions and contributions

In order to convert idle bandwidth into tokens while boosting the health of Tribler network, we design and implement a platform that automates the process of selecting profitable BitTorrent swarms. We focus on achieving positive results using real-world swarms since this is where previous works on this subject did not prove themselves.

In this thesis we give an answer to the following research questions:

1. How to design and implement a system that automates the process of joining BitTorrent swarms in order to receive a higher credit rating on a private-tracker-like platform?

2. Is it possible to design a swarm selection policy that achieves positive results when using real-world swarms and how does implement such a policy?

In order to answer these questions, we provide the following main contributions:

- We design and implement a system called "Credit Mining", relying on existing features of Tribler such as its channels, download engine, and bandwidth accounting capabilities[1].

---

[1]https://github.com/EinNarr/tribler/tree/CreditMining_operation

9

- We design various swarm selection policies that can be used within Credit Mining, and evaluate their real-world performance. The experiments are done with the help of a Web crawler[2] based on scrapy library to fetch torrent information, a script[3] to download torrents files from their info hash and update Tribler channels with these torrents in batches.

- We also implement a prototype of Kodi plug-in[4] to enable the ability to run Tribler, as well as our boosting system for Tribler.

These contributions will give Tribler the capability of automatically mining bandwidth token, a feature much needed in order to achieve an easy-to-use platform with private-tracker-like service quality. We hope that this system will further expand the user-base of Tribler.

1.5   Thesis layout

The remainder of this document exists of five chapters are organized as follows. Chapter 2 defines the requirements of our Credit Mining and discusses how previous implementations failed to meet these requirements. In Chapter 3, we present the design and implementation of Credit Mining. The experiments and the evaluation are discussed in Chapters 4 and 5. Finally, we conclude in Chapter 6 and propose the future work of this project.

---

[2]https://github.com/EinNarr/TorrentSpider
[3]https://github.com/EinNarr/TriblerChannelHelper
[4]https://github.com/EinNarr/plugin.video.tribler

Chapter 2

Problem Description

Peer-to-peer file sharing systems critically depend on the voluntary donations of bandwidth by their users. However, many people do not contribute sufficiently in such systems, who are called free-riders. This problem is central in this thesis. In Tribler, we plan to eliminate free-riding by reinforcing a micro-economic system. In this system, uploading to other peers will be rewarded with bandwidth tokens and downloading will cost these tokens. Users can enjoy more benefit in Tribler network with more tokens and will be punished when they run out of tokens. In this way, it becomes consistent to help the community and achieve personal gain. To help users automatically earn token and contribute bandwidth conveniently, we need a new framework in Tribler to do the work. We name this framework "Credit Mining".

In Section 2.1 and Section 2.2, we introduce our requirements to this new framework in both the performance and usability perspective. Furthermore, in Section2.3, we review the existing solution and its limitations.

2.1 Performance requirements

Credit Mining should be capable of automatizing and optimizing the process of seeding swarms as well as earning tokens in the Tribler network. Thus the minimal requirement should be as follows:

1. Universal usability to work on a set of different swarms, regardless of their individual availability.

2. Achieving more upstream than downstream in the long term.

3. A reasonable amount of resources usage.

Considering Tribler is still under development, and Credit Mining should be as profitable as possible to all its users, it should also achieve the following requirements:

1. Providing a positive yield as early as possible.

2. Achieving upload gain($upload - download$) at a reasonable speed.

3. Potential to handle the huge number of swarms in Tribler network.

4. Maintainability for further changes that will occur in Tribler.

5. Availability for low-cost devices.

2.2 Usability requirements

Credit Mining is designed for both elite users and average users in the Tribler network. From the feedback of the current pre-alpha version of Credit Mining, even the loyal users are sometimes confused about what Credit Mining does and how to properly use it [10][12]. We are aware that to extend the user-base, it must remain simple and usable to all the users. In our implementation, the usability requirements are as follows:

1. Able to start with minimal configuration.

2. Zero or minimal human interaction from the user during execution.

3. Able to run in the background.

4. Available for advanced users to specify the content to mine on.

2.3 Related works and problems

Trust has always been in the center of nearly any kind of community. Besides cyber security domains, trust is also widely discussed in other domains. Article [30] introduced how trust can be used in a smart power grid to detect intrusion. In Article [22], sociologists discussed how cross-national differences in social and spatial mobility could influence the level of trust in the community. In Article [31], another sociologist researched different modes of trust production and how it affected the industrial formation in the US. Article [27][23][28][29] analyzed the nonnegligible relation between trust and democracy we all enjoy.

Credit Mining aims to reinforce trust in a programmable, block-chain-based system. According to the Github Wiki of Tribler [8], Credit Mining first entered early Beta in 2013. It was later published in 2015 by Mihai Capotă[14] and improved by Ardhi Putra Pratama in 2017 [18]. However, due to this discontinuous develop process, there are a remarkable number of less optimized development decisions and technical debts. In this section, they are discussed in detail.

2.3.1 Complexity problem

Time complexity is not mentioned in any of previous Credit Mining related works [14][18]. We recover the design decisions by analyzing the source code and unit test code [4].

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 a messy hash table storing various objects corresponding to this swarm. These include the download handle object, the swarm definition object, the current state information object, a hash table of extracted state information, and even a hash table of the retrieved information of all the known peers in the swarm and the flags of their states in Credit Mining. Moreover, the whole hash table is poorly maintained. There is hash table nested in another hash table which is also nested in another. And no comment or document introduce how this complicated hash table is structured. Elements are not initialized collectively but inserted to the table whenever they are generated.

The state of a swarm in Credit Mining is stored in one of the sub-hash-tables mentioned above. When querying for a subset of the swarms, such as all the enabled swarm, Credit Mining has to traverse through all the elements in the swarm pool. The resource and time consumption is linearly related to the number of swarms, which is inefficient in real-world scenarios where the swarm pool is enormous.

2.3.2 Pre-download

Pre-download, or speculative download, is a core feature of Credit Mining in previous implementations. It is the module that assists in predicting the potential of the swarms. The prediction methods discussed in the previous works [14][18] are all based on the swarm characteristics that are available to all peers. Since some of the information like the swarm size is only provided after the client establishes the download, this module is designed to download a minimal amount of content from the swarm and fetch the swarm size information.

However, this module does never work well in reality. Firstly, it consumes more than an acceptable amount of resource to work. To quickly iterate through the whole swarm pool, it needs to create download handles rapidly, start downloads, and check the download progresses periodically with a short interval. Once the progress hits a certain threshold, characteristics of this swarm will be recorded, and the download task will be removed. It is implemented without proper scheduling. All the swarms are rushed to be added to this module once discovered. It immediately causes high resource usage and lagging when tested on our high-performance test machine.

In Pratama's research [18], there is also a mechanism to download the rarest pieces of the content first to get a better knowledge of the swarm. This feature consumes much more resources by periodically traversing and sorting every piece of the content in every swarm. The complexity increase from $O(n)$ to $O(mn \log m)$, where m and n are the number of pieces in one swarm and the number of swarms. Considering the number of swarms that Credit Mining is expected to handle in the

real world, this is not effective. Moreover, in the libtorrent manual [24], the piece picker is already described as "optimized for quickly finding the rarest pieces". This feature is a waste of resources. It is repeating what libtorrent is already doing inefficiently.

Scheduling the pre-download tasks and limiting the maximum swarms being speculated in parallel can lower the CPU usage effectively. However, by observing the time cost on average to get the characteristics of a swarm is relatively high and highly unpredictable. The evaluations of previous works were tested on an insufficient number of swarms [14][18], which concealed the problem. It will take considerably longer to traverse all the swarms in even a single channel in the real world. For example, the most popular channel on Tribler contains more than 10 thousand swarms. Even if all the swarms can be eventually speculated after a long time, the characteristics of the swarms speculated earlier will be no longer reliable by the time all the swarms are speculated due to the significant time difference.

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

In addition, there is another project on-going in the Tribler team, trying to solve this problem by distributing the task to all the peers in the community. In this approach, each peer will only need to investigate a limited number of peers and get the rest gossiped by other peers [6]. This could be a powerful aid to Credit Mining in the future.

2.3.3 Swarm selection

In previous works, multiple policies are introduced and evaluated in detail. The policy that always selects the swarms with higher leecher/seeder ratio has proved to be effective. However as explained before in Section 2.3.2, this policy only works fine in the lab environment when the size of swarm pool is not too large to traverse and getting the information of every single swarm is realistic. As the size of swarm pool grows in the real world, the policies fully relying on the knowledge of the swarms are no longer practical due to the reason that Tribler lacks a function to fetch the content popularity of its swarms effectively.

Moreover, the test groups of previous experiments are too weak to be convincing. In Capotă's experiments, only random policy and creation time policy are compared against the seeder-leecher ratio policy [14]. The problem is that random policy is hardly a "policy", while the creation time policy is too naive to work efficiently. Pratama's work makes further improvement by proposing a scoring policy using more factors of a swarm to evaluate its potential [18]. However, there is no description of how and why the weight of each factor is selected in the evaluation, neither can we repeat his experiment.

This situation might change after another project called "swarm size" mentioned at the end of Section 2.3.2 is completed [6]. The knowledge-based polities might
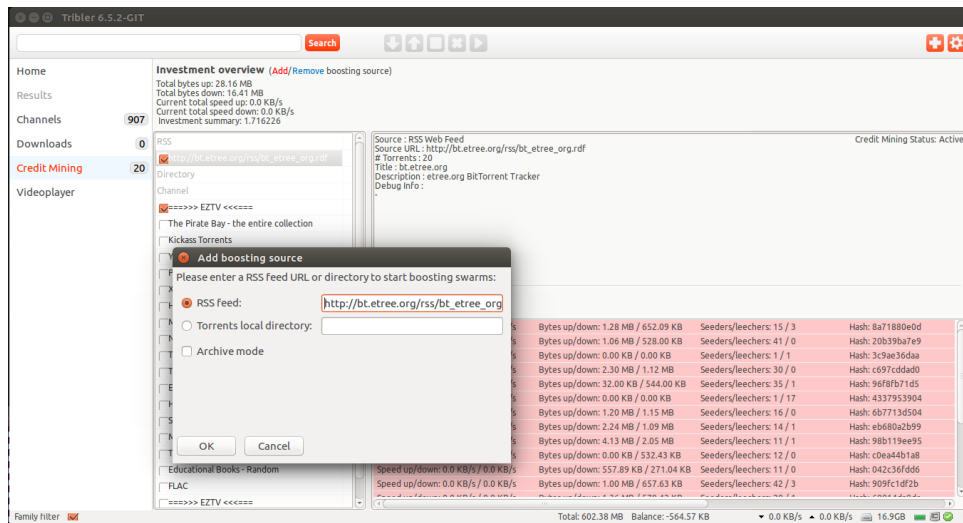
Figure 2.1: Credit Mining GUI in previous implementation [18].

be brought back with its support.

2.3.4 Software engineering

Similar to the technical debts in other subsystems in Tribler mentioned in M.A. de Vos's research [17], the Credit Mining prototype also contains many architectural impurities and incomplete testing framework problems. Moreover, it lacks proper documentation or comments of the implementation details, making it hard to maintain.

Neither Credit Mining nor its so-called "unit-test" cases in the latest branch are functional at the beginning of this thesis. It then becomes a "pulling ourselves up by the bootstraps" problem. Since neither the code nor the test is trust-worthy, when an error is raised, there was no way to determine what the cause of the error is. It can either be caused by the code, the test case, or even both.

By the end of Pratama's thesis, a Graphical User Interface(GUI) was implemented for Tribler 6.x with the wxPython GUI framework shown in Figure 2.1. However, Tribler had entirely moved onward to 7.x versions before the completion of his thesis, in which PyQt 5 is used instead of wxPython. This makes all previous Credit Mining GUI implementation meaningless. Moreover, this implementation contains quite complicated indicators and channel selection panels which are not user-friendly enough.

Though never mentioned in the previous reports [14][18], Credit Mining is not able to be dynamically turned on and off, or "hot swapped". We consider this feature as part of the technical debt. According to our observation that previous implementations lack proper unit tests, has an incompatible GUI and is severely out-dated from the main branch of Tribler, we speculate that Credit Mining was

rushed during the finalization phase. Lack of hot-swapping feature is also the result of the rush.

Credit Mining is not hot-swapping capable since it is only initialized during the startup of Tribler. When enabling or disabling Credit Mining in the setting pages, only the configuration is changed and saved, Credit Mining itself is not initialized or destroyed. Thus a full restart of Tribler is always required.

If Credit Mining is disabled, the function to shutdown Credit Mining and delete its reference from Tribler is called. However, Python only deallocates an object when its reference count becomes zero, thus is not able to handle reference cycles [20]. There are many reference cycles in various classes in Tribler. In previous implementations, since Credit Mining only needs to be destroyed before shutting down Tribler, it will not cause any trouble if garbage is not recycled completely. However, if Credit Mining can be toggled multiple times as it should be able to, the amount of non-collectible garbage will keep increasing and result in memory leak.

Chapter 3

Design and Implementation

In previous chapters, we introduced the related works and purposed the idea to solve free-riding with micro-economy system in Tribler. To earn tokens with idle bandwidth in Tribler micro-economy system is similar to earning profit in market economy. A user needs to first spend some tokens to download some content as investment, then he is able to seed it and gain tokens as profit. Just like in market economy, not all investments are profitable or profitable enough. We need a system to dig out the gold from the ores, which are the countless BitTorrent swarms on Tribler network.

Credit Mining is an automatic framework built to dynamically join or leave Bit-Torrent swarms based on their profit potentials. By joining swarms, users can contribute their bandwidth to maintain the availability and improve the performance of the swarms, while investing some bandwidth token and hopefully getting more bandwidth tokens as profit. Credit Mining aims to maximize the speed of earning tokens for the users, essentially maximizing the total upload minus the total download. Nevertheless, Credit Mining also provides a friendly user experience.

The most remarkable feature of Credit Mining is that it does not require users to interact with the system manually. Previously, seeding is commonly considered as the action that a user manually select one or some torrents, download the contents and then upload to the rest of the swarms. This comes with the limitation that it is nearly impossible for a user to always find the swarms need the most help. Manually downloading the contents with no desire is boring and meaningless to the users.

Credit Mining is going to take seeding to a whole new level. Working on a large number of swarms, Credit Mining no longer views "seeding" by torrents or files, but breaks the contents into segments. It no longer tends to download the whole file or all files defined by the same torrent. Users do not need to care about what files they needs to download. Credit Mining will take care of it and always tend to find segments that need the most help from the contents it knows.

To simplify the design of our system, throughout this thesis we will consider the following to be the case:

1. The number of tokens awarded to a user is only linearly related to his total

upload minus total download.

2. There is only one single miner in the swarm, meaning that the competition against other miners is not considered.

Firstly, Section 3.1 will discuss the system architecture and how it cooperates with existing Tribler components. Next, Section 3.2 will discuss the notion of a Credit Mining policy, a component essential for achieving a positive bandwidth yield. In Section 3.3 we will talk briefly about the graphical user interface. And finally in Section 3.4, we will summarize our software engineering effort.

3.1 System architecture

Our credit mining system is comprised out of three main components.

Manager The manager component is the main component of the system. It fetches swarms from the sources, ensures that swarms are started/stopped, manages disk space and executes policies.

Source The source component is responsible for listening for new swarms and notifying the manager.

Policy Policies determine which swarms are to be mined for credits. Usually, a policy will look at the swarm itself, along with the state of the swarms in order to determine which swarms are best suitable for credit mining.

Each source component listens for newly discovered swarms within a specified channel. Once a new swarm is discovered, it will be added to the swarm pool that is maintained by the manager. The manager will periodically run the policy to evaluate the swarms and start/stop downloading the swarms accordingly. The ultimate goal for the policy is to earn tokens fasts. The process of creating and keeping track of bandwidth tokens is handled automatically by Tribler itself. Trustchain is a blockchain-like system handling the record and transaction of tokens. The simplified architecture is shown in Figure 3.1.

3.2 Credit Mining policies

The most crucial component of our system is the policy. A policy is a filter that given the whole swarm pool and Credit Mining's working state, while returning the control commands towards the swarms to the manager. These commands can be what swarms to start, stop, set download limit or remove. Once Credit Mining is enabled, the policy will be applied periodically.

Policy class is implemented as a standard interface to swarm management that it can be easily added or extended. Limited by the current state of Tribler and knowledge of swarms we have access to, policies may need to be improved in the further. Modularity is essential for reducing the technical debt of this project.
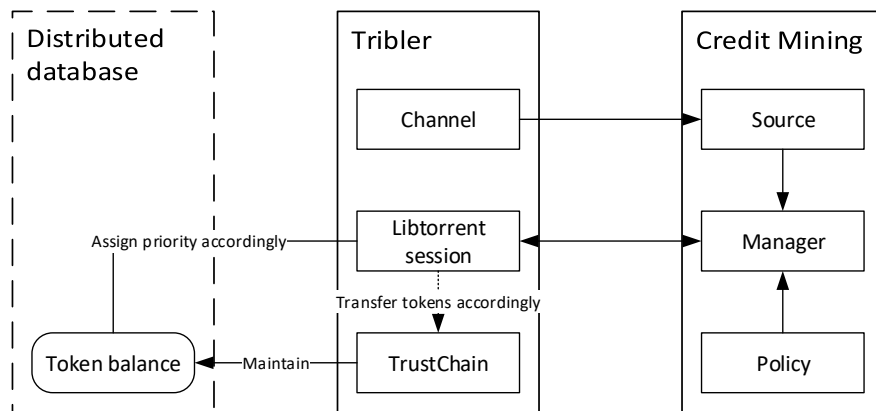
18

Figure 3.1: Simplified credit mining architecture.

3.2.1 Building blocks of a policy

We formulate a methodology for creating and optimizing a policy. A policy should be build up with some or all of the following blocks:

Opportunity discovery The discovery of swarms with potential, the source and creation time are the key elements to be considered. In Tribler, the sources are mainly the Tribler channels. Currently, we leave the choice of channel to our users. Within the limit of the total number to be invested, we tend to collect newer torrents for further selection, since newer torrents are more active in general.

Initial investment The first blind investment tends to filter the worst swarms from the pool. These include the dead and extremely overseeded swarms. We should aim to filter the torrents with the worst upload potential with a minimal cost of download bandwidth to avoid their harm to the yield.

Addition investment From survivors of the initial investment, we further evaluate the potential select the good ones. More download bandwidth can be assigned to a swarm as investment if it is predicted to be profitable. In this phase, more resource can be invested into each swarm in comparing to the initial investment.

Dead swarm detection There are a lot of abandoned or unpopular torrents on the Internet. They need to be detected and suppressed to avoid much resource waste.

Exhaustion detection After being mining for a long time started, the profitable swarms will eventually become non-profitable, but they will still reserve the
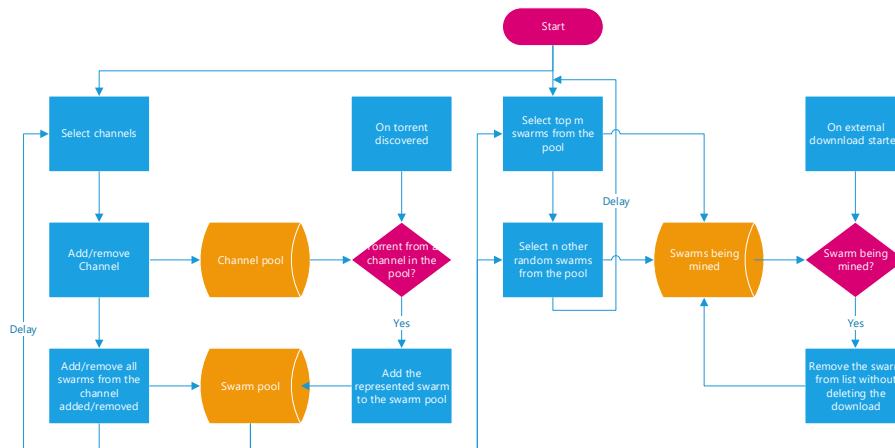
19

Figure 3.2: Workflow of Credit Mining under Vitality Policy.

storage. We call this "exhaustion". For long-term efficiency and autonomy of the system, exhausted swarms need to be detected and removed.

3.2.2 Vitality Policy

As an initial attempt to design a suitable Credit Mining policy, we implemented the Vitality Policy. The Vitality Policy is highly scalable because it evaluates the swarms by testing the swarms' actual performance, thus does not need to get the information of the whole swarm pool. The policy is inspired by the "optimistic unchoking" mechanism in BitTorrent as described in Section 1.2.2. Likewise, we select a certain of amount of swarms to investigate according to their prior performance, retire worst swarms being investigated, and select several swarms randomly to keep the vitality of Credit Mining.

The workflow of Credit Mining under Vitality Policy is as described in figure 3.2. When the policy is applied for the first time, it randomly picks a number of swarms equal to the maximum swarms that can be mined in parallel for the initial investment(duplicate can happen in the first interval). The maximum number can be configured by the user, based on the bandwidth he is willing to share. From then on, every time the policy is applied, it first sorts all the swarms according to the upload amount during the 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 the real world, the swarms whose upload is below a certain threshold are regarded as "dead" and also put into the "to stop" list. This feature serves as both dead swarm detection and exhaustion detection. The surviving swarms are granted additional investment. The policy then randomly selects the same amount of swarms which are not enabled at this moment for their initial investment to fill
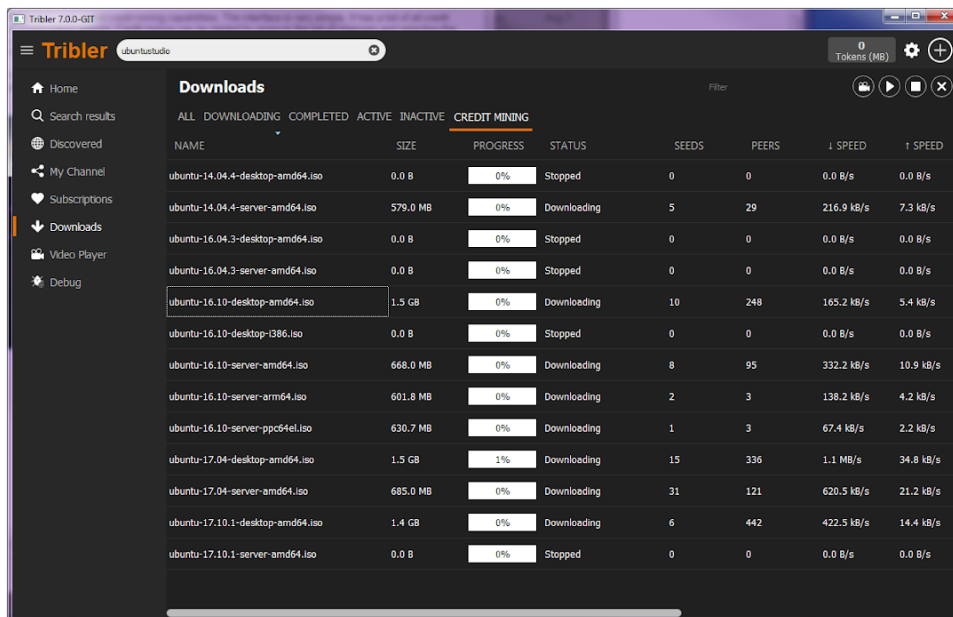
20

Figure 3.3: Credit Mining GUI in our implementation.

in the slots left by the swarms to be stopped. These swarms are put to a "to start" list and return to Credit Mining manager along with "to stop" list.

In the following chapters, we will evaluate this policy and other alternative policies in detail.

3.3   Graphical user interface

We extended the default Tribler graphical user interface (GUI) and added credit mining capabilities. The interface is straightforward. It has a list of all credit mining downloads including their states and a small set of configuration options. Credit mining can be started by going to the list of channels and selecting the "Mine credits" button next to a user channel of choice. The user only has to select which channel(s) to mine and has no further influence on the mining process. The status of the process can be viewed from the "Credit Mining" tab (see figure 3.3).

Like all other features in Tribler, communication with the GUI is achieved through a JSON web API. This allows developers to quickly develop alternative GUIs. We have already started working on a Kodi plugin (figure 3.4) to port our work as well as the whole Tribler project to low-budget Android devices as well as the televisions in living rooms.
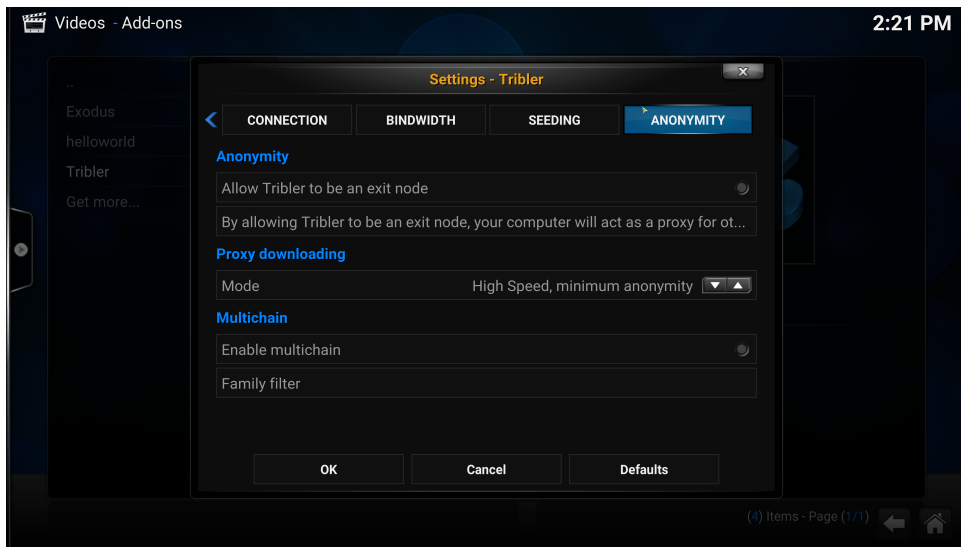
Figure 3.4: Kodi plugin for Tribler.

3.4 Software engineering work

We re-implemented the whole Credit Mining module to replace the one existing in Tribler. Aside from the new policies, the re-implementation mainly focused on improving coding style, code coverage of unit tests, modularity, asynchronized callback chains, data-structures and time complexity.

Measuring by line of code, we inserted 5199 lines and deleted 1961 (duplicates included) in the re-implementation. To implement the Kodi plugin for Tribler, we inserted another 2604 lines.

Chapter 4

Experiments and Performance Analysis

In the previous chapter we explained how we aim to solve the free-riding problem with the micro-economy system in Tribler. In this chapter, we deploy our Vitality Policy introduced in Section 3.2.2. After introducing the test environment in Section 4.1, we first test our proposed algorithm in a controlled environment to validate our implemented mechanism in Section 4.2 and 4.3. After that, we use real-world BitTorrent swarms to validate our assumption and measure performance in Section 4.4. The lessons we learn is summarized in Section 4.5.

4.1   Test environment

In this chapter, tests are run on two different devices according to the need. The first and the second experiments are executed on a 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 and all the experiments in the next chapter are executed on a remote server with the following specifications:

- CPU: Intel Xeon Processor E3-1230 v2 (8M Cache, 3.30 GHz to 3.70 GHz)
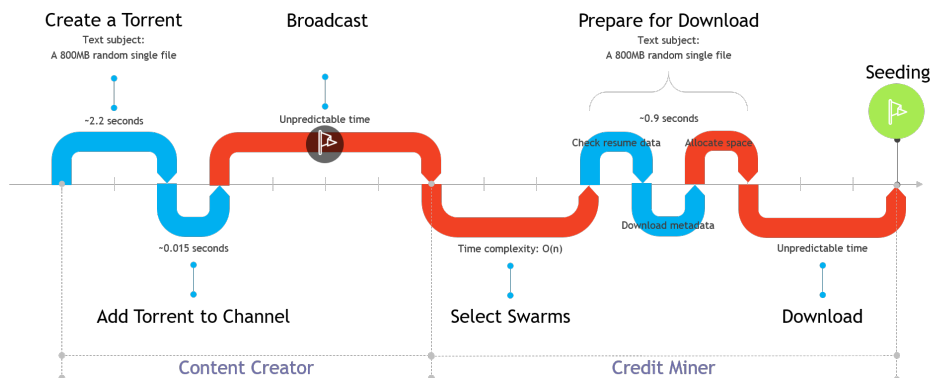
- RAM: 16GB

- SSD: 120GB

Figure 4.1: Timeline of Credit Mining.

- Kernel Version: Linux 4.4.83-1-pve #1 SMP PVE 4.4.83-96

Tribler uses a peer-to-peer database to store all the channel information and swarm metadata, making it impossible to determine the time to fetch specific information from other peers. To eliminate the influence from this random factor, instead of testing on an existing remote channel, we create a test channel on the test device filled 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 any channel, regardless whether the user has the ownership of this channel or not. So mining on our test channel is functionally equivalent to mining on other channels in the real application.

A Tribler swarm is a subset of the BitTorrent swarm defined by the same torrent file. The difference is that Tribler peers use Tribler as their client, and transmit extra information among other Tribler peers through the distributed overlay implemented in Tribler. The credit record and proof of contribution in the micro-economy are both transfered through this overlay. From BitTorrent protocol's perspective, a tribler peer is no different from a normal BitTorrent. So in all the experiments in this thesis, we do not differ Tribler peers from BitTorrent peers to get more test subjects.

4.2 Functional validation

The first experiment aims to verify that the system is functional. We generate a random 800-megabyte file to mimic the content distributed through Tribler in the real world. Then we add the torrent created from this file to our channel and use another BitTorrent client instance to seed it. This experiment is done several times to get a more accurate result.

We monitor the time from creating a torrent from the test file on 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 4.1. We notice that the

24

most time-consuming process is the torrent generation. Swarm selection can also consume much time if the size of the 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. However, since the torrents are distributed through a Tribler channel in the real world, the time it takes for the Credit Miner to receive the torrents is not predictable.

4.3 Controlled environment validation

The second experiment aims to validate the policy mechanism of Credit Mining. The test channel contains ten different swarms with various setups. The content of each swarm is a randomly-generated-800-megabyte file. Peers mimicked by six individual local libtorrent sessions controlled by a script. Each session takes a different port from 6881 to 6886. They are notified of the existence of each other as well as the Credit Miner. The sessions are configured to only communicate with localhost, and all external IP addresses are filtered. The *allow_multiple_connections_per_ip* option in libtorrent session is switched on to make it possible for the sessions to communicate with each other.

The upload and the download speed limits of each torrent in each session are set to 100kb/s and 200kb/s respectively to simulate real-world bandwidth limitations. Otherwise, the upload and download speed is almost the read and write speed of hard disk. If no limit were configured, the whole test would end in seconds, and the result would be completely random. The "lucky" session which first established connections will consume most bandwidth of CPU and hard disk, starving the rest session and making the test result unreliable. What's more, without a speed limitation, as long as there is an uploader, the download speed will always be maximum, limited only by the computation capability but not the network. This leads to a situation where the number of sessions has hardly any impact on the test result.

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

Figure 4.2 shows the how Credit Mining selects swarms during the 30-minute experiment, excluding the swarms randomly selected by ranking. Figure 4.3 show how it joins swarms, including the swarms randomly selected. During this experiment, the maximum number of active swarms is 3 selected by ranking, and another 3 randomly joined from previously inactive swarms. Note that this experiment is to verify whether the system can differ the performance of each swarm. Since the size of the swarm pool is relatively small compared to the active swarms within each
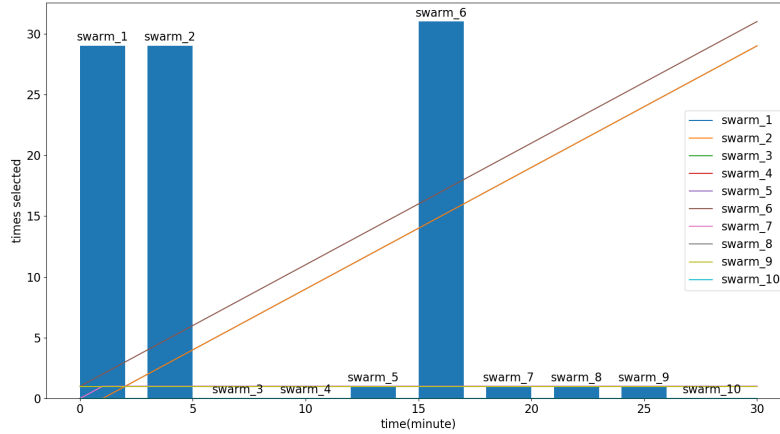
Figure 4.2: The number of times each swarm is selected by ranking.

iteration, Figure 4.3 cannot correctly show the significant time difference of each swarm. When the size of the swarm pool grows, the proportion of joined times between the swarms with good and bad potential will also grow.

By comparing Figure 4.2, Figure 4.3 and the raw experiment log, we observe that swarm 5, 6, 7, 8 and 9 are randomly selected in the first iteration of Credit Mining(one duplicate happens in the initial selection). After this iteration, swarm 8 and 9 are dropped due to their bad performance. Swarm 1, 2 and 3 are 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 and compared, but dropped at the end of iteration since they have lower performance than swarm 1, 2 and 6.

In Capotă's work [14], seeder/leechers ratio is proved to be an efficient approach to rate a swarm. If their policy were optimal, swarm 6, 7, and 8 would have been selected in our experiment since they would have ranked higher than others in performance. However, our algorithm chooses swarm 1, 2, and 6. This proves that the seeder/leecher ratio policy does not always provide the optimal solution. Swarm 6 keeps getting selected in each round because it has equal profitability compared to swarm 7 and 8. Credit Mining will select a different swarm only if it expects higher profitability from this swarm. The swarm joined earlier has a slight advantage to other swarms with the same configuration. As a result, Credit Mining in this experiment locks on only swarm 6 other than swarm 7 or 8.

To get more accurate results, we repeated this experiment several times. The result was similar to the experiment shown in Figure 4.2 and Figure 4.3. Swarm 1 and swarm 2 are always the top choices of Credit Mining once investigated. There is always a third swarm being selected just like swarm 6 in Figure 4.2. Through our
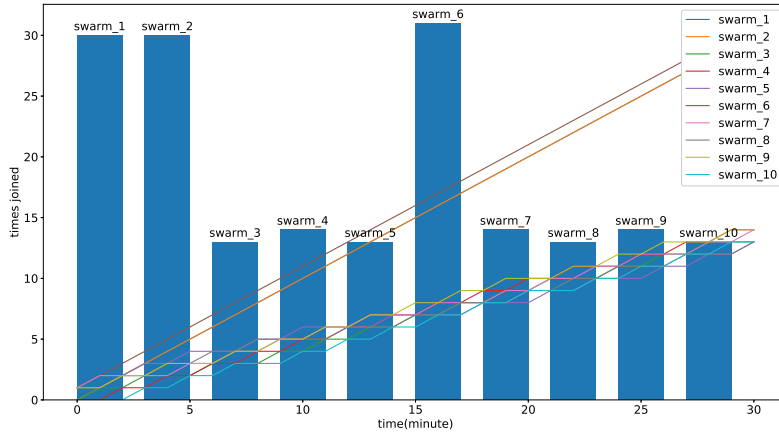
26

Figure 4.3: The number of times each swarm is joined.

experiments, this swarm could be either swarm 6, swarm 7 or swarm 8.

4.4 Real-world experiment

The real-world experiment focuses on analyzing how many resources Credit Mining will use in a real-world scenario and its performance. This experiment is executed on the test channel with 25 most popular torrents at that time from a public torrent-discovery website on the Internet. We allow at most 10 downloads active at the same time and apply the policy every 5 minutes.

The experiment lasts from March 7, 2018, 18:10 GMT to March 8, 2018, 13:20 GMT. The system resource usage is displayed in Figure 4.4. Note that since the CPU usage is about 0.1%, it appears that Credit Mining along with Tribler does not consume much CPU. Memory usage is always under 100MB. However, we observed that there are several unusual slight stepping in RAM usage during the experiment. This cannot be explained from the Credit Mining mechanism or the experiment log file. It might be the result of some undetected memory leak in the Tribler core.

The total payload upload and total download curves are shown in figure 4.5. Credit Mining seeded 300GB during the whole experiment. The upload speed keeps increasing during the first hours after Credit Mining is turned on. This partly because we first need to download data, before we can upload it. Also, this is due to the feature that Credit Mining start with random swarms and then replaces them with better-performing ones. From figure 4.5 speed slowed down after seeding for 7 hours. We assume the reason is that the demand from these swarms is mostly fulfilled already. If Credit Mining is running with a more extensive and dynamic mining pool, the speed might increase after Credit Mining finds the better-

27
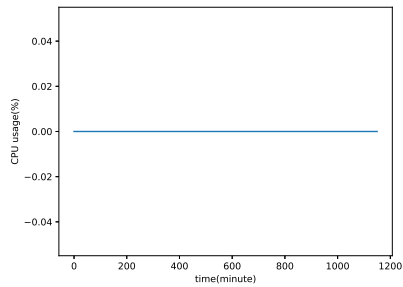
(a) CPU usage during the experiment.
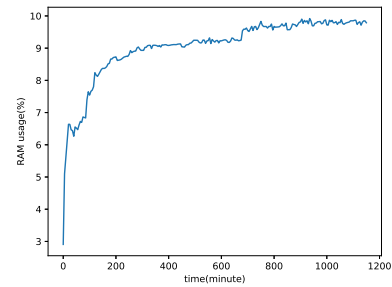
(b) RAM usage during the experiment.

Figure 4.4: System resource usage during the experiment.

performing ones to replace the mostly fulfilled ones.

The total upload fails to catch up with total download before they both slow down. Although upload exceeds download eventually, the trend appears not profitable enough, and the time consumption is too long to meet our requirement.

4.5   Lessons learned

Our real-world experiment proves our ability to conduct credit mining. However, the yield we obtained is not positive enough to meet the requirements. The key factors are determined to be the failure in reducing the bandwidth cost in the initial investment. Although we only give 5 minutes for the initial investment of each swarm, if the swarm is over-seeded, we might have already downloaded much from the swarm before the swarm is detected to be not profitable. Based on the Tribler codebase, we refactor the Credit Mining core to apply the lessons learned, aiming to achieve positive yield for the first time.
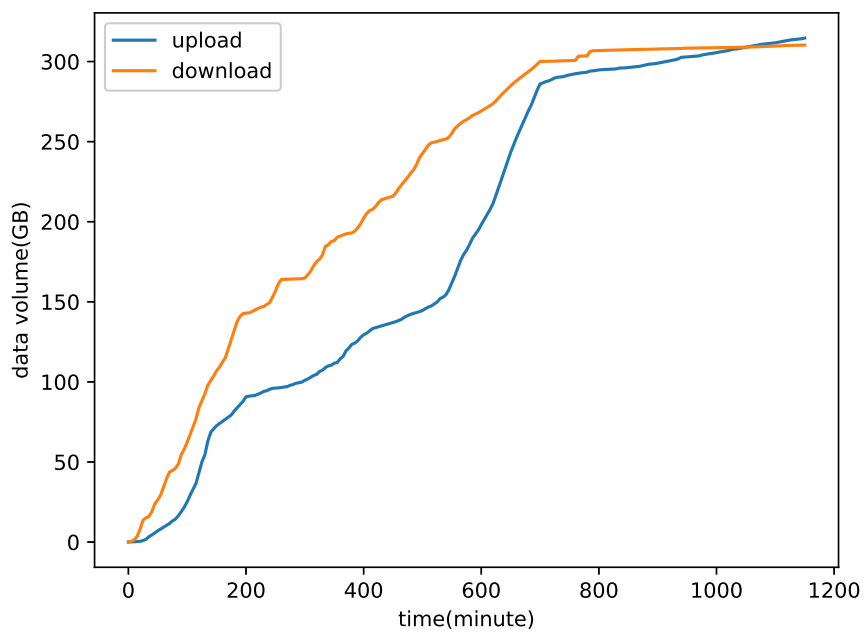
Figure 4.5: The payload upload and download performance with real-world swarms.

Chapter 5

Advanced Experiment and Evaluation

While conducting the experiments, it becomes clear that without any prior knowledge of the swarms, it is hard for us to implement a policy with enough positive yield. The policy can distinguish the most profitable swarms but with a tremendous amount of failed investment. In this chapter, we are to apply the knowledge and experience we learned from these initial experiments and modify the policy to improve the performance further.

The key insights and experimental results will be presented in the following sections, each improved from another, growing more complex and seeking more efficiency.

5.1 One-shot experiment with recent swarms

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

5.1.1 Policy design

To further avoid wasting download bandwidth in the initial investment phase, we set a limit to invest at most 25 megabytes to each swarm in this phase. Once the limit is reached, the download task will be switched to upload-only mode to avoid over-investing. After three minutes we rank all the swarms by their total upload so far. The top 3 swarms are invested with additional download bandwidth, and dead or unprofitable swarms remain choked by the 25-megabyte initial investment limit. The workflow is shown in Figure 5.1.
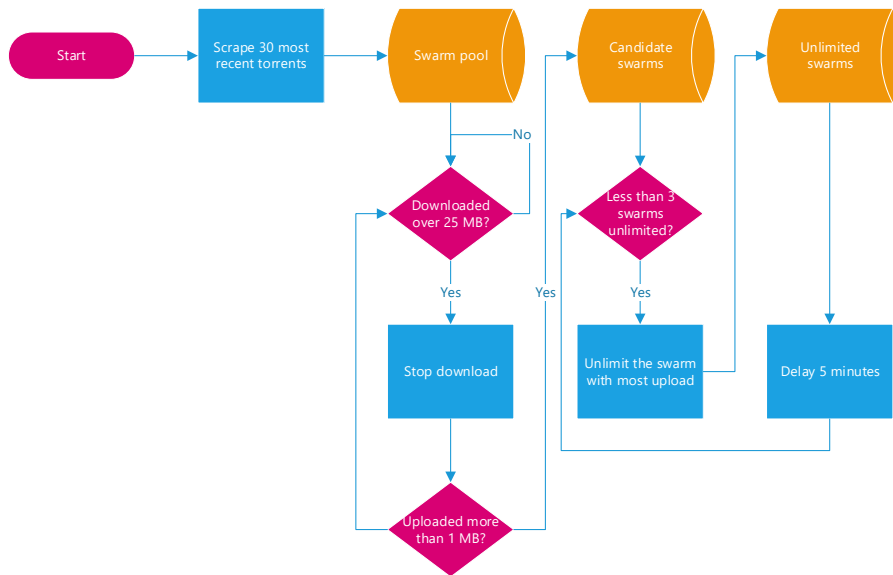
Figure 5.1: Workflow of the one-shot experiment with the recent swarms.

5.1.2 Test subjects

We experimented with 30 torrents from a famous torrent-discovery website fetched by their magnet links. The experiment lasted from July 17, 2018, 08:31 GMT to July 17, 2018, 12:12 GMT.

5.1.3 Experimental result and analysis

The result of the experiment is shown in Figure 5.2 and Figure 5.3. Figure 5.2 shows the number of swarms in each category over time. "Swarms added" counts the swarms which are added for the initial investment. "Swarms active" counts the swarms which have non-zero total download. "Swarms investigated" counts the swarms reaching the 25-megabyte investment limit in the initial investment phase, which then become candidates to be given additional investment.

From the figure 5.2, we can observe that there is a slight difference in "swarms added" and "swarms active". The swarms we added are the most recent ones fetched from a general torrent-discovery website, which also means they are fairly random in all aspects other than creation time. The difference is created by the swarms whose torrent files are not reachable and the swarms that have no peers. We also observe that the percentage of "swarms investigated", which are swarms invested with no less than 25 megabytes download, is also quite low. This shows that there is a high percentage of dead swarms on the Internet. These swarms will not provide any profit to Credit Mining.
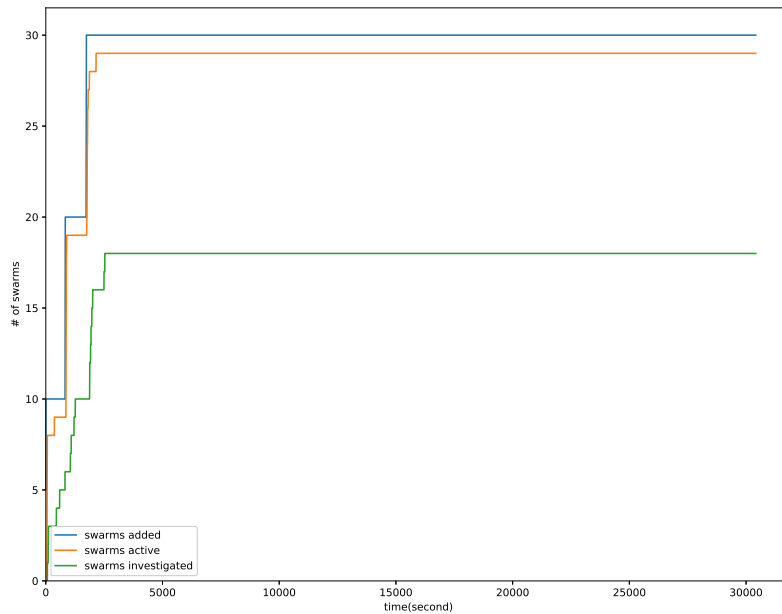
Figure 5.2: Number of swarms in each category over time in the one-shot experiment.

Figure 5.3 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 investment limit. After that, the upload can soon catch up with and eventually exceed the download.

In this policy, we only have one chance to decide to which swarms we shall give additional investment. It might fail to invest the most profitable swarms if they cannot outperform others before we rank them. Also if a swarm is outstanding in the first few minutes, it will be given the full amount of additional investment, regardless of its performance afterward. In our repeated experiences, the additionally invested swarms are not always providing positive yield.

To reduce the risk of issuing additional investment to unprofitable swarms and be more tolerant to the profitable swarms, we proposed a "three-level promotion" policy in the next section.
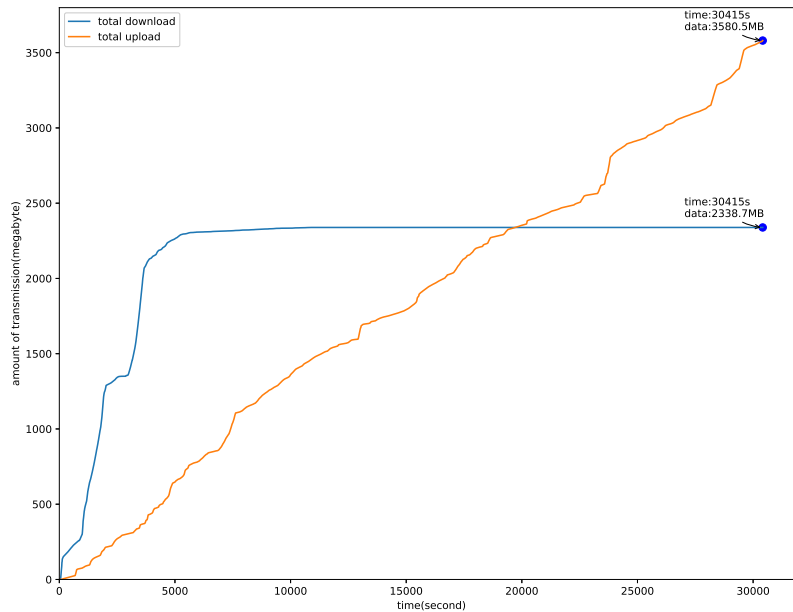
Figure 5.3: Total data transmission over time in the one-shot experiment.

5.2 Three-level promotion policy

5.2.1 Policy design

In this policy, instead of directly give the full additional investment, we "promote" a swarm to the next level and give it a bit more additional investment if this swarm outperforms others at the same level. We hope to increase the efficiency of the additional investment phase by dividing it. At each level, we first download x megabytes and then switch to upload-only mode. Periodically, we rank the invested swarms at each non-top level respectively by their upload and promote the best at each level to the next level. Only the swarms which have uploaded more than y megabytes are taken into this ranking to avoid wasting additional investment on over-seeded swarms.

In this policy, we define three levels. Level 1, with $x = 25$ and $y = 1$, level 2, with $x = 250$ and $y = 25$, and the level 3 with $x = 1024$ and $y = \infty$ since it do not need to be promoted any more. Swarms fail to fulfill the x in level 1 are specially labeled as level 0, since these are mostly dead swarms. Considering the limited storage, we set the maximum number of all invested swarms to 1000, level 2 swarms to 100, level 3 swarms to 25. Level 1 and 2 swarms are ranked by upload every five minutes, after which the top swarm at each level is promoted to the
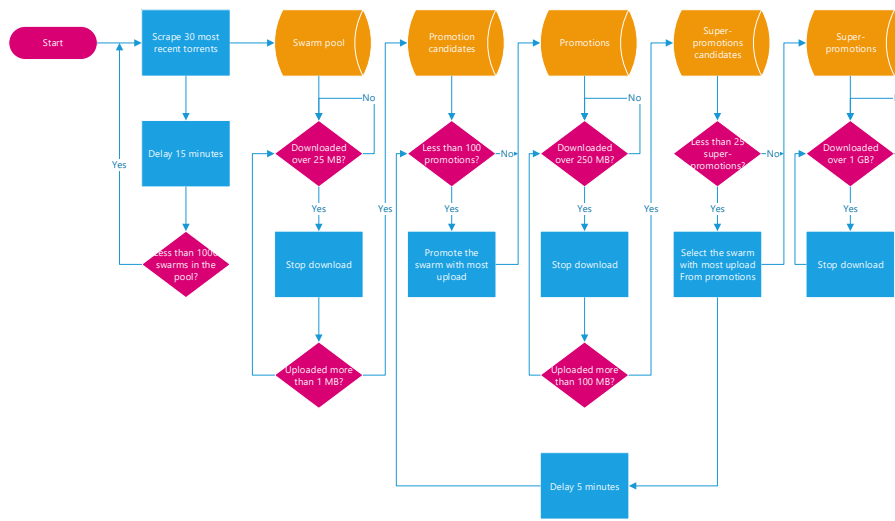
34

Figure 5.4: Workflow of the three-level promotion policy.

next level giving a bit more additional investment. We also add swarms to Credit Mining periodically instead of entirely to maintain the freshness of the swarms. The workflow is displayed in Figure 5.4.

5.2.2 Test subjects

We experimented with the torrents from the same website. The experiment lasted from July 24, 2018, 07:26 GMT to July 26, 2018, 11:48 GMT.

5.2.3 Experimental result and analysis

The results are shown in Figure 5.5. It is visible that the download and the upload curves show a very similar pattern to the result from the 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 significantly increased both in speed and in the total amount. Though there is no dramatic increase in upload/download ratio, the net upload gain(upload-download) is increased dramatically, which is on the right direction since the profit we earn is measured in the total bandwidth contribution rather than the ratio.

5.3 Multi-level promotion policy

5.3.1 Policy design

To boost the efficiency of every byte of our bandwidth and storage, based on the three-level policy, we attempt to subdivide the additional investment further and
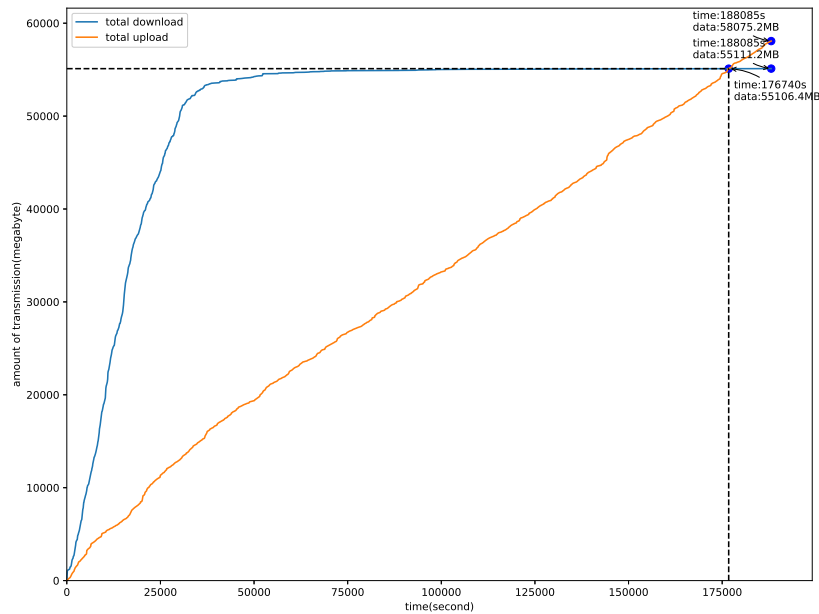
Figure 5.5: Total data transmission over time in the three-level promotion policy.

set more aggressive requirement to promotion. In this experiment, considering the relatively high percentage of dead and over-seeded swarms on the Internet, the *x* value of level 1, which is the initial investment, is reduced to 5. The *x* values for level 1 to 10 are respectively 5, 8, 12, 18, 27, 40, 60, 90, 135 and 200. There are 11 levels in total including the initial investment, the additional investment, and the swarms which failed to complete the initial investment. To further avoid the promotion of unprofitable swarms, we set the requirement to promote a swarm to reaching 1.0 upload/download ratio making $y = x$ at every level. In other words, one swarm must earn back the tokens spent on investing itself until it is qualified to consume more resources and be further invested. The workflow is shown in Figure 5.6.

5.3.2   Test subjects

We experimented with the torrents from the same torrent-discovery website as the previous experiments. The experiment lasted from August 1, 2018, 15:39 GMT to August 5, 2018, 14:54 GMT.
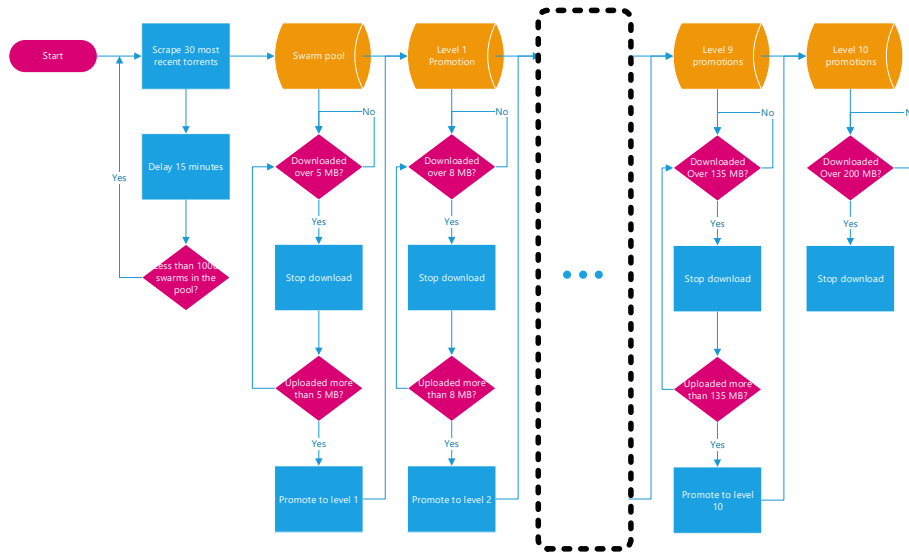
36

Figure 5.6: Workflow of the multi-level promotion policy.

5.3.3 Experimental result and analysis

Figure 5.7 shows the data transmission over time in this experiment. We can observe that with the help of the more aggressive requirement to the promotions that every invested swarm must pay its debt, the yield shows positive nearly from the beginning. We also visualize the upload gain from all the invested swarms at different levels over time in Figure 5.8. It clearly shows that the majority of the profit comes from the swarms at the highest level. Although invested swarms at other levels can provide negative yields in general, they are suppressed since they can be detected earlier in the early phases before we waste much bandwidth on them. In our experiment, most of the negative-performing swarms are detected before exceeding the initial investment phase, which cost only 5 megabytes each at most, minimizing the impact on the overall profit.

We also analyze the storage usage in Figure 5.9. It is evident that the level 10 swarms, which are the most profitable ones, use more than half of all storage. Besides level 10, level 1, 8 and 9 swarms also consume some noticeable shares of the storage. Level 1 swarms are no more than 5 megabytes each but still takes 10.13% of all the storage. This also proves that most valueless swarms are suppressed in the initial investment phase. Level 8 and 9 provide 10.27% and 3.07% of the profit level 10 provides, but takes 21.56% and 15.64% of storage level 10 takes. If we could suppress these swarms earlier, additional storage can be saved for more profitable swarms.
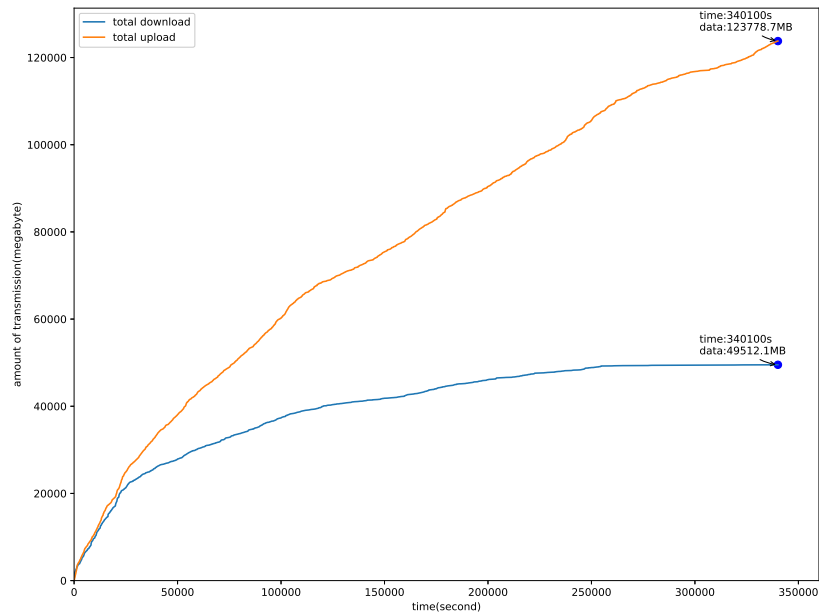
Figure 5.7: Total data transmission over time in the multi-level promotion policy.

5.4 Greedy and continuous policy

To further increase the storage efficiency, based on the multi-level policy, we try configuring a more aggressive upload/download ratio requirement for promotion, as well as detecting and replacing the exhausted or dead swarms. We expect to get a deliberately aggressive policy with continuous operation.

The promotion system is based on the multi-level policy we discussed in Section 5.3, but with a different ratio requirement for promotion. Instead of a fixed 1.0 share ratio, the requirement for promotion linearly increases with the level. The share ratio requirement is 2.0 at level 1, 3.0 at level 2, and all way up to an extreme ratio of 10.0 at level 9. By increasing the difficulty of putting additional investment into a particular swarm, we expect to see fewer but elite swarms reach the highest level and make more profit with the same amount of storage.

Another addition to our final experiment is the mechanism to retire exhausted investment. This ensures we regularly clear our storage and obtain a continuous running algorithm. At five points in the lifetime of the investments, we evaluate their yield and remove the under-performing investments. An aggressive mechanism requires a swarm to produce a profit of at least 5 megabytes upload after 1 hour, 50 megabytes after 12 hours, 100 megabytes after 24 hours and 250 megabytes after 48 hours. Otherwise, this swarm will be removed. We also remove any swarm that
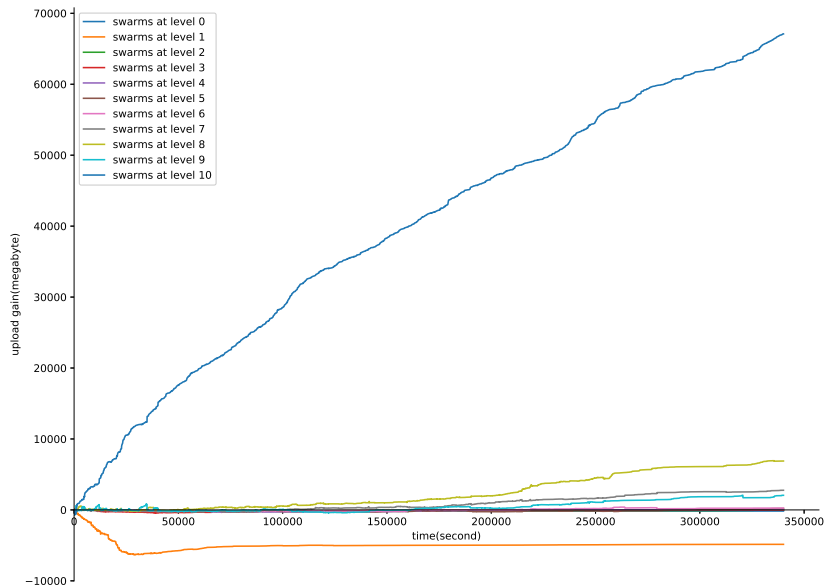
38

Figure 5.8: Upload gain from different level overtime in the multi-level promotion policy.

is invested for over 72 hours. This mechanism should deal with dead swarms and unpopular swarms.

We experimented with torrents from the same website as the previous experiments. The experiment lasted from August 3, 2018, 08:58 GMT to August 10, 2018, 13:48 GMT. The results in Figure 5.10(a) are surprisingly disappointing. Instead of improving, the new policy becomes less profitable that total upload is almost identical to total download. We suspect that 1 hour is too aggressive for the first deletion, giving up our investment too quickly.

To offer more opportunity to each swarm, we try to improve the configuration from two aspects. In the first one, we release the first deletion time to 6 hours, giving every swarm a better chance to survive the first deletion check. In the other experiment, we lower the share ratio requirement to the same level of Multi-level Policy in Section 5.3, which is 1.0 for all levels. The experiments last from August 15, 2018, 09:05 GMT to August 19, 2018, 10:26 GMT and from August 10, 2018, 06:01 GMT to August 14, 2018, 15:29 GMT respectively. Their total transmission over time is shown in Figure 5.10(b) and Figure 5.10(c). We can see that in the first experiment of these two, the curves show a similar pattern to the previous experiment, but the average upload and download speed are far less. This should be due to the fact that the promotion requirement is too aggressive that swarms are not
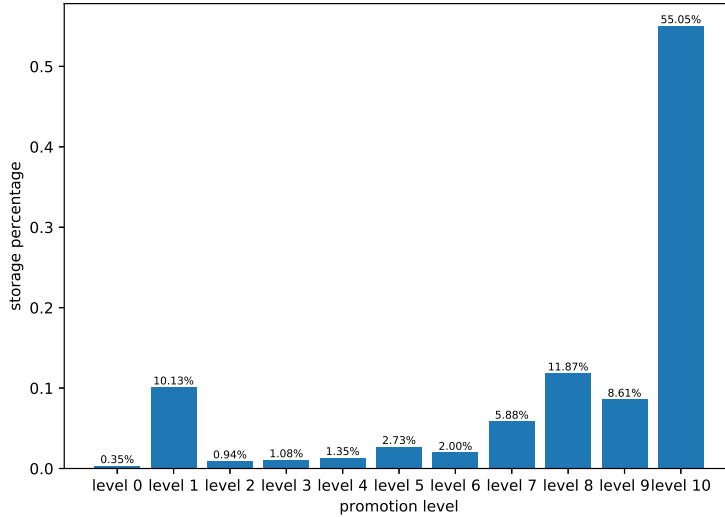
Figure 5.9: Percentage of storage usage of investigations in different level in the multi-level promotion policy.

easily promoted, thus lack of enough content to upload. This is against the desire of Credit Mining.

The other experiment also shows a similar pattern to other experiences in this section, but the yield is negative. We do observe that average upload speed is relatively high compared with other policies. However, since removal is too rapid, we failed to earn but lose profit after the investment.
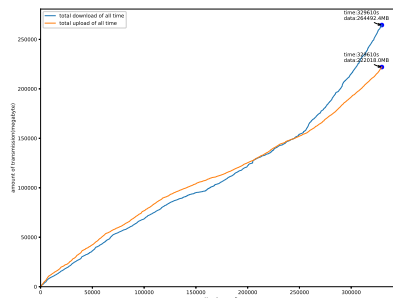
The lesson we learn is that periodical deletion is causing more impact on the profit than we expected since the majority of swarms on the Internet are not profitable at all. Every profitable swarm we find with the "shotgun methods" should be valued and handled wisely.
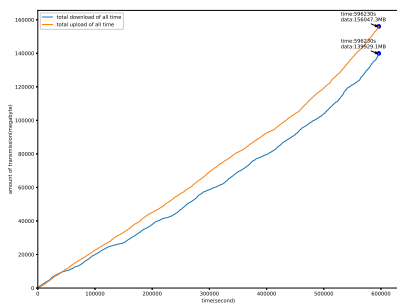
(a) Aggressive policy
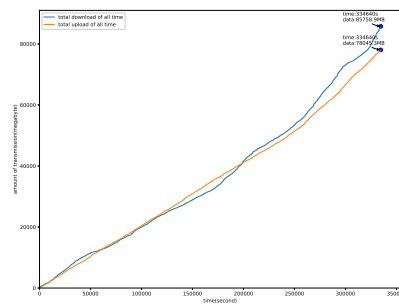
(b) Aggressive policy with longer deletion time.

(c) Aggressive policy with lower promotion requirement.

Figure 5.10: Total data transmission over time in the aggressive polices.

41

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Credit Mining is to solve the problem of finding the most profitable subset of swarms out of a much larger pool of swarms, with profit meaning the effective number of bytes contributed to the swarm. Previous work had either scalability problems or had acceptable performance only in synthetic benchmarks. In our implementation, built on top of the Tribler file-sharing client, we achieved a platform that is both workable and scalable on real-world swarms.

In order to find a suitable swarm selection policy, we first tried a basic policy based on maximizing the total upload. Unfortunately, this policy did not perform well. Once we realized that the lack of profitability was due to the fact that many swarms were dead or over-seeded, and also that we first needed to wait for a return on investment (i.e., we needed to wait for the upload to catch up with the download), we were able to improve our algorithm. To our knowledge, this is the first time that such an algorithm achieved a profit using real-world swarms.

We implemented and evaluated a number of additional policies, most of which resulted in a profit. The most suitable for Credit Mining so far is the multi-level promotion policy. There is still space to optimize the performance further, especially after we can achieve swarm size information from Tribler in the future.

6.2 Future work

6.2.1 Optimizing policies

Currently, the most optimal policy is yet to be found. All our policies must be running in a community where there is a reasonable percentage of active under-seeded swarms. Otherwise, the policy will waste too much time traversing across the swarms before it finds the demanded ones. Also, if Credit Mining is restarted, it will 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. We plan to get a database of download performance and swarm information and train the model on this database. However, there is currently no such database in existence, and we lack the time and resources 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[6] to make it realistic. Thus we decide to leave this to the future, after the completion of this sub-system.

6.2.2 Game among the Credit Miner

Another problem is that Credit Mining currently regards all other peers in every swarm as normal peers. However, it is possible that multiple miners are joining the same swarm. In that case, transferring data between the miners is not as beneficial to the swarm. Also if too many miners investigate 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.

6.2.3 Sybil attack

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

6.2.4 Improved token economy

The last problem is that our policies only focus on the total upload amount of the swarms. The miners should be also be rewarded for other contributions, such as keeping the availability of the dying swarms. The policies should be updated to keep this into account once Tribler starts to reward such behaviors.

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] Free-ride policy: give users with lower token balances less priority at the exit nodes. `https://github.com/Tribler/tribler/issues/3573`. (Accessed on 09/17/2018).

[4] Github - tribler/tribler: Privacy enhanced bittorrent client with p2p content discovery. `https://github.com/Tribler/tribler`. (Accessed on 02/25/2018).

[5] Implemented tunnel payouts. `https://github.com/Tribler/tribler/pull/3492`. (Accessed on 09/17/2018).

[6] Swarm size community: content popularity · issue #2783 · tribler/tribler. `https://github.com/Tribler/tribler/issues/2783`. (Accessed on 02/24/2018).

[7] Towards global consensus on trust. `https://github.com/Tribler/tribler/issues/3357`. (Accessed on 05/07/2018).

[8] Tribler: an attack-resilient micro-economy for media. `https://github.com/Tribler/tribler/wiki`. (Accessed on 05/07/2018).

[9] A trustful blockchain-based token economy to prevent bandwidth free-riding. `https://github.com/Tribler/tribler/issues/3337`. (Accessed on 06/09/2018).

[10] What is credit mining? - general - tribler discussion forums. `https://forum.tribler.org/t/what-is-credit-mining/3692`, October 2016. (Accessed on 08/08/2018).

[11] The bittorrent protocol specification. `http://www.bittorrent.org/beps/bep_0003.html`, February 2017. (Accessed on 05/16/2018).

[12] Tribler 7.1 mining credits preview - general - tribler discussion forums. `https://forum.tribler.org/t/tribler-7-1-mining-credits-preview/4356`, February 2018. (Accessed on 08/08/2018).

[13] Eytan Adar and Bernardo Huberman. Free riding on gnutella. 5, 04 2001.

[14] M. Capotă, J. Pouwelse, and D. Epema. Decentralized credit mining in p2p systems. In *2015 IFIP Networking Conference (IFIP Networking)*, pages 1–9, May 2015.

[15] 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.

[16] Bram Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72, Aug 2003.

[17] 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.

[18] 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-47b0-9d57-8e605eaaaed1/datastream/OBJ/download, March 2017.

[19] D. Hughes, G. Coulson, and J. Walkerdine. Free riding on gnutella revisited: the bell tolls? *IEEE Distributed Systems Online*, 6(6), June 2005.

[20] 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).

[21] 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 Prasun Sinha, editors, *Distributed Computing and Networking*, pages 270–284, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[22] Michael W Macy and Yoshimichi Sato. Trust, cooperation, and market formation in the us and japan. *Proceedings of the National Academy of Sciences*, 99(suppl 3):7214–7220, 2002.

[23] Kenneth Newton. Trust, social capital, civil society, and democracy. *International Political Science Review*, 22(2):201–214, 2001.

[24] Arvid Norberg. libtorrent manual. `https://www.libtorrent.org/features.html`. (Accessed on 02/24/2018).

[25] 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).

[26] 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 and Computation: Practice and Experience*, 20(2):127–138, 2008.

[27] Paul Slovic. Perceived risk, trust, and democracy. *Risk analysis*, 13(6):675–682, 1993.

[28] John L Sullivan and John E Transue. The psychological underpinnings of democracy: A selective review of research on political tolerance, interpersonal trust, and social capital. *Annual review of psychology*, 50(1):625–650, 1999.

[29] Piotr Sztompka. Trust and emerging democracy: Lessons from poland. *International sociology*, 11(1):37–62, 1996.

[30] Yichi Zhang, Lingfeng Wang, Weiqing Sun, et al. Trust system design optimization in smart grid network infrastructure. *IEEE Trans. Smart Grid*, 4(1):184–195, 2013.

[31] Lynne G Zucker. Production of trust: Institutional sources of economic structure, 1840–1920. *Research in organizational behavior*, 1986.