Decentralized Credit Mining in P2P Systems

Capota, Mihai; Pouwelse, Johan; Epema, Dick

DOI
10.1109/IFIPNetworking.2015.7145334

Publication date
2015

Document Version
Accepted author manuscript

Published in
2015 IFIP Networking Conference

Decentralized credit mining in P2P systems

Mihai Capotă
Delft University of Technology
The Netherlands

Johan Pouwelse
Delft University of Technology
The Netherlands

Dick Epema
Delft University of Technology
The Netherlands

***Abstract*—Accounting mechanisms based on credit are used in peer-to-peer systems to track the contribution of peers to the community for the purpose of deterring freeriding and rewarding good behavior. Most often, peers earn credit for uploading files, but other activities might be rewarded in the future as well, such as making useful comments or reporting spam. Credit earned can be used for accessing new content, or for receiving preferential treatment in case of network congestion. We define credit mining as the activity performed by peers for the purpose of earning credit. In this paper, we design, implement, and evaluate a system for decentralized credit mining that maximizes the contribution of idle peers to the community by automatically uploading popular files. Building on previous theoretical insights into the economics of communities, we select autonomous algorithms for bandwidth investment as the basis of our credit mining system. Additionally, we describe our experience with important challenges arising from Internet deployment, that are frequently neglected in emulation, including duplicate content avoidance, spam prevention, and the cost of keeping peer information updated. Furthermore, we implement an archival mode of operation, which prevents the disappearance of old content from the community. We show the feasibility and usefulness of our credit mining system through measurements from our implementation on top of Tribler, an Internet-deployed peer-to-peer system.**

I. Introduction

Incentives are key to the functioning of peer-to-peer (P2P) systems. Altruism does lead people to contribute resources even when they are not rewarded, but only to a certain extent. Consequently, systems that reward contribution through built-in incentives have historically proven to be more successful than those with no incentives. For example, BitTorrent [1] displaced Gnutella [2] as the most used P2P file-sharing system soon after its introduction thanks to built-in incentives which effectively discourage freeriding.

More recently, private P2P communities have extended the incentive mechanisms embedded in BitTorrent by employing accounting mechanisms that track user activity. These private communities incentivize good user behavior by employing credit (or sometimes *sharing ratio enforcement*) and are also successful at increasing user contribution, as we previously showed [3]. At the same time, the accounting mechanisms have made it difficult for honest, non-freeriding, but inexperienced users to earn enough credit (or keep a high enough sharing ratio) to maintain community membership [4].

Tribler [5] is a P2P system that uses a decentralized form of accounting called BarterCast [6]. Tribler is backwards compatible with BitTorrent and uses many of the concepts introduced by the latter: content to be shared is divided in *pieces* that are cryptographically hashed; the hashes are collected in a *torrent* file which is used to uniquely identify the content; the group of peers sharing a torrent is a called a *swarm*; servers called *trackers* can be used by peers when joining a swarm to find other peers.

In this paper, we design, implement, and evaluate using Internet experiments a decentralized Credit Mining System (CMS) aimed at helping users contribute to Tribler. The goal of the CMS is simple: earn credit on behalf of the user—without requiring user intervention—by contributing upload bandwidth to the community. The user can then spend the credit earned, for example, to obtain a fast download speed while downloading new content in case of a flash crowd [7]. (Tribler peers can rank download requests in order of requester credit, and can give priority to the requests of the peers with the most credit.)

Our CMS is completely decentralized—it is part of the Tribler P2P client and does not require the collaboration of any other Tribler peer to function. The operation of the CMS can be seen as a sequence of three steps. First, the user selects a source of swarms for the CMS to take into consideration for credit mining. This is a form of white-listing and ensures the user has control over the content shared through their computer. Second, the CMS periodically selects a subset of swarms for active credit mining. The user may provide a large number of swarms and it is not technically feasible for the CMS to actively participate in all of them. Third, the CMS joins the selected swarms and attempts to maximize earned credit by downloading as little as possible and uploading as much as possible.

Long-term content availability is a problem in P2P systems, caused by the gradually falling user demand for old content [8]. This also makes credit mining old swarms inefficient. However, users may want to improve the avail-

ISBN 978-3-901882-68-5 © 2015 IFIP

ability of old content and we provide a special mode of operation for the CMS to help them. In *archival mode*, the CMS selects swarms not based on upload potential, but on the number of replicas present in the system.

In addition to the functionality outlined above, we include in our CMS design several subsystems aimed at tackling challenges arising from implementing and deploying the CMS over the Internet. We identify duplicate content and select for credit mining only the swarm with the most peers. We detect spam using collaborative filtering. Finally, we optimize the network traffic necessary to maintain up to date information on the swarms in the CMS.

We implement our design in Tribler and deploy it over the Internet, using swarms from a real-world BitTorrent community. In our evaluation, we explore several parameters that influence the functioning of the CMS and use the results to select default values that lead to good performance. Furthermore, we verify the feasibility of widespread adoption of the CMS through an experiment where a high proportion of peers use it.

The contributions we make in this chapter are the following:

1) We design CMS, a credit mining system for Tribler, which automatically selects from a whitelist of swarms provided by the user the swarm with the best upload potential, and joins this swarm with the goal of maximizing its upload/download ratio (Section III);
2) We solve problems arising from deploying CMS on the Internet by detecting duplicate content, removing spam, and minimizing overhead network traffic (Section IV);
3) We implement CMS in Tribler and test it with real-world swarms, showing its credit mining effectiveness, and proving its compatibility with widespread community deployment (Sections V and VI).

II. Background

In this section, we describe accounting in P2P systems, focusing on centralized, as well as decentralized solutions. We also introduce Tribler, the system we use for the implementation of the CMS.

A. Accounting and credit in P2P systems

Accounting mechanisms are widespread in P2P systems because they incentivize users to contribute. Arguably the most successful P2P system, BitTorrent, uses tit for tat [1], a pairwise accounting mechanism where peers keep track of their interaction with other peers and reward the peers with the biggest contribution. However, tit for tat does not incentivize long-term contribution, beyond the transfer of a single file.

Several BitTorrent communities use centralized accounting to track the overall contributions of users over time [3]. BitTorrent clients report the upload and download to the community servers such that each user has an associated *sharing ratio*—the ratio between upload and download. Communities employ *sharing ratio enforcement* to provide certain privileges, like access to the newest content, only to the users with sharing ratios above certain thresholds. The sharing ratio is a form of credit, and in certain communities it can be transferred between users.

Decentralized accounting mechanisms spread the burden of tracking peer contributions from a centralized server to the individual peers in the network [9]. Decentralized accounting mechanisms that require every peer to store the complete and up-to-date contribution information of all peers are also suffering from limited scalability. Instead, BarterCast [6] uses only the local view of each peer to compute the relative contribution of other peers by applying flow network techniques on the peer interaction graph.

Accounting is used to provide benefits to users that earn credit. In certain private communities, users are required to maintain a minimum credit balance, otherwise they are expelled from the community [3]. In other communities, new content is first made available to users with sufficient credit, and only later to the other users.

B. Tribler

Tribler is a P2P system developed at the Delft University of Technology and released as open-source software. It is based on a custom protocol, called *Swift* [10], but is also compatible with BitTorrent. In addition to file transfer, it provides collaborative wiki-style editing, decentralized search, and integrates the BarterCast accounting mechanism.

As opposed to BitTorrent, where users have no long-term identifiers, each Tribler user is assigned a permanent identity, called *PermID*, which is used by BarterCast and other subsystems. The PermID is actually an automatically generated public key that also enables Tribler to encrypt the communication between peers. At the same time, each data transfer between two Tribler peers generates a BarterCast record which is cryptographically signed by both peers. BarterCast records provide unforgeable proof of contribution and are the basis of the Tribler accounting mechanism.

Tribler gives users the possibility to publish their own content in a decentralized way. Any Tribler user can create a personal channel and add content to it. Other users can subscribe to the channel and receive new content as it is published. The P2P infrastructure for publishing and subscribing to channels, as well as collaborative editing and other Tribler features is provided by *Dispersy* [11], a generic message synchronization system. Dispersy uses Bloom filters to enable peers to exchange messages in a P2P network under challenging conditions, including high churn and lack of end-to-end connectivity.

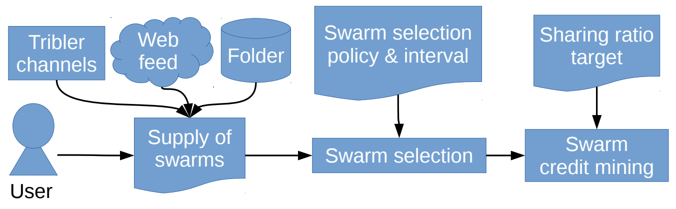


Fig. 1. An overview of the three steps of the credit mining process.

Applications running on top of Dispersy define the message semantics—Dispersy is only concerned with synchronization. For example, Tribler uses several type of messages to implement channels, such as a message for creating a channel, a message for adding content to a channel, a message for removing content from a channel, etc.

Each peer adds all messages to a Bloom filter and periodically exchanges the Bloom filter with other peers to determine what new messages must be exchanged. For example, if peer A is subscribed to a channel, when A and another peer, B, exchange Bloom filters, if A observes in the Bloom filter of B a message adding new content to the channel, A will request the message from B, thereby discovering the new content.

III. Credit Mining System design

In this section, we describe the design of the CMS. Considering the three steps involved in the credit mining process, depicted in Figure 1, we discuss first the supply of swarms to the CMS, second, the algorithm for selecting swarms for investment, and third, the behavior of the CMS within a swarm. Finally, we describe the design of the archival mode of operation of the CMS.

A. Supply of swarms

We allow the user to select the swarms that are to be credit mined by the CMS, thereby creating a form of white-list. In other words, we make the credit mining process *opt-in*: while the CMS is autonomous and does not require user input, it does require a source of swarms to start. The user can supply swarms to the CMS using three sources, which we describe in turn.

The first source is a list of Tribler channels selected by the users for boosting. Whenever the user wants to support a channel, without necessarily downloading the content shared through the channel, they can add the channel to the CMS so that its swarms are taken into consideration.

The second source is a web feed coming from a web server, which allows the CMS to access swarms as soon as they are created. Web feeds are a popular means for communities to distribute swarms and we have used them before successfully for this purpose [3].

The third source is a folder containing torrent files. This can be a folder with torrents downloaded by the user from the web, or it can be the folder with torrents collected by Tribler through gossiping during normal operation. When two Tribler peers meet, they exchange lists of swarms to determine the similarity between them, a feature useful for search. At the same time, the torrents corresponding to the swarms are saved in a local cache, which is accessible to the user.

B. Swarm selection

The number of swarms available through a source is not bounded, so the CMS has to explicitly select a subset of *active* swarms to which to dedicate the bandwidth and storage resources available. Furthermore, swarm selection is a periodic process, because the characteristics of swarms change constantly and the CMS must be able to improve the credit mining by changing the set of active swarms.

The design of the CMS includes a pluggable policy for swarm selection. We have previously studied methods to predict the upload potential of swarms using simulation [12]. We have shown that a multivariate adaptive regression splines (MARS) model can select swarms with good upload potential for the majority of peers in our simulation. At the same time, we obtained positive results for the same simulation setup using a simpler swarm selection policy, namely selecting the swarm with the lowest proportion of seeders. Because the regression model we used for simulation is difficult to integrate into Tribler, in this paper we use simpler policies, which still give good results, as we will show in Section VI.

The first policy is the *seeder ratio* (SeederRatio) policy, which selects for credit mining the swarms with the lowest ratio of seeders to all peers (seeders and leechers). Intuitively, this policy selects the most undersupplied swarms, i.e., the swarms with the least bandwidth supply—seeders—compared to the total bandwidth demand—leechers.

The second policy is the *swarm age* (SwarmAge) policy, which selects swarms based on their age. Previous work has shown that the age of a swarm plays a significant role in the potential for upload [8]. Specifically, the newest swarms offer the best potential. Most users will download any content only once or a few times, and there is a bounded number of users. It follows that there is a bounded number of times any content will be downloaded in total. So it is expected that an early download will have more potential for upload—because of subsequent downloads from other peers—than late downloads.

The third policy, *Random*, is used mainly to as a baseline for evaluating the first two policies, but also to stress test the other components of the CMS. Using the Random policy, the CMS selects swarms for credit mining uniformly at random from the swarms available.

C. Behavior within a swarm

The behavior of the CMS within a swarm is crucial to the success of credit mining. The goal of the CMS when joining a swarm is the same as its global goal: to upload as much as possible while downloading as little as possible in order to maximize the credit gained.

In previous work, we studied the theoretical properties of in-swarm behavior algorithms through a mathematical model [13]. We also identified a heuristic that provides the desired in-swarm behavior. The heuristic, introduced by the Libtorrent open-source software library [14], downloads the content in a swarm one piece at a time and only downloads additional pieces when it estimates there is enough potential for upload in the swarm.

The algorithm is governed by one parameter, the sharing ratio target, which represents the desired ratio between upload and download for swarms in the CMS. A value of 3 for this target—the default in Libtorrent—means that the CMS will only download pieces from a swarm if it estimates it will be able to upload it back to at least 3 other downloaders in the swarm. Furthermore, once a piece is downloaded, the CMS waits for it to actually be uploaded 3 times before it proceeds to download other pieces. This condition acts as a fail-safe mechanism for the situation when the upload potential estimation is wrong, and guarantees that in the worst case the loss of the CMS in a swarm is equal to the size of one piece.

A crucial characteristic of the algorithm is its resilience to widespread deployment in the community. If two CMS peers meet, they recognize each other through a flag present in the BitTorrent handshake message. Thus, the two CMS peers do not needlessly upload and download data from each other, preventing the waste of resources.

D. Archival mode

Previous work has shown that it is more advantageous to seed new files than old files for earning credit in a community[8]. At the same time, users are frequently interested in the long term availability of certain files. Therefore, we provide a special mode of operation for the CMS, called archival mode.

All swarm sources—channels, web feeds, and folders—are compatible with archival mode. When the users adds a new source to the CMS, they can mark it for archival. When a swarm is in archival mode, the CMS will ensure that it is always seeded by a minimum of two seeders—the minimum necessary to provide fault tolerance. In practice, if the CMS observes that an archival swarm has two or less seeders, it downloads and seeds it. Whenever the number of seeders raises again above three, the CMS pauses seeding, while continuing to monitor the swarms through trackers scrapes.

Archival mode swarms generate less credit than the normal swarms in the CMS. However, they always get priority over normal swarms because the user explicitly asked the CMS to seed them by using archival mode.

At the same time, the CMS does not necessarily seed all archival mode swarms at all times. When other peers are seeding, the CMS pauses seeding and starts instead uploading in normal swarms which have the potential to generate more credit. Consequently, archival mode offers more flexibility than simply having the user seed swarms in Tribler outside of the CMS.

TABLE I
Duplicate content example: `ubuntu-14.04-desktop-amd64.iso` shared in different swarms because of different piece sizes and private flag usage

Hash	Piece [B]	Flag
`18AC50D74C61883B3AB4C40F5DD3E35F157DE1A2`	1 048 576	N/A
`4D753474429D817B80FF9E0C441CA660EC5D2450`	524 288	N/A
`F88ED0C16CF7F452A5C737A0B7503F925E11FE00`	524 288	0

IV. Internet deployment challenges

In this section, we focus on several issues tangential to credit mining that are crucial for a system deployed on the Internet.

A. Duplicate content detection

In many P2P communities where the addition of new content is open to all users, duplicate content is a problem that leads to a poor distribution of resources[15]. For example, searching for "Ubuntu 14.04 AMD64 torrent" on Google at the time of writing yields several results. Three contain exactly the same content, the Ubuntu ISO file named `ubuntu-14.04-desktop-amd64.iso`, 1 010 827 264 bytes in size. However, they have different hashes, because they use different piece sizes or needlessly use the BitTorrent private flag, as summarized in Table I.

We design a duplicate content detection subsystem for the CMS that groups files based on readily available metadata, specifically, file size and file name. Our design deliberately does not include inspecting the actual data, therefore providing fast results. The subsystem uses the file size information recorded in the torrents as the first criterion for duplicate detection. If two files have the same size, the subsystem compares their names. The subsystem uses the Levenshtein distance [16]—a type of edit distance—to decide if two equally sized files are duplicates or not. Whenever the file names are less than five edits apart, they are considered duplicates.

For every group of duplicates detected, the CMS only uses the swarm with the largest number of seeders for credit mining. This peer-level behavior aligns with the community goal of consolidating swarms, which has been proven to increase performance [17]. However, this may seem counter intuitive at first sight from a credit mining perspective. Were the peer to choose the swarm with less seeders, the potential for upload would be greater. However, in the long term, the swarm with less seeders will disappear first from the community, so the long term potential for upload is greater for swarms with more seeders.

Each peer runs the duplicate detection algorithm whenever a new torrent is added to the CMS. In case the new torrent is a duplicate of an existing torrent, the selection for credit mining may stop the existing torrent if it has less seeders. Furthermore, the CMS reevaluates the selection periodically to ensure the torrent with most seeders is used for credit mining.

B. Spam detection

Spam, defined as misrepresenting content in order to circumvent established retrieval and ranking techniques, is a pervasive phenomenon in P2P systems [18].

For the CMS implementation, we use an explicit spam detection mechanism based on the votes of users in the Tribler community. Dispersy, the Tribler subsystem that implements channels, enables users to do collaborative spam filtering at the channel level. A specific Dispersy message is created for every vote cast by a user on a channel, classifying the channel as spam or not spam. Each peer adds the Dispersy message representing the user vote to its Bloom filter. Dispersy then distributes the messages such that a global consensus can be formed. Each peer is thus aware of the spam status of channels as resulting from collaborative filtering. The CMS uses this information for detecting spam among the swarms in its sources. If a swarm is present in more spam than not spam channels, the CMS considers it spam and does not take it into consideration for credit mining.

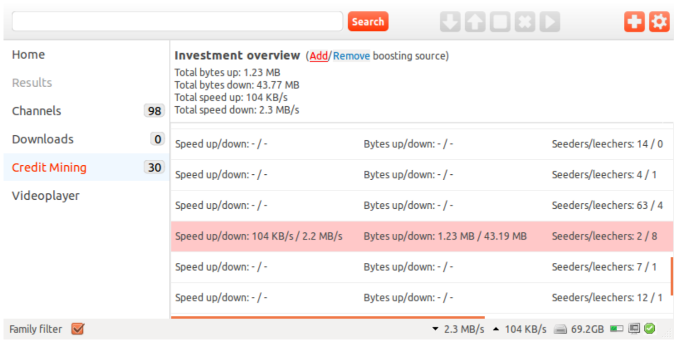
C. Updating information

It is important that the information used by the CMS swarm selection algorithm as input is constantly updated to reflect changes in the P2P community. The network traffic generated while updating this information may constitute a non-negligible overhead compared to the upload generated by the CMS.

Each peer periodically retrieves updated information from trackers through a standard BitTorrent protocol called *scraping*. We design an efficient tracker scraping subsystem that minimizes network traffic. As a result, the tracker load is also minimized, which again represents an alignment of peer- and community-level goals. The main idea is for each peer to group together torrents by tracker and make a single scrape request for all torrents in a group. Naturally, there has to be a limit on the number of torrents part of a single request, i.e., the maximum size of the message to be sent over the network. In accordance with the BitTorrent UDP Tracker Protocol, we limit the number of torrents in a single scrape request to 74 [19]. Each peer scrapes trackers at an interval of equal to the swarm selection interval, which we experimentally explore in Section VI.

V. Tribler implementation

From a software engineering perspective, Tribler consists of modules divided in two layers: the core and the graphical user interface, to facilitate the development of alternative user interfaces. Our CMS implementation covers



Investment overview (Add/Remove boosting source)
Total bytes up: 1.23 MB
Total bytes down: 43.77 MB
Total speed up: 104 KB/s
Total speed down: 2.3 MB/s

```
Speed up/down: - / -                 Bytes up/down: - / -                 Seeders/leechers: 14 / 0
Speed up/down: - / -                 Bytes up/down: - / -                 Seeders/leechers: 4 / 1
Speed up/down: - / -                 Bytes up/down: - / -                 Seeders/leechers: 63 / 4
Speed up/down: 104 KB/s / 2.2 MB/s   Bytes up/down: 1.23 MB / 43.19 MB    Seeders/leechers: 2 / 8
Speed up/down: - / -                 Bytes up/down: - / -                 Seeders/leechers: 7 / 1
Speed up/down: - / -                 Bytes up/down: - / -                 Seeders/leechers: 12 / 1
```

Fig. 2. The Tribler credit mining user interface, showing an overview, the list of potential swarms, and one active swarm.

both layers. Like the rest of Tribler, our implementation uses the Python programming language and consists of approximately 2000 lines of changes on top of Tribler, including the unit tests.

We allow for easy extension of the CMS through new pluggable policies. For swarms selection, new policies must implement a comparison function for swarms, which has access to all swarm parameters in Tribler, including number of seeders and leechers, and creation date, which we used for the policies tested in this paper.

The CMS can be configured through a configuration file which specifies parameters, such as the swarm selection policy or the sharing ratio target. However, we have also implemented a full graphical user interface for the CMS, which is depicted in Figure 2.

First of all, the user interface allows users to manage the credit mining sources, i.e., add and remove sources. Second, it shows an overview of the credit mining activity, including the total amount of bytes uploaded and downloaded, and the upload and download speed of the CMS. Finally, users can check the list of swarms that are part of the CMS and understand exactly what the CMS is doing. We believe this transparency will encourage people to actively use credit mining in Tribler.

VI. Evaluation of credit mining in Tribler

In this section, we evaluate the CMS as implemented in Tribler through experiments we conduct over the Internet. We use Tribler instances that we control to connect to real-world swarms we do not control, and record the activity of the CMS. First, we evaluate the CMS configuration parameters and propose a default value for each parameter. We then evaluate the effect of a community-wide CMS deployment of by inserting many CMS peers at the same time in a small number of swarms.

A. Experimental setup

As source of swarms, we use the RSS web feed of `etree.org`, a community that shares music with permission from authors. (In fact, BitTorrent was originally designed to serve this community [20].) The RSS web

TABLE II
PARAMETERS USED IN THE EXPERIMENTS

Parameter	Values
Swarm selection policy	SeederRatio, SwarmAge, Random
Swarm selection interval	5 min, 10 min, 30 min
Sharing ratio target	1, 3, 5

feed contains the most recent 30 torrents published in the community. New torrents are added at a rate of approximately 20 per day, according to our observation. We check the RSS web feed for new torrents every 30 minutes, the default Tribler interval. At any moment, we credit mine a maximum of 13 swarms simultaneously, the Tribler default for the maximum number of active swarms.

In Table II we summarize the CMS parameters we change during the experiments. These parameters cover both the swarm selection stage—policy and interval—and the in-swarm behavior stage—sharing ratio target. We evaluate each of the three parameters in turn. For each of the parameters, we run three instances of Tribler, one for each value of the parameter, while the other parameters have a fixed value. The full range of potential values for each parameter is impossible to explore exhaustively using real-world experiments. We limit this experiment to three values per parameter, which we select based on our knowledge gained from designing and implementing the CMS.

Throughout the experiments, we use as a performance evaluation metric for the CMS the net credit gain in Tribler, which is equal to the net upload gain in bytes:

$$\text{net upload gain} = \text{uploaded bytes} - \text{downloaded bytes}$$

When discussing the efficiency of the CMS, we also refer to the normalized upload gain, which normalizes the net upload gain by the amount of bytes downloaded:

$$\text{normalized upload gain} = \frac{\text{net upload gain}}{\text{downloaded bytes}}$$

B. *Choosing a swarm selection policy*

In our first experiment, we evaluate the swarm selection policy. Figure 3 depicts the evolution over a two-day period of the credit earned by the CMS expressed as net upload gain. The colored dots represent individual swarms while the black line represents the overall result. The swarm selection policy differs between the three subgraphs, while the other two parameters, swarm selection interval and sharing ratio target, are fixed to 5 minutes and 3, respectively. Note that we use a logarithmic vertical axis in Figure 3 (as well as Figures 4 and 5), to better illustrate the evolution of each swarm. At the beginning of credit mining in a swarm, the net upload gain is always negative, because it is necessary to download at least one piece before uploading. Furthermore, because of transfer

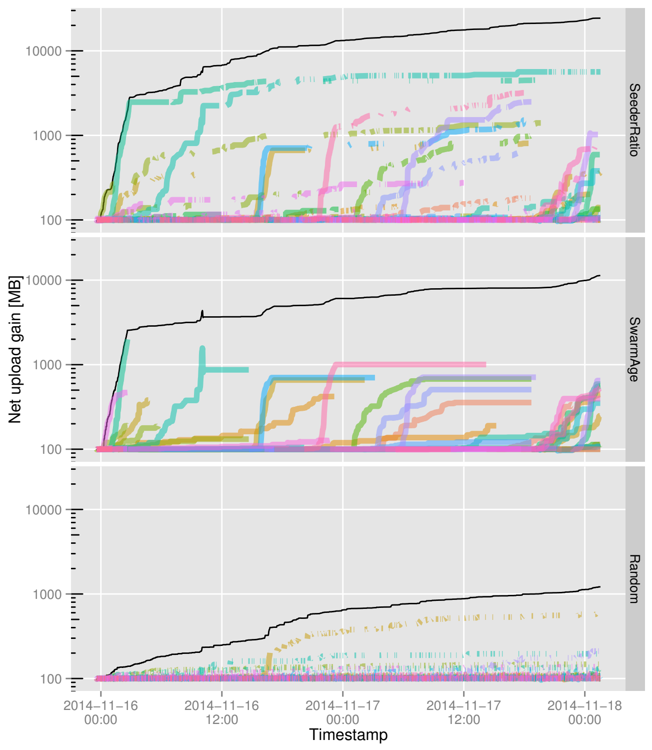


Fig. 3. The net upload gain for the three swarm selection policies. Colored lines represent individual swarms, the black line represents the total. Logarithmic vertical axes with data translated by 100 MB.

errors, some pieces require more than one download attempt, also resulting in a negative upload gain. In order to take the logarithm given these negative values, we apply a translation to the data, adding 100 MB to each swarm and to the total.

Analyzing Figure 3, we conclude that the SeederRatio policy is the most efficient for credit mining. The credit earned by the CMS during the experiment with the SeederRatio policy, 24.27 GB, is more than twice as high as the 11.25 GB earned with the second best policy, SwarmAge, and more than 20 times as high compared to 1.12 GB of the worst policy, Random. Following the evolution of individual swarms (identified by color), we notice a similar relative evolution of swarms in each of the subgraphs; even for the Radom policy, we note that the successful swarms are also successful when using the other two policies. The individual swarm analysis also reveals why the SwarmAge policy performs worse than SeederRatio; while the beginning of credit mining is almost identical for both policies, SwarmAge stops mining a swarm when newer swarms appear, while the SeederRatio policy returns to swarms in need of upload throughout the experiment.

C. *Choosing a swarm selection interval*

In Figure 4, we plot the results of evaluating the second parameter, the swarm selection interval. It is important to note that we set tracker scraping to use the same interval.

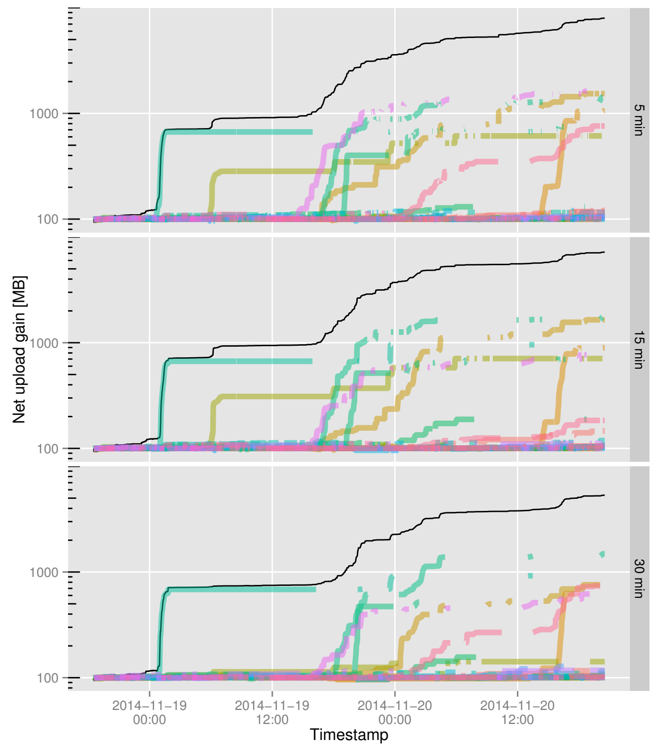


Fig. 4. The net upload gain for the three swarm selection intervals. Colored lines represent individual swarms, the black line represents the total. Logarithmic vertical axes with data translated by 100 MB.

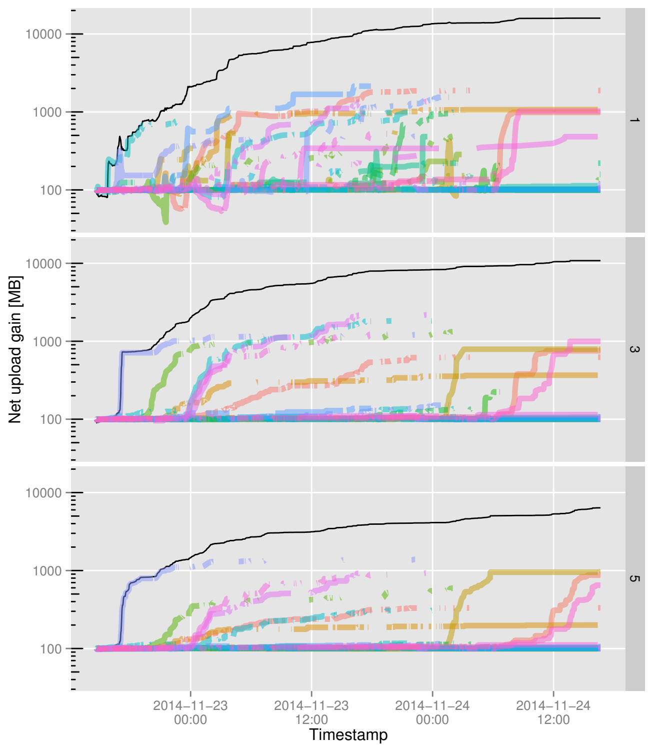


Fig. 5. The net upload gain for the three sharing ratio targets. Colored lines represent individual swarms, the black line represents the total. Logarithmic vertical axes with data translated by 100 MB.

Intuitively, the smaller the interval, the more up-to-date the CMS information will be, and the more credit should be gained. The other two CMS parameters are set to SeederRatio and 3.

Indeed, we observe an inverse correlation between the interval duration and the amount of credit gained, with values 5 min, 15 min, and 30 min producing 7.87 GB, 7.10 GB, and 5.23 GB, respectively. Note that the numbers for credit gained should only be compared among themselves and not with the numbers in Figure 3, because the set of swarms used differs. Individual swarm behavior (identified by color) is almost identical across all subgraphs, the only difference being the amount of credit gained for each swarm. Recall that the three Tribler instances in the experiment use the same set of swarms for credit mining. Therefore, by the time swarms appear in the Tribler instance using the 30 min interval, the potential for upload is already largely fulfilled by the other two Tribler instances with shorter intervals. However, even the 30 min interval generates substantial credit, so it may be desirable to use it as a default value, considering the reduced overhead it brings to both the peer and the community tracker.

D. *Choosing a sharing ratio target*

Finally, in Figure 5 we depict the results of the third experiment, evaluating last CMS parameter, the sharing ratio target; the other two parameters are set to SeederRatio and 5 minutes. Considering the total net credit gain, setting the sharing ratio target to 1 produces the best results, 15.86 GB, followed by 3, 10.71 GB, and 5, which produces 6.28 GB.

However, if we compute the normalized credit gain, we observe that it is proportional to the sharing ratio target; the normalized credit gains are 1.08, 3.81, and 5.92 for sharing ratio targets 1, 3, and 5, respectively. Using a sharing ratio target of 1 is inefficient (there are only 1.08 GB of upload gained for each GB downloaded), but this lack of efficiency is compensated by an increase in the activity of the CMS: data is transferred in many swarms which accumulates to an overall high net upload gain. Thus, if we consider the resources used for credit mining (e.g., network traffic, storage space), it is desirable to select a higher sharing ratio target instead of 1. Overall, we conclude that the Libtorrent default of 3 is a good value for the sharing ratio target, resulting in a good balance between the amount of credit gained and resources used.

E. *Effect of widespread credit mining*

In this fourth experiment, we emulate the widespread deployment of the CMS by running 20 instances of Tribler simultaneously using the same source of swarms, the `etree.org` RSS web feed. From our observation, most of the swarms have fewer than 5 downloaders, not including

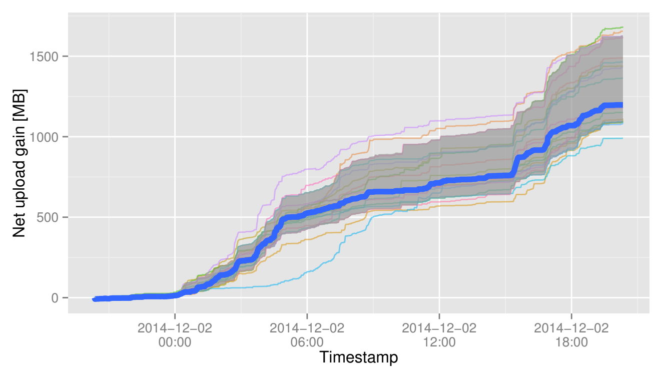


Fig. 6. The net upload gain of 20 credit mining peers deployed simultaneously, represented by colored lines. The thick line represent the median and the shaded area represents the 10th to 90th percentile region.

our Tribler peers. Thus we produce during this experiment a high proportion of credit mining peers.

Figure 6 illustrates the effect of simultaneously running the 20 credit mining peers, all using the SeederRatio swarm selection policy, a swarm selection interval of 5 minutes, and a sharing ratio target of 3. For clarity, we only plot the overall net upload gain for each peer, without the evolution of individual swarms. Each colored line represent one peer, the thicker line represents the median of all peers, and the shaded area represents the gap between the 10th and 90th percentiles.

Even taking into account the variability of the swarms used as input, we notice a significantly lower net upload gain for the individual credit miners compared to the previous experiments, with a median of 1.20 GB after one day of credit mining. However, there are no catastrofic effects of widespread CMS deployment. In fact, all peers still obtain a positive net upload gain after one day of activity, with a minimum of 1.00 GB. The distance between the 10th percentile and the 90th percentile is only 0.54 GB, showing that credit mining peers have similar chances of contributing to the community.

VII. Conclusion

The evolution of P2P systems has been tightly coupled to the evolution of incentive mechanisms. Accounting mechanisms based on credit or on sharing ratio are currently a widespread method for incentivizing seeding in BitTorrent communities. In this paper, we have introduced CMS, a decentralized system for credit mining that enables honest users to contribute their idle upload bandwidth to the community. Using the CMS is beneficial both for the user, who earns credit, and for the community, where the contributed upload speed results in faster downloads for other users.

We have explored the parameters that control the CMS behavior and provided insight into their effect on credit mining efficiency and resource usage. Ultimately, the balance between credit earned and resources used is best left to the user and we plan to implement in Tribler advanced settings for changing this balance.

Furthermore, we have shown that CMS deployment is reasonable even considering widespread deployment in the community. The behavior of CMS peers within a swarm prevents them from needlessly wasting resources on uploading and downloading from each other. Instead, the CMS peers take turns at providing upload bandwidth to the actual downloaders in the community, thus always obtaining a positive net upload gain.

Acknowledgments

The authors would like to thank Egbert Bouman for help with the Tribler code and Elric Milon for help with setting up the experiments.

References

[1] B. Cohen, "Incentives Build Robustness in BitTorrent," in *Workshop on Economics of Peer-to-Peer Systems*, Berkley, CA, USA, 2003. [Online]. Available: http://sims.berkeley.edu/research/conferences/p2pecon/papers/s4-cohen.pdf

[2] E. Adar and B. A. Huberman, "Free riding on Gnutella," *First Monday*, vol. 5, no. 10, Oct. 2000. [Online]. Available: https://doi.org/10.5210/fm.v5i10.792

[3] M. Meulpolder, L. D'Acunto, M. Capotă, M. Wojciechowski, J. A. Pouwelse, D. H. J. Epema, and H. J. Sips, "Public and private BitTorrent communities: A measurement study," in *International Workshop on Peer-to-Peer Systems IPTPS*. San Jose, CA: USENIX Association, 2010. [Online]. Available: https://www.usenix.org/conference/iptps-10/public-and-private-bittorrent-communities-measurement-study

[4] A. L. Jia, R. Rahman, T. Vinkó, J. A. Pouwelse, and D. H. Epema, "Fast download but eternal seeding: The reward and punishment of Sharing Ratio Enforcement," in *IEEE International Conference on Peer-to-Peer Computing (P2P)*. IEEE, Aug. 2011, pp. 280–289. [Online]. Available: https://doi.org/10.1109/P2P.2011.6038746

[5] N. Zeilemaker, M. Capotă, A. Bakker, and J. A. Pouwelse, "Tribler: P2P media search and sharing," in *ACM International conference on Multimedia (MM)*. New York: ACM Press, 2011, pp. 739–742. [Online]. Available: https://doi.org/10.1145/2072298.2072433

[6] M. Meulpolder, J. A. Pouwelse, D. H. J. Epema, and H. J. Sips, "BarterCast: A practical approach to prevent lazy freeriding in P2P networks," in *IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE, May 2009, Conference proceedings (article), pp. 1–8. [Online]. Available: https://doi.org/10.1109/IPDPS.2009.5160954

[7] B. Zhang, A. Iosup, J. A. Pouwelse, and D. H. J. Epema, "Identifying, analyzing, and modeling flashcrowds in BitTorrent," in *IEEE International Conference on Peer-to-Peer Computing*. IEEE, Aug. 2011, pp. 240–249. [Online]. Available: https://doi.org/10.1109/P2P.2011.6038742

[8] I. A. Kash, J. K. Lai, H. Zhang, and A. Zohar, "Economics of BitTorrent communities," in *International conference on World Wide Web (WWW)*. New York: ACM Press, 2012, p. 221. [Online]. Available: https://doi.org/10.1145/2187836.2187867

[9] M. Piatek, T. Isdal, A. Krishnamurthy, and T. Anderson, "One hop reputations for peer to peer file sharing workloads," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. San Francisco, California: USENIX Association, 2008, pp. 1–14. [Online]. Available: https://www.usenix.org/conference/nsdi-08/one-hop-reputations-peer-peer-file-sharing-workloads

[10] R. Petrocco, J. A. Pouwelse, and D. H. J. Epema, "Performance analysis of the Libswift P2P streaming protocol," in *IEEE International Conference on Peer-to-Peer Computing (P2P).* IEEE, Sep. 2012, pp. 103–114. [Online]. Available: https://doi.org/10.1109/P2P.2012.6335790

[11] N. Zeilemaker, B. Schoon, and J. A. Pouwelse, "Large-scale message synchronization in challenged networks," in *ACM Symposium on Applied Computing (SAC).* New York: ACM Press, 2014, pp. 481–488. [Online]. Available: https://doi.org/10.1145/2554850.2554908

[12] M. Capotă, N. Andrade, J. A. Pouwelse, and D. H. J. Epema, "Investment Strategies for Credit-Based P2P Communities," Tech. Rep. PDS-2014-005, 2014. [Online]. Available: http://www.pds.ewi.tudelft.nl/fileadmin/pds/reports/2014/PDS-2014-005.pdf

[13] M. Capotă, J. A. Pouwelse, and D. H. J. Epema, "Towards a Peer-to-Peer Bandwidth Marketplace," in *Distributed Computing and Networking (ICDCN)*, M. Chatterjee, J.-n. Cao, K. Kothapalli, and S. Rajsbaum, Eds. Coimbatore, Tamil Nadu, India: Springer, 2014, pp. 302–316. [Online]. Available: https://doi.org/10.1007/978-3-642-45249-9_20

[14] A. Norberg, "Libtorrent." [Online]. Available: {http://libtorrent.org}

[15] O. Papapetrou, S. Ramesh, S. Siersdorfer, and W. Nejdl, "Optimizing Near Duplicate Detection for P2P Networks," in *IEEE International Conference on Peer-to-Peer Computing (P2P).* IEEE, Aug. 2010, pp. 1–10. [Online]. Available: https://doi.org/10.1109/P2P.2010.5570001

[16] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," in *Soviet physics doklady*, vol. 10, 1966, p. 707.

[17] G. Dán and N. Carlsson, "Centralized and Distributed Protocols for Tracker-Based Dynamic Swarm Management," *IEEE/ACM Transactions on Networking*, vol. 21, no. 1, pp. 297–310, Feb. 2013. [Online]. Available: https://doi.org/10.1109/TNET.2012.2198491

[18] D. Jia, W. G. Yee, and O. Frieder, "Spam characterization and detection in peer-to-peer file-sharing systems," in *ACM conference on Information and knowledge mining (CIKM).* New York: ACM Press, 2008, p. 329. [Online]. Available: https://doi.org/10.1145/1458082.1458128

[19] O. van der Spek, "UDP Tracker Protocol for BitTorrent," BitTorrent, BitTorrent extension proposal, 2008. [Online]. Available: http://bittorrent.org/beps/bep_15.html

[20] K. Peterson, "BitTorrent file-sharing program floods the Web," Seattle Times, 2005. [Online]. Available: {http://seattletimes.com/html/businesstechnology/2002146729_bittorrent10.html}