Delft University of Technology
Parallel and Distributed Systems Report Series

Investment strategies for credit-based P2P communities

Mihai Capotă, Nazareno Andrade,
Johan Pouwelse, and Dick Epema
mihai@mihaic.ro

Completed December 2014.

Report number PDS-2014-005

ISSN 1387-2109

Published and produced by:
Parallel and Distributed Systems Group
Department of Software and Computer Technology
Faculty Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

Information about Parallel and Distributed Systems Report Series:
reports@pds.ewi.tudelft.nl

Information about Parallel and Distributed Systems Section:
http://www.pds.ewi.tudelft.nl/

Abstract

P2P communities that use credits to incentivize their members to contribute have emerged over the last few years. In particular, private BitTorrent communities keep track of the total upload and download of each member and impose a minimum threshold for their upload/download ratio, which is known as their sharing ratio. It has been shown that these private communities have significantly better download performance than public communities. However, this performance is based on oversupply, and it has also been shown that it is hard for users to maintain a good sharing ratio to avoid being expelled from the community. In this paper, we address this problem by introducing a speculative download mechanism to automatically manage user contribution in BitTorrent private communities. This mechanism, when integrated in a BitTorrent client, identifies the swarms that have the biggest upload potential, and automatically downloads and seeds them. In other words, it tries to invest the bandwidth of the user in a profitable way. In order to accurately asses the upload potential of swarms we analyze a private BitTorrent community and derive through multiple regression a predictor for the upload potential based on simple parameters accessible to each peer. The speculative download mechanism uses the predictor to build a cache of profitable swarms to which the peer can contribute. Our results show that 75 % of investment decisions result in an increase in upload bandwidth utilization, with a median 207 % return on investment.[1]

[1] This paper is an extension of our work published at Euromicro PDP 2013 [1].

Contents

List of Figures

List of Tables

1 Introduction

User-contributed resources are the basis of high performance in P2P systems. Because of that, an adequate incentive structure that encourages users to contribute is paramount to the success of a P2P system. BitTorrent presently stands at a crossroads in this respect. On the one hand, BitTorrent has become arguably the most successful P2P system at least in part because of its built-in contribution incentive mechanism, *tit for tat* [2, 3], which rewards users that are downloading, called *leechers*, with better performance if they contribute high upload bandwidth to help other leechers. On the other hand, *seeding* – uploading after the download is finished – is not incentivized in the BitTorrent protocol, and there is insufficient attention given to the simultaneous distribution of many files.

The result of these drawbacks is the creation of hundreds of private BitTorrent communities that employ extra incentive mechanisms to encourage their users to contribute resources. Private BitTorrent communities address the lack of incentives for seeding in BitTorrent by tracking the sharing behavior of users across swarms [4, 5]. Typically, users who do not keep a minimum ratio of uploaded and downloaded data—the *sharing ratio*—are penalized and/or expelled from the community. This leads to an oversupply of upload bandwidth which gives users of private communities very good download performance [4].

Unfortunately, the upload bandwidth oversupply in private communities has a negative side effect. It makes it difficult for users that are willing to contribute to do so. A user may have to compete with too many other users to upload in a swarm, and may be unable to attain a sufficient sharing ratio to maintain membership in the community [6]. If, on top of that, a user downloads some content that is not popular and therefore won't be downloaded by many others in the future (which is a common phenomenon [7]), it becomes very difficult to upload that file to rebalance the sharing ratio.

We argue that while sharing ratio enforcement currently encourages contribution, the skewed distribution and temporal variability of content popularity often make it exceedingly hard for users to contribute effectively in BitTorrent private communities. In short, for a user, there is a fundamental mismatch between the acts of downloading and uploading when it comes to maintaining a good sharing ratio while pursuing their own interest in files.

In this paper, we propose, as a solution to this problem, the decoupling of downloading and seeding, and put forward an automatic mechanism for users who are willing to contribute resources to the community to do so easily. This mechanism speculatively downloads and seeds swarms with the sole purpose of increasing the upload/download ratio. As long as the choice is made by an informed algorithm, the user stands to gain, and so does the community, since the download performance of the other users increases.

The solution we introduce can be described in terms of a cache that contains the swarms that have been downloaded speculatively. Its most important component is the swarm replacement algorithm: How to select which new swarm should go into the cache and what old swarm should be removed? To answer this question, we build a regression model with as input the swarm characteristics available to each peer, and obtain a prediction of the amount of bytes the peer will be able to upload to that swarm in the near future. Note that, as opposed to a traditional cache, we do not store in our system the swarms downloaded by the user of the BitTorrent client. Instead, we consider any swarm available in the community as a candidate for download.

The rest of the paper is structured as follows. In Section 2 we present related work. We describe the problem being addressed in Section 3. Section 4 presents in detail the design of our solution. Section 5 presents the evaluation setup we use and Section 6 describes the results we obtain. We conclude with Section 7.

2 Related work

A considerable body of research focuses on incentive mechanisms in BitTorrent, especially considering individual download sessions [2, 8, 9]. However, users can be observed downloading many files, often in overlapping download sessions. Since the pure BitTorrent protocol does not provide any way to identify peers across download sessions, many other works propose reputation systems that are supposed to keep track of user behavior. Research has focused mainly on decentralized methods to build reputation systems [10, 11]. In practice however, centralized solutions are widespread, in the form of private BitTorrent communities that use a centralized server to keep track of user identity and activity [4]. Although we use data from a private community as opposed to a decentralized reputation system, we note that our work is orthogonal to the practical implementation of such a system; the only requirement for our solution to work is having a system that keeps track of user contribution across swarms – a *credit* system.

The oversupply problem of private BitTorrent communities is well documented in the literature, along with the problems it raises for maintaining membership [4, 6, 5].

There are many examples of caching in P2P systems [12, 13, 14, 15]. While our system resembles a cache, remember that it plays an active role in selecting the swarms that it can store and does not rely on the user for this. In fact, one of the goals of our work is to decouple a user's contribution to the community from the user's preference for content, since it is not likely the two will match. On this topic, Wu et al. [7] propose a solution based on a centralized server that assigns users to semi-static groups that contribute upload bandwidth to specific channels in a live video streaming system. Instead, we solve the problem for file-sharing systems, but in a completely decentralized manner. Carlsson et al. [16] propose another solution, again based on central servers, that "inflate" swarms with peers that are not currently fully utilizing their upload bandwidth. Garbacki et al. [17] propose a mechanism for bandwidth exchange between peers that decouples peer contribution from peer interest, but is based on explicit help requests peers send to one another to request the download of specific swarms.

Regression models have been used before to predict characteristics of BitTorrent, like online time [18], or peer download speed [19]. However, to the best of our knowledge, we are the first to predict the upload speeds that peers can achieve in BitTorrent swarms.

3 Problem statement

Because of oversupply, in many private BitTorrent communities it is hard for users who are willing to contribute upload bandwidth to actually do so. This unnecessarily limits the sharing ratio of these users, as well as the global download performance in the community. The goal of this work is to make use of any idle upload bandwidth users have in an automatic and decentralized manner that is compatible with regular BitTorrent clients, such that the sharing ratio of users is increased. We name problem of deciding whether and what to automatically download in order to increase upload bandwidth usage the *Bandwidth Investment Problem*.

To solve the bandwidth investment problem, a BitTorrent client must identify swarms – also called *torrents* – that are undersupplied in the community, download some portion of their content and seed those torrents using the idle upload bandwidth. In some sense, this is what experienced users often do manually, in addition to downloading torrents they are interested in. We aim to automate this process.

As a success criteria, if a solution is applied to this problem and used for a reasonable amount of time – on the order of hours –, it should cause an increase in the sharing ratio of its user. To compare the quality of several solutions, we can run them at the same time and compare the

sharing ratios they produce. From the perspective of bandwidth investment, we are comparing the profits produced by the different strategies.

There are two constraints we also consider solutions must satisfy. Any mechanism that qualifies as a solution must be integrated into a regular BitTorrent client which is under user control. As such, it always has to give priority to user actions. If the user decides to download a torrent, the mechanism must yield bandwidth so that the user download can proceed as fast as possible. Similarly, it must limit storage usage so that the user always has available storage.

In addition to the previous constraint, the information used to decide which swarms are undersupplied must already be available to regular BitTorrent peers. We do not believe a mechanism that requires changes to the established BitTorrent protocol has chances to be adopted.

4 Design

In this section, we introduce and describe in detail the speculative downloading mechanism, the solution we propose for the Bandwidth Investment Problem.

4.1 Speculative download

The main contribution of this paper is the speculative download mechanism depicted in Figure 1 that identifies, downloads, and seeds undersupplied swarms in private communities. The new module to be added to BitTorrent clients is called the *Speculative Download Engine* (SDE). All user commands pass through it, so that it can react appropriately, e.g., yield bandwidth when the user is downloading. At the same time, it gets input from a *Discovery* module, which provides a list of available swarms, together with their characteristics. This module already exists in many BitTorrent clients as an RSS feed parser, and many communities offer customizable RSS feeds for announcing newly available torrents. The commands issued by the SDE are executed by the *Data transfer* module which implements the BitTorrent protocol to transfer data between other peers and the *Storage* module. After a warm-up period, during which the engine adds data to the storage, the mechanism will start operating in a full-cache mode, where every new swarm has to replace an old one. For simplicity, and since the user is not interested in the actual data, we establish a standard size for downloads instead of downloading all data from a swarm, and use partial seeding [20]. This way, the storage is divided in pieces of equal size and the replacement policy does not have to consider the space swarm data occupies.

Considering the speculative download mechanism is similar to a cache, we can also analyze its functionality in terms of cache replacement. Given a set of available swarms in the community and a set of swarms already in the cache, find the best swarm not in the cache and the worst swarm in the cache and decide whether to replace the swarm in the cache with the one available. The decision is based on the predicted upload speed in the swarms. In the next subsection, we show how this decision can be made.

4.2 Predicting future upload speed

For any BitTorrent peer and swarm, given the set P of characteristics of the swarm that are available to all peers (number of seeders, number of leechers, file size, swarm age), the set I of characteristics of the peer (download and upload bandwidth), and, optionally, the set H of records of the peer's historical activity in the swarm (download and upload history), we must give an estimate of the expected upload speed the peer will achieve in the swarm.We can asses the accuracy of an upload speed predictor by comparing the prediction to the actual upload

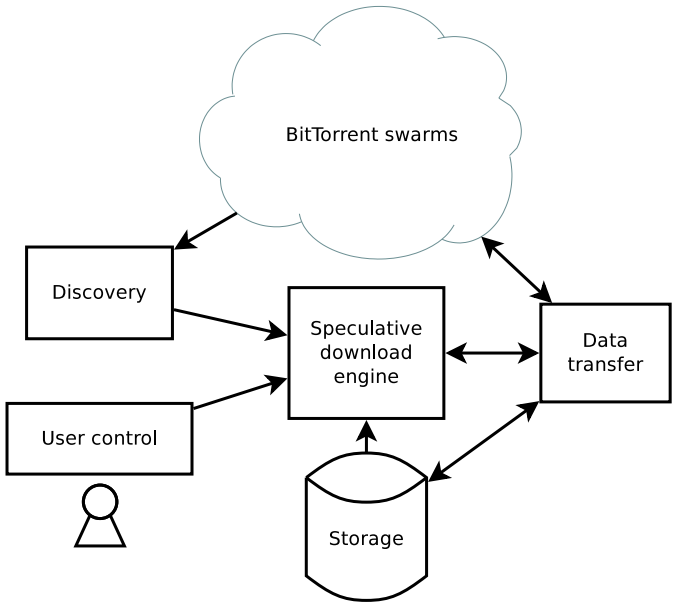

Figure 1: Speculative download mechanism.

speed a peer achieves when participating in the swarm. We then use this predictor to drive the SDE.

In this paper, we predict upload speed using regression analysis. Given a trace of the behaviour of a peer with s samples, each describing the activity of the peer in a swarm during a certain time interval, we construct a multiple regression model

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \cdots + \beta_n x_{ni}, \quad i = 1, \ldots, s$$

where the upload speed of the peer in the swarm during the interval when sample i was captured, y_i, is a dependent variable determined by the independent variables $\{x_{1i}, \ldots, x_{ni}\} = P \cup I \cup H$. For a trace with s samples, we have s equations. Usually we have more samples than independent variables, so $s > n$, which means we have an overdetermined system of equations that is inconsistent and has no exact solution. We choose parameters $\beta_1, \ldots, \beta_n$ and the intercept β_0 so that they fit the data best using the least squares method. We then use the parameters β to predict the upload speed of the peer in any new swarm at any time, given characteristics $x_{1new}, \ldots, x_{nnew}$.

Because our data has unknown distributions for characteristics, we use a technique called *multivariate adaptive regression splines (MARS)* [21], which is an extension of the linear regression we presented. MARS builds a model for the dependant variable using an iterative process. Starting from a model using only the intercept, MARS creates intermediary models by gradually adding independent variables such that the least squares error of the model is minimized.

4.3 Putting it all together

We can now present in detail the operation of the cache replacement mechanism of the SDE using upload prediction. This is depicted in Algorithm 1.

Algorithm 1 Cache replacement algorithm

1: $N = \{\text{Swarms not in cache}\}$
2: $C = \{\text{Swarms in cache}\}$
3: **for all** $s \in N$ **do**
4: $\quad prediction[s] = predict\_upload\_speed\_new(s)$
5: **end for**
6: **for all** $s \in C$ **do**
7: $\quad prediction[s] = predict\_upload\_speed\_old(s)$
8: **end for**
9: $h = highest(N, prediction)$
10: $l = lowest(C, prediction)$
11: **if** $prediction[h] > prediction[l]$ **then**
12: $\quad remove\_from\_cache(l)$
13: $\quad add\_to\_cache(h)$
14: **end if**

The algorithm predicts the upload speed of all the swarms that have not been downloaded yet (N), and, separately, it predicts the upload speed of all the swarms that have been downloaded already (C). Note that we can use different regression models for the two sets of swarms, since we have more information for C (set H is available, i.e., history of download and upload). Next, the algorithm sorts the two sets of swarms according to the predicted upload speeds and chooses the swarm with the highest predicted upload speed from the new set, and the swarm with the lowest predicted upload speed from the old set. If the prediction for the former is higher than the prediction for the latter, it does the cache replacement. This algorithm can be run at regular intervals, or whenever the upload speed drops below a preconfigured threshold.

It is important to also consider the cost of downloading the new swarm that is to be put in the cache in order to have a realistic model. Recall that our speculative download engine uses partial seeding and that we are only downloading a fixed amount of data from every swarm we put in the cache. We are free to set this cache block size, denoted b, to maximize the efficiency of our algorithm; 16 MiB is a good value according to our experiments. Thus, for every cache replacement operation, we have to not only achieve a better download speed, but also make sure we recuperate the cost of downloading the new cache block. It is necessary to introduce a new parameter for this, a time window w after which we expect to start profiting from the investment in the new swarm. The cache replacement condition on Line 11 of Algorithm 1 becomes:

$$(prediction[h] - prediction[l]) \times w > b$$

5 Trace based evaluation

In this section, we evaluate the efficiency of the SDE using traces of activity from a BitTorrent community.

5.1 Dataset

Our dataset, characterized in Table 1, consists of records of the activity of 84 025 peers in the Bitsoup private community between April and July 2007. The data was collected by crawling the website of the community once per hour (thus $w = 1\text{h}$). It includes, for every peer in every torrent, the amount of data uploaded and downloaded, and the session duration. The mean

Table 1: Characteristics of the dataset

Community name	Bitsoup
Collection interval	April to July 2007
Number of users observed	84 025
Sampling interval	1 h
Mean number of sessions	76 370

number of sessions observed was 76 370. In addition, we have data about the file size for every torrent from the website. For the swarms that started after our monitoring began, we also have the age of the swarm. This gives us all the variables for the prediction algorithm belonging to sets P and H, regarding swarms and past peer activity, respectively (see Section 4.2). We approximate the variables in set I describing the characteristics of the peer, download and upload bandwidth, by using the maximum values observed in the trace for download and upload speed.

We only use samples in which the peer is participating in a single swarm at a time. This gives us a record of the download and upload performance of the peer at a time when the peer is not involved in any other activity in the community, creating a level playing field where each swarm is likely to receive the full bandwidth of the peer.

We differentiate between two types of BitTorrent sessions recorded in the trace. Sessions can either be considered new, where the peer is observed downloading in the trace, or old, where the peer has finished downloading and is now seeding. For the old set, we can use the past peer activity in the swarm to inform the prediction algorithm and improve prediction accuracy.

5.2 Evaluation methodology

To evaluate the sharing speculative download mechanism using the Bitsoup trace, we use Algorithm 2. This is very similar to Algorithm 1, which describes the normal operation of the SDE cache replacement when deployed in a BitTorrent client. However, in addition to predicting the upload speeds of swarms, it also checks the quality of those predictions against the data in the trace. The indicator for prediction quality is the gain in upload speed resulting from the cache swap:

$$gain = \frac{upload\_speed(added\_swarm) - upload\_speed(removed\_swarm)}{upload\_bandwidth}$$

The gain is a dimensionless quantity and it can take values between 1, when a peer is replacing an idle swarm with a swarm with an upload speed that saturates the peer's upload bandwidth, and -1, when a peer replaces a swarm that was saturating its upload bandwidth with a swarm with no upload potential.

For every peer in the community, we select the best swarm not in the cache (i.e., from set N), and the worse swarm in the cache (i.e., from set C). We do this selection according to the upload speed prediction obtained using our predictor function (Lines 11 and 12), just like Algorithm 1 would do. In addition, we also select the best new swarm and the worse old swarm according to the upload speed recorded in the trace (Lines 13 and 14). The decision to do the cache replacement is similar to the one in normal operation, taking into account the time window w and cache block size b. Additionally, *bandwidth* is added as a factor because the trace records upload speeds relative to the upload bandwidth of the peer. If the replacement is made, the resulting gain is evaluated using the actual data from the trace (Line 16). This represents the

Algorithm 2 Evaluating prediction efficiency

1: $N = \{\text{Swarms not in cache}\}$
2: $C = \{\text{Swarms in cache}\}$
3: **for all** $s \in N$ **do**
4: $prediction[s] = predict\_upload\_speed\_new(s)$
5: $trace[s] = get\_upload\_from\_trace(s)$
6: **end for**
7: **for all** $s \in C$ **do**
8: $prediction[s] = predict\_upload\_speed\_old(s)$
9: $trace[s] = get\_upload\_from\_trace(s)$
10: **end for**
11: $h^p = highest(N, prediction)$
12: $l^p = lowest(C, prediction)$
13: $h^t = highest(N, trace)$
14: $l^t = lowest(C, trace)$
15: **if** $(prediction[h^p] - prediction[l^p]) \times w \times bandwidth > b$ **then**
16: $prediction\_gain = trace[h^p] - trace[l^p]$
17: $optimal\_gain = trace[h^t] - trace[l^t]$
18: **end if**
19: **return** $prediction\_gain, optimal\_gain$

extra upload speed the peer gains by replacing the swarm it predicted was the worst in its cache with the swarm it predicted was the best available and not already in its cache. We also compute the optimal gain that is achievable by the peer if the prediction is perfect and identifies the real best and worst swarms in the trace (Line 17).

Algorithm 2 is run once for every peer in the cache. The sets of new and old swarms are built from the sessions of the peer in the trace. All leeching sessions of the peer are put in the new set N, and all seeding sessions of the peer are put in the old set C. It is important to note that time does not play a role in the evaluation, as we are not considering the effects of running the algorithm on the trace, but only evaluate the quality of each peer decision individually. The cache replacement algorithm will try to replace a swarm from C with one from N, according to their predicted upload speeds. The fact that the sessions have taken place at different times in the trace is irrelevant because this is not one of the parameters that influence the upload speed. Notice also that the trace is only used for evaluating the prediction. It is not in any way "replayed" by our algorithm, and the algorithm is not running a simulation of the activity in the community. We are only interested in evaluating the upload speed prediction of single peers using real-world data.

The practical performance of a prediction algorithm depends on the size of the input data. The more swarms used as input, the lower the chances the prediction algorithm will find the cache swap producing the maximum upload speed gain. Because of the importance of input data size, we create from the dataset using random sampling three constrained input data scenarios, in addition to the complete set containing all sessions. Table 2 show the size of swarm sets for all input data scenarios, small (S), medium (M), large (L), and unconstrained (U). We use more new swarms than old swarms in our evaluation because this is the case in real world usage. Because we use random sampling when creating S, M, and L, we repeat the prediction 5 times for every peer and take the mean.

One threat in using regression analysis is over-fitting. This means obtaining a model that gives highly accurate results for the data that was used to create it, but that has poor prediction

Table 2: Size of swarm sets per peer for all input data scenarios

	S	M	L	U
$\|N\|$	10	20	40	|All leeching sessions in trace|
$\|C\|$	5	10	20	|All seeding sessions in trace|

performance. We avoid over-fitting, by splitting our data in two sets, one that we use for creating the regression model and one that we use for testing the prediction which we present in Section 6. The ratio for splitting in our experiments is 80/20.

In addition to SDE, we define three other algorithms to put the results into context. *Random* shows the results of a simulation of repeatedly selecting a random new swarm to replace a random old swarm. On average, this produces a gain equal to the difference between the mean upload speed of the new and old swarms. *SeederRatio* represents a simple heuristic that selects the new swarm with the lowest ratio of seeders to replace the old swarm with the highest ratio of seeders. Finally, we also present an algorithm called *Optimal*, which always replaces the worst old swarm with the best new swarm, thereby producing the best possible gain in upload speed.

6 Results

We start the results presentation by looking at gains in upload speed, i.e., how much faster is the peer uploading after the cache swap relative to its maximum upload speed. Figure 2 shows the CDFs of gains for SDE compared to Random, SeederRatio, and Optimal, for the situations when there is a *potential gain*, i.e., Optimal gain is positive. Random is not making any difference to the upload speed in the median case. Still, when factoring in the cost of downloading a new swarm, it is immediately obvious that Random would result in a decrease of the sharing ratio in the median case (as we will show in Figure 4). SeederRatio produces a small increase in upload speed for the median case, and produces significantly less negative results than Random. We can conclude that SeederRatio is a simple to implement strategy that is effective at increases to upload speed.

For virtually all peers, SDE performs better than SeederRatio. Although there is some variation with input data size, the performance of the different algorithms does not vary significantly. Note that in some cases, SDE decides not to make the cache replacement. These are included in the CDF as having 0 gain.

While SDE does not attain optimal performance, it is important to note that all gains presented in this section are the average case for a single cache replacement. Instead, in normal usage, the algorithm will be run many times by a peer, every time upload speed drops below a certain threshold, so the effects will be cumulative. Furthermore, we expect the upload speed of a peer using our algorithm to be constantly close to maximum, so the necessary gain to reach the maximum will normally be low.

In Figure 3, we present the cases when SDE decides to make a replacement. This time we only show the unconstrained scenario U, since the others are similar. We observe that SDE manages to make a cache replacement that results in an upload bandwidth usage increase in 75 % of the cases. To put this into perspective, note that Random produces no effect for the median case. SeederRatio is once again better than Random, but not as good as SDE.

We analyze the return on investment (ROI) for cache replacement decisions in potential gain situations in Figure 4. For every cache replacement, we compute the ROI considering that for

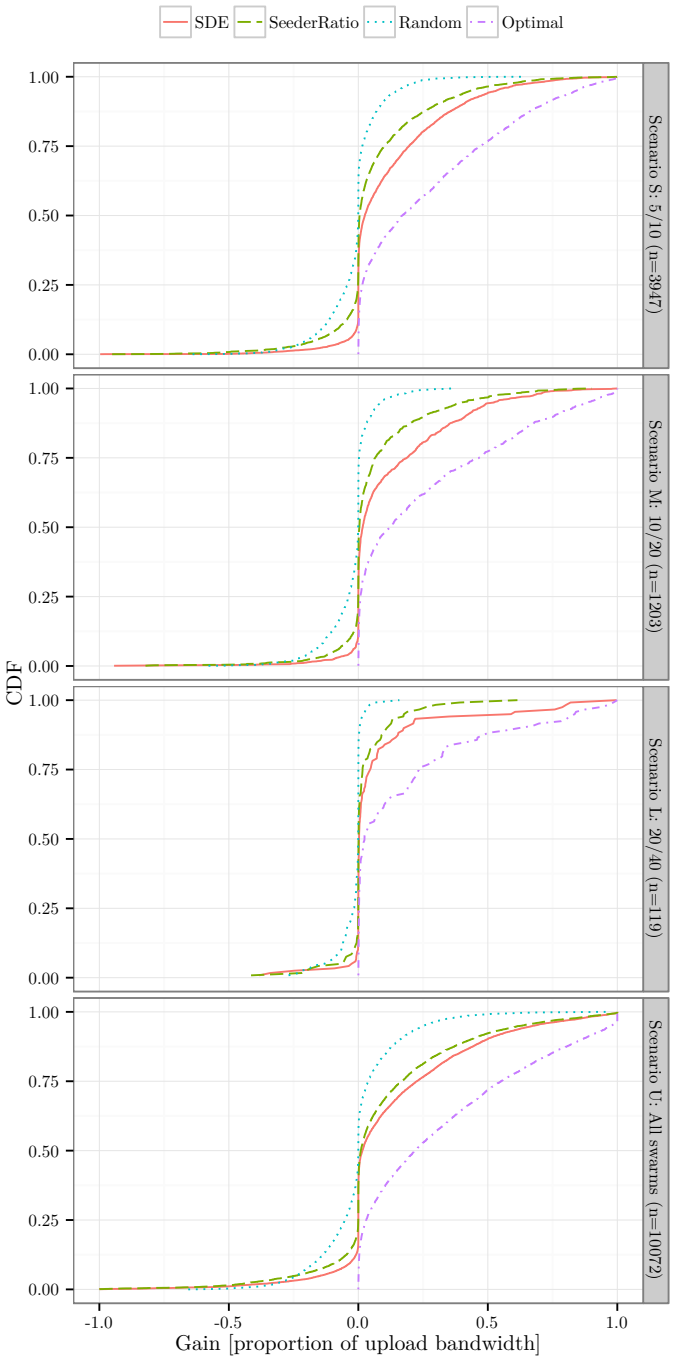

Figure 2: CDF of the gain in upload bandwidth utilization for the SDE, SeederRatio, Random, and Optimal algorithms, in all four scenarios, when Optimal gain > 0. (n denotes the number of users in the experiment.)

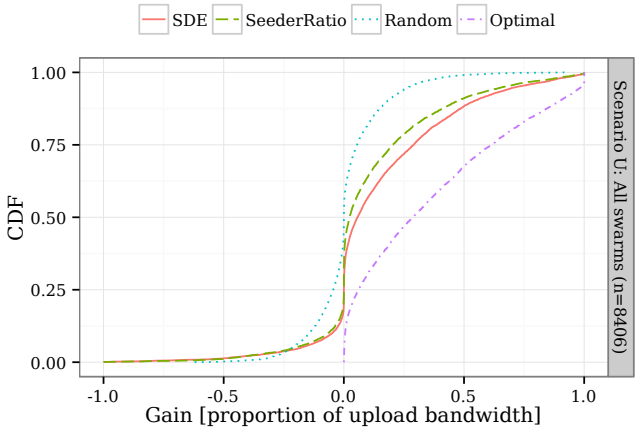

Figure 3: CDF of the gain in upload bandwidth utilization for the SDE, SeederRatio, Random, and Optimal algorithms, in scenario U, when Optimal gain > 0 and SDE performed a cache replacement. (n denotes the number of users in the experiment.)

one hour (the time window w for our dataset) the peer will benefit from the upload speed gained. The investment represents downloading 16 MiB (the cache block b we use) from a new swarm.

$$ROI = \frac{gain \times upload\_bandwidth}{b}$$

We can see that using random choices for swarms results in a negative ROI for the majority of peers in all input data scenarios. Using SeederRatio choices instead gives a positive median ROI for all scenarios except L. On the other hand, using SDE, the median ROI is always positive, ranging from 74 % for the L scenario to 222 % for the S scenario. For U, median ROI is 207 %. We conclude that SDE makes highly profitable investment decisions.

The converse situation for potential gain is when there is no profitable replacement, i.e., no new swarm has a better upload potential than any of the old swarms and Optimal gain < 0. Intuitively, this situation should be rare in private communities. Nevertheless, we examine the performance of SDE in such situations in Table 3. For all scenarios, we see that SDE makes the correct choice of not doing any cache replacement most of the times (for scenario L, the perfect result comes from the small number of samples). When SDE does make a mistake, the cost is usually low: 2–9 % of upload bandwidth is lost in the median case. Note that the statistics for mistakes and loss refer only to the situations where SDE makes a cache replacement.

7 Conclusion

In this paper, we provide a solution to the problem of bandwidth investing in private BitTorrent communities. We show how this problem occurs in the first place when oversupply makes it hard for honest users to contribute their upload bandwidth. The solution design is based on speculative downloading: identifying swarms in need of upload bandwidth in the community, downloading, and seeding them. At the heart of the solution is the Speculative Download Engine (SDE)—a prediction algorithm which we implement using multiple regression.

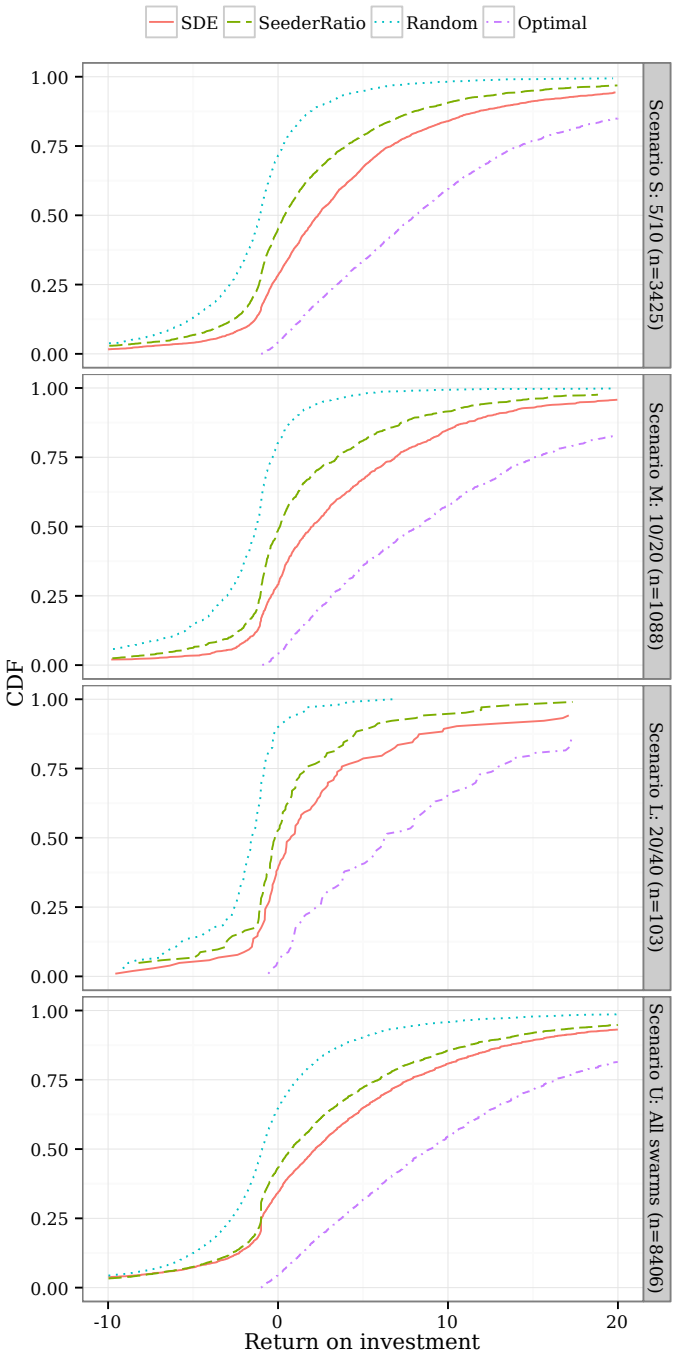

Figure 4: CDF of the return on investment for the SDE, SeederRatio, Random, and Optimal algorithms, in all four scenarios, when Optimal gain > 0. (n denotes the number of users in the experiment.)

Table 3: Performance when there is no potential gain

	S	M	L	U
Correct (%)	68	75	100	63
Median mistakes (%)	100	33	NA	100
First quartile loss	0.008	0.033	NA	0
Median loss	0.033	0.023	NA	0.091
Third quartile loss	0.132	0.019	NA	0.095

We test our solution with real-world data from a private BitTorrent community with more than 80 000 peers. The results show that SDE is successful for different types of input scenarios and that it produces a sizeable return on investment. In most situations when there is no possibility for gain, SDE correctly predicts a cache replacement should not take place.

In future work, we plan to deploy SDE in a BitTorrent client and conduct a live Internet performance evaluation. This will also reveal the effects of repeatedly running SDE for a peer, as well as the effects of multiple peers collectively using SDE. We hypothesize that, while the positive effect of using SDE will diminish when deployed throughout the community, there will be no negative effect for download performance, since using SDE leads to an increase in available upload bandwidth. Because the presence of other SDE-enabled peers in swarms will be reflected in the swarm characteristics, i.e., the number of seeders will increase, SDE will predict less potential for gain and will make fewer replacement decisions.

Acknowledgments

This work was partially supported by the European Union's Seventh Framework Programme through projects P2P-Next and QLectives (grant numbers 216217 and 231200).

References

[1] M. Capotă, N. Andrade, J. A. Pouwelse, and D. H. J. Epema, "Investment Strategies for Credit-Based P2P Communities," in *Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pp. 437–443, IEEE, Feb. 2013. 1

[2] A. Legout, G. U. Keller, and P. Michiardi, "Rarest first and choke algorithms are enough," in *ACM IMC*, pp. 203–216, ACM, 2006. 4, 5

[3] C. Zhang, P. Dhungel, D. Wu, and K. W. Ross, "Unraveling the BitTorrent ecosystem," *IEEE TPDS*, vol. 22, pp. 1164–1177, July 2011. 4

[4] M. Meulpolder, L. D'Acunto, M. Capotă, M. Wojciechowski, J. Pouwelse, D. Epema, and H. Sips, "Public and private bittorrent communities: A measurement study," in *IPTPS*, 2010. 4, 5

[5] Z. Liu, P. Dhungel, D. Wu, C. Zhang, and K. W. Ross, "Understanding and improving ratio incentives in private communities," in *IEEE ICDCS*, pp. 610–621, IEEE, June 2010. 4, 5

[6] A. L. Jia, R. Rahman, T. Vink, J. Pouwelse, and D. Epema, "Fast download but eternal seeding: The reward and punishment of sharing ratio enforcement," in *IEEE P2P*, pp. 280 – 289, IEEE, 2011. 4, 5

[7] D. Wu, Y. Liu, and K. W. Ross, "Modeling and analysis of multichannel P2P live video systems," *IEEE/ACM TON*, vol. 18, pp. 1248–1260, Aug. 2010. 4, 5

[8] D. Qiu and R. Srikant, "Modeling and performance analysis of BitTorrent-like peer-to-peer networks," *SIGCOMM Comput. Commun. Rev.*, vol. 34, pp. 367–378, Oct. 2004. 5

[9] R. Rahman, T. Vinkó, D. Hales, J. Pouwelse, and H. Sips, "Design space analysis for modeling incentives in distributed systems," *SIGCOMM Comput. Commun. Rev.*, vol. 41, pp. 182–193, Aug. 2011. 5

[10] 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 *IPDPS*, pp. 1–8, IEEE Computer Society, 2009. 5

[11] M. Piatek, T. Isdal, A. Krishnamurthy, and T. Anderson, "One hop reputations for peer to peer file sharing workloads," in *NSDI*, pp. 1–14, USENIX, 2008. 5

[12] J. Terrace, H. Laidlaw, H. E. Liu, S. Stern, and M. J. Freedman, "Bringing P2P to the Web: Security and privacy in the Firecoral network," in *IPTPS*, USENIX, 2009. 5

[13] S. Iyer, A. Rowstron, and P. Druschel, "Squirrel: a decentralized peer-to-peer web cache," in *PODC*, pp. 213–222, ACM, 2002. 5

[14] A. Wierzbicki, N. Leibowitz, M. Ripeanu, and R. Wozniak, "Cache replacement policies revisited: the case of P2P traffic," in *IEEE CCGrid*, pp. 182–189, IEEE, 2004. 5

[15] M. Hefeeda and O. Saleh, "Traffic modeling and proportional partial caching for Peer-to-Peer systems," *IEEE/ACM TON*, vol. 16, pp. 1447–1460, Dec. 2008. 5

[16] N. Carlsson, D. L. Eager, and A. Mahanti, "Using torrent inflation to efficiently serve the long tail in Peer-Assisted content delivery systems," in *IFIP NETWORKING*, vol. 6091, ch. 1, pp. 1–14, Springer Berlin / Heidelberg, 2010. 5

[17] P. Garbacki, D. Epema, and M. van Steen, "An amortized tit-for-tat protocol for exchanging bandwidth instead of content in p2p networks," in *IEEE SASO*, pp. 119–128, IEEE, 2007. 5

[18] D. Nie, Q. Ma, L. Ma, and W. Tan, "Predicting peer offline probability in BitTorrent using nonlinear regression," in *ACM ICEC*, vol. 4740, ch. 40, pp. 339–344, Springer Berlin / Heidelberg, 2007. 5

[19] A. H. Rasti and R. Rejaie, "Understanding peer-level performance in BitTorrent: A measurement study," in *IEEE ICCCN*, pp. 109–114, IEEE, Aug. 2007. 5

[20] A. Norberg, G. Hazel, and A. Grunthal, "Extension for partial seeds," bittorrent extension proposal, BitTorrent, 2008. [Online] `http://bittorrent.org/beps/bep_0021.html`. 6

[21] J. H. Friedman, "Multivariate adaptive regression splines," *The Annals of Statistics*, vol. 19, no. 1, pp. pp. 1–67, 1991. 7