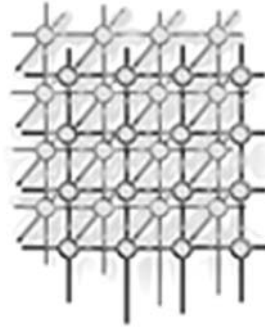


Tribler: A social-based peer-to-peer system

J.A. Pouwelse¹, P. Garbacki¹, J. Wang¹, A. Bakker²,
J. Yang¹, A. Iosup¹, D.H.J. Epema^{1,*} M. Reinders¹,
M.R. van Steen², and H.J. Sips¹



¹ Department of Computer Science, Delft University of Technology, the Netherlands

² Department of Computer Science, Vrije Universiteit, the Netherlands

SUMMARY

Most current P2P file-sharing systems treat their users as anonymous, unrelated entities, and completely disregard any social relationships between them. However, social phenomena such as friendship and the existence of communities of users with similar tastes or interests may well be exploited in such systems in order to increase their usability and performance. In this paper we present a novel social-based P2P file-sharing paradigm that exploits social phenomena by maintaining social networks and using these in content discovery, content recommendation, and downloading. Based on this paradigm's main concepts such as taste buddies and friends, we have designed and implemented the TRIBLER P2P file-sharing system as a set of extensions to Bittorrent. We present and discuss the design of TRIBLER, and we show evidence that TRIBLER enables fast content discovery and recommendation at a low additional overhead, and a significant improvement in download performance.

1. Introduction

Traditional P2P file-sharing systems focus exclusively on technical issues and are therefore unable to leverage the power of social phenomena. However, we believe that social phenomena such as friendship, trust, and a sense of community may be at least as important as technical issues, and may indeed have a large positive impact on the usability and performance of P2P file-sharing and content-delivery systems. For example, viewing users as *social partners* rather than as solitary *rational agents* [12] could alleviate the problem of freeriding [3] by exploiting the fact that people tend not to steal (bandwidth) from the social group they belong to.

To confirm our belief, we propose in this work a novel *social-based P2P file-sharing paradigm*, which facilitates the formation and maintenance of social networks and exploits social phenomena for improved content discovery, recommendation, and sharing. Our contribution is threefold. First, we relate the social-based P2P file-sharing paradigm to the current research challenges in P2P research

*Correspondence to: d.h.j.epema@tudelft.nl, P.O. Box 5031, 2600 CD, Delft, the Netherlands



(Section 2). Second, we present the design and implementation of TRIBLER, which adheres to the social-based paradigm by adding social-based functionality to the widely used Bittorrent system (Section 3). To facilitate the formation and maintenance of social networks, TRIBLER introduces permanent user identifiers, and will in the future be able to import existing user contacts from other social networks such as MSN (Section 4). Rather than using direct content-based searching, TRIBLER performs content discovery and recommendation based on the notion of *taste buddies*, that is, users with the same tastes or interests (Section 5). Third, we show evidence that TRIBLER achieves a significant improvement in download performance, by invoking the joint efforts of social peer groups (Section 6). In Section 7 we discuss related work, and in Section 8 we present our conclusions and ideas for future work. The full TRIBLER documentation and source code are available from <http://Tribler.org>.

2. Research challenges in P2P file sharing

With current P2P file-sharing systems continuously having more than 1,000,000 users, their performance and behavior have become of great interest. Starting in 2003, we have studied the performance of Bittorrent [15], which has for a number of years been the most popular P2P file-sharing system. Based on this work and on related studies, we formulate the following five grand research challenges for P2P file sharing, and we argue for the importance of the social-based paradigm in solving these challenges. In particular, with our social-based P2P network called TRIBLER we want to address all these challenges.

The most difficult research challenge is the *decentralization* of the functionality of a P2P system across its peers. Full decentralization eliminates the need for central elements in the system, which must be set up and maintained by some party and which may form serious bottlenecks, points of failure, or security threats. In particular, connecting to the network and validating user identities are difficult to implement without any central element. To date, no P2P file-sharing system exists which fully decentralizes all functionality efficiently and without a high risk of a loss of integrity. Bittorrent is not fully decentralized as it depends on web sites and trackers for finding content. Social groups form a natural method to efficiently decentralize P2P systems because people/peers who know each other tend to exchange information.

The second challenge is to guarantee the *availability* of a P2P system as a whole. The operation of such a system should not depend on the availability of any particular participating peer, or of any central component (something we don't want if we aim for decentralization), as the failure of such a component can be disruptive to service [15]. Given the short periods of availability of peers (in [15] we found less than 4% of the peers to have an uptime of over 10 hours), the availability problem is critical. Proven social incentives such as rewards and social recognition could stimulate users to leave their P2P software running for longer periods, thus improving the overall availability of the network.

The third challenge is to maintain the *integrity* of the system and to achieve *trust* among peers. By definition, P2P systems use donated resources. However, donors cannot always be trusted, and maintaining system integrity has proven to be difficult in operational P2P systems [7]. Data at several levels can be attacked in a P2P system, namely system information (e.g., pointers to content), metadata, and the actual content itself. This significant problem, often ignored by P2P system designers, can be

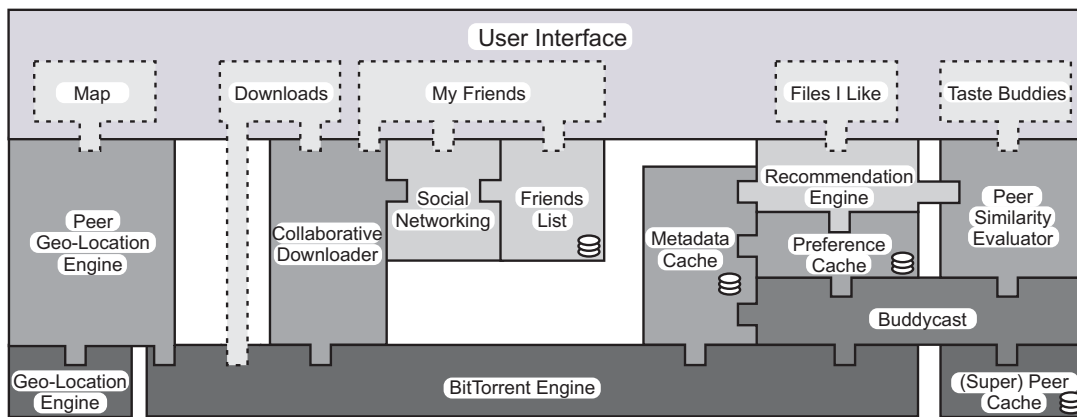


Figure 1. The system architecture of TRIBLER.

solved with a social-based network, in which users actively help to clean or remove polluted data and users can select trustworthy representatives.

The performance of a P2P system highly depends on peers donating resources. Even though the resource economy is by definition balanced (e.g., every MByte downloaded corresponds to a MByte uploaded), autonomous peers are free to decide whether to donate resources or not. Hence, as the fourth challenge, *providing proper incentives* is vital to induce cooperation and to achieve good performance [3]. Again, social recognition can help to alleviate this problem.

The fifth challenge in P2P systems is to achieve *network transparency* by solving the problems caused by dynamic IP addresses, NAT boxes, and firewalls. The Internet has fundamentally changed due to the wide-spread use of these three technologies, and as a consequence, peers no longer have the freedom to send anything anywhere without the help of another peer acting as a mediator. Social networks enable communicating peers to automatically select trusted mediators from the members of their social proximity that are online, thus eliminating the need for fixed mediators.

3. The architecture of Tribler

In this section, we present the architecture of our TRIBLER social-based P2P file-sharing system, which is built on top of the Bittorrent protocol. Figure 1 depicts this architecture, with rectangles representing its modules and files, and extrusions representing *makes-use-of* relationships. To achieve backwards compatibility with the existing Bittorrent P2P system, we have only made modifications and extensions to the existing Bittorrent client software. We have based our system on the popular ABC open-source Bittorrent client [1] so as to have a tested code base for our implementation and a large user base in a relatively short period of time. We now discuss several important concepts in TRIBLER.

Social groups: The prime social phenomenon that we exploit in TRIBLER is that “kinship fosters cooperation” [13]. In other words, a similar taste for content can form the foundation for an online community with altruistic behavior. In order to implement effective social groups in TRIBLER, we



use an approach borrowed from evolutionary biology (see for instance [13]): we have implemented the ability to distinguish friend, foe (e.g., a polluter), and newcomer. For this, we de-anonymize peers and facilitate social-group formation. De-anonymization is achieved by having every user choose a nickname; TRIBLER transfers user nicknames between users automatically. The *Social Networking* module in Figure 1 is responsible for storing and providing information regarding social groups (the group members, their recently used IP numbers, etc.).

Megacaches: Virtually all current P2P file-sharing systems lack persistent “memory” about previous activity in the network; peers usually exchange queries for files and file metadata, but completely ignore other types of information. The *context information* that needs to be saved in order to improve the performance consists of information on social relations, altruism levels, peer uptimes, taste similarities, etc. In TRIBLER, every piece of context information received by a peer that is relevant to it based on its interests and tastes is stored locally in its so-called *Megacaches*, and this information is exchanged within social groups using an epidemic protocol [10] (the Buddycast protocol, see Section 5). The small database icons in Figure 1 identify the four Megacaches in TRIBLER, which are the *Friends List* with information on social networks, the (*Super*) *Peer Cache* with information on superpeers and peers in general, the *Metadata Cache* with file metadata, and the *Preference Cache* with the preference lists of other peers. The sizes of the *Friends List*, the (*Super*) *Peer Cache*, and the *Preference Cache* are below 10 MB at any time, which for the *Preference Cache* is enough to store the preference lists of thousands of taste buddies (see also Section 5).

The main problem concerning the Megacaches is the overhead traffic required to keep them up-to-date. For the *Metadata Cache*, we have observed that in Bittorrent the number of newly injected files per day is limited to roughly 1500 [15] when *content pollution* [7] is kept to a minimum. Then, by reducing the average size of the metadata for each file to just 400 bytes using Merkle hashes [11], the overhead is reduced to approximately 600 KBytes/day. With this amount of overhead, all metadata can be replicated among all peers, moving content discovery from network-based keyword searching to local metadata browsing.

Taste-buddy-based content discovery: Locating content is critical for P2P systems. Current solutions are based on one or a combination of query flooding, distributed hash tables, and semantic clustering. We take a next step by connecting *people* with similar tastes called *taste buddies* instead of focusing on *files*, and by using full metadata replication.

Using the *Files I like* module (see Figure 1), each peer indicates its preference for certain files expressed as a number between 1 and 5. By default, the preference list of a peer is filled with its most recent downloads. We have developed an algorithm called *Buddycast* which uses an epidemic protocol to exchange preference lists using the overlay swarm (see Bootstrapping) and which can efficiently discover a user’s taste buddies (see Section 5). The *Peer Similarity Evaluator* module in Figure 1 is able to compare preference lists and determine the similarity in taste of two peers, which is measured as the cosine of their vectors of user ratings of content.

The *Recommendation Engine* module is able to compile a list of files a user most likely wants. First, a user-item rating matrix is built from the preference lists [5], and then a user-based recommendation is generated by TRIBLER, based on standard collaborative filtering techniques. Finally, the user interface facilitates metadata browsing by augmenting each file entry with the estimated interest to the user.

Downloading: The *Bittorrent Engine* module in Figure 1 downloads files using a Bittorrent-compatible protocol. This module can also use the *Collaborative Downloader* module’s capabilities



Table I. Performance of the TRIBLER system.

Test description	Performance [ops/s]
Geo-lookup	1730
Overlay swarm connect	7045
Connect+challenge/response	865
Connect+challenge+Buddycast	844

to achieve a significant increase in file download speed by exploiting idle upload capacity of online friends (see Section 6).

User interface: The user interface is key to making a social-based network usable and, as such, is a critical part of TRIBLER. The *User Interface* module from Figure 1 is split into five components: *Map*, *Downloads*, *My Friends*, *Files I like*, and *Taste Buddies*. The use of the *Downloads*, *Files I like*, and *Taste Buddies* modules is clear from the description above. The main goal of the interface is to facilitate the formation of social groups. For this purpose, the *My Friends* module clearly displays the friends, the friends-of-friends, and the taste buddies of the user. This *visual proximity* gives the user a more personal contact with his peers, and may help reduce asocial behavior.

Another goal of the user interface design was to ease the process of visual identification of potential collaborators. When a user is downloading a file, the observed IP addresses of members of the corresponding download swarm are geo-located and then displayed on a world map using the *Map* module. We have built a *Peer Geo-Location Engine* module on top of a freely available *Geo-Location Engine* (<http://hostip.info>). The user interface also facilitates the use of the *Collaborative Downloader* module: The user can see which friends helped him in the past, which friends he donated bandwidth to, and which friends who are currently online can help speedup his new downloads.

Bootstrapping: Finding other peers in a P2P systems after software installation is called bootstrapping. In Bittorrent, peers have to repeatedly connect to a tracker in order to discover other peers. Furthermore, the original Bittorrent protocol restricts communication to within the swarms of peers that download the same files, making the bootstrapping process unnecessarily repetitive. To solve the bootstrapping problem in TRIBLER, we use two mechanisms. First, a TRIBLER peer automatically contacts one of a set of pre-known *superpeers* only once immediately after installation in order to obtain an initial list of other peers in the system (through a Buddycast message, see Section 5), so that it can start participating in the epidemic information dissemination in TRIBLER. Second, we define a special *overlay swarm*, which is a swarm of which every TRIBLER peer is a member that has no tracker and that is used for content and peer discovery, again using Buddycast.

In order to assess the overhead incurred by some of the operations in the TRIBLER protocol, we have done some performance tests on a powerful computer (a 4-CPU 2.0 GHz machine with 16 GB of main memory and a 1 Gb/s Internet connection). Table I shows the results of these tests. The first test determines the number of geo-lookups this machine can handle per second. The second test shows the performance of connecting to a peer in the overlay swarm with the standard Bittorrent protocol messages. In the third test, in addition also the public key of the peer that connects is validated using a challenge/response algorithm (see Section 4). The fourth test adds to the third test the exchange of a



preference list with 100 file names by means of the Buddycast algorithm. These tests show that a single peer can handle a significant workload, although indeed a certain amount of overhead is added to the Bittorrent protocol.

4. Social networking

A fundamental limitation in most file-sharing systems is the *session boundary*—all context information is lost when a user disconnects from the network. Due to dynamic IP numbers, it is difficult to store context information about peers across sessions. Storing long-term context information in databases like our Megacaches enables the existence of trust-based social groups, but only if the user identities are stable. To solve this problem, we have introduced permanent, unique, and secure peer identifiers (*PermIDs*), which are the public keys of a public-key scheme using elliptic-curve cryptography, and which are exchanged by e-mail. To prevent peers from faking the identity belonging to a PermID (*spoofing*), we have implemented a challenge-response mechanism for validating PermIDs. In this mechanism, a random number selected by the challenger is encrypted by the challengee with its private key and subsequently decrypted by the challenger with the corresponding public key—authentication succeeds when the result is identical to the original number. Social-network creation in TRIBLER will be facilitated by the ability to import contacts from other social networks of which peers are members, such as MSN and GMail.

We will use Bloom filters [6], which are very dense hash-table-like data structures for storing and (probabilistically) testing set membership, for distributing and pairwise comparing the contents of the Megacaches. Because of their reduced size, Bloom filters can significantly reduce the bandwidth requirements of epidemic protocols, which are the basis of our solution for content discovery and social networking. In a recent prototype [16] of a design for decentralized swarm discovery in Tribler, we have already implemented Bloom filters for filtering peers from messages that are already known to their destinations.

The size of a Bloom filter depends on the number of expected connections peers have. To assess their size while we do not yet have sufficient information on the operation of TRIBLER, we resort here to analyzing another existing social network, `Friendster.com`. We have created a crawler for this network and we have extracted 3.3 million relations between 27,000 people. Figure 2 shows the probability density of the numbers of friends and friends-of-friends in `Friendster.com`. In this network, we find that a person has on average 243 friends and 9,147 friends-of-friends. These figures are to within an order of magnitude similar to the figures reported in [14] for several P2P file-sharing networks. Based on these numbers, we can compute that 260 bytes are needed to discover the common friends-of-friends of two peers using a Bloom filter. This very low bandwidth requirement will enable TRIBLER peers to exchange information with *thousands* of other peers in a short time frame, which is a significant improvement over traditional epidemic protocols.

5. The Buddycast algorithm

In order to make recommendations and to enable peer and content discovery, and more generally, to disseminate the contents of the Megacaches in the TRIBLER overlay swarm, we have designed the

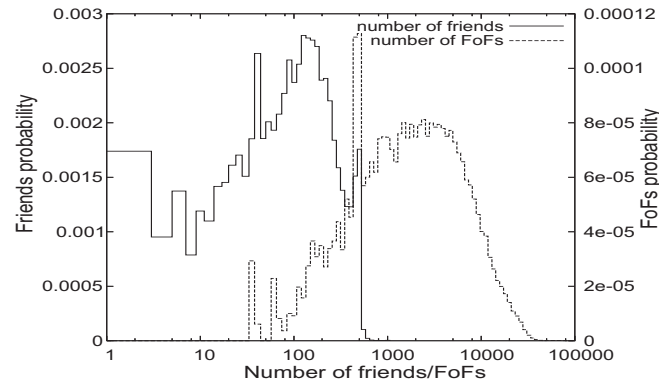


Figure 2. The probability density of the numbers of friends and friends-of-friends (FoFs) in Friendster.com.

Buddycast algorithm, which is an epidemic protocol that works as follows. Each peer maintains in its Megacache a number of taste buddies with their preference lists and a number of random peers (i.e., peers for which no preference list is known). Periodically (in the current version of TRIBLER every 15 seconds), a peer either connects to (a) one of its taste buddies (*exploitation*) or to (b) a random peer (*exploration*) to exchange a *Buddycast* message, or it replies to such a message received from another peer. In contrast to other epidemic protocols such as Newscast [10], we use both exploitation and exploration, we limit the randomness of peer selection during the exploitation, and we implicitly cluster similar peers into (trusted) social groups. This is very close to the approach proposed in [17].

A *Buddycast* message contains the identities of a number (currently 10) taste buddies along with their top-10 preference lists, a number (currently also 10) of random (and fresh, see below) peers, and the top-50 preferences of the sending peer. The age of each peer is included in the message to help others know the freshness of peers. After exchanging *Buddycast* messages, both peers merge the information in the message received into their own Megacache. To maximize the exploration and avoid redundant messages, every peer also maintains a list with the most recently contacted random peers, and avoids reconnecting to the peers in this list for a certain period of time (currently 4 hours). Users can disable the functionality of *Buddycast* to protect their privacy.

To find a good balance between exploration and exploitation, an *exploration-to-exploitation ratio* λ for selecting either a random peer or a taste buddy is used, which is currently set to 1, but which we may want to make adaptive. In the case of exploration, a random peer is selected based on its freshness, to improve the chance of its still being online when connecting to it. In the case of exploitation, a taste buddy is selected from a number (currently 100) of the most recently contacted taste buddies based on its similarity with the sending peer. As a consequence, the fresher or more similar a peer, the higher its chance to be selected.

For a first assessment of the operation of the *Buddycast* algorithm, we have investigated the availability of the peers contained in *Buddycast* messages at the moment of receiving such messages. To this end, we have run a continuously fresh TRIBLER peer for a period of 519.7 hours, and we have

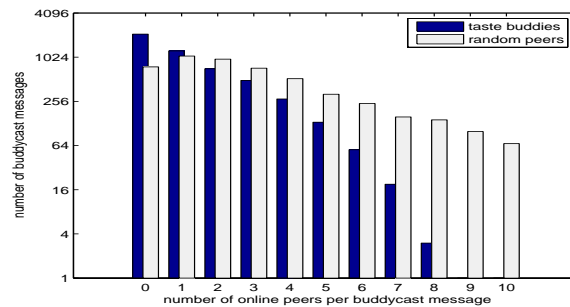


Figure 3. The availability of peers in Buddycast messages.

probed all the peers in every Buddycast message it received to see how many of them were still online. At the time of this experiment, there were about 15,000 TRIBLER clients in the world who had actually downloaded files. The results are shown in Figure 3 (please note the logarithmic scale on the vertical axis). During the experiment, our peer received 5,049 messages in total. As can be seen in the figure, often at most only 4 taste buddies in the Buddycast messages were still online, which agrees with our earlier finding that online periods in Bittorrent are short [15]. From the figure we also see that random peers have a higher chance to be online, which indicates that selecting peers based on freshness can benefit the effectiveness of exploration.

6. Collaborative downloading

In this section we present the *collaborative download protocol* called 2Fast that we have developed for TRIBLER, in which a user invokes the help of his friends to speed up downloads. Early downloading protocols (e.g., Gnutella) have no incentives for donating upload bandwidth, which has serious limitations in real environments, because unconstrained bandwidth sharing is sensitive to freeriding [3]. The Bittorrent tit-for-tat mechanism was the first system which offered an incentive for uploading. The current Bittorrent mechanism also has its disadvantages, because without enough seeding peers (i.e. peers which possess the complete file in question), the download speed of a peer is restricted by Bittorrent's tit-for-tat bartering protocol to its upload link capacity. Hence, peers with asymmetric Internet access, such as ADSL or ADSL-2, cannot fully use their download capacity.

The 2Fast protocol makes use of social groups, where members who trust each other collaborate to improve their download performance (see Figure 4). The idea of downloading with the help of others was first introduced in [18], where altruistic peers contribute their bandwidth by joining a swarm even if they are not interested in the content being distributed in this swarm. The inherent assumption of sufficient altruism in the network without any incentives makes this simple approach impractical in real-world environments. Our 2Fast protocol solves this problem by introducing social-group incentives. For a full account of 2Fast, see [9].

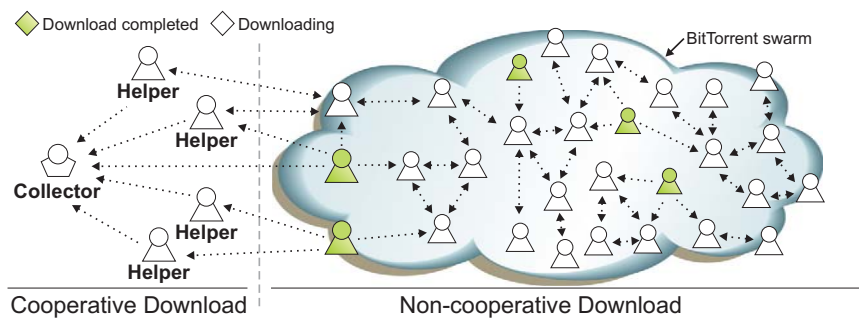


Figure 4. Overview of collaborative downloading.

In 2Fast, peers from a social group that decide to participate in a collaborative download take one of two roles: they are either *collectors* or *helpers*. A collector is the peer that is interested in obtaining a complete copy of a particular file, and a helper is a peer that is recruited by a collector to assist in downloading that file. Both the collector and the helpers start downloading the file using the classical Bittorrent tit-for-tat and the collaborative download extensions. As in Bittorrent, a helper selects a chunk of the file for downloading based on the rarest-first policy among the chunks in possession of its bartering partners. However, before actually downloading this chunk, it asks the collector for approval, which will only be granted when the chunk is unique, that is, when no other helper already downloads the same chunk. After downloading a file chunk, the helper sends the chunk to the collector without requesting anything in return. In addition to receiving file chunks from its helpers, the collector also optimizes its download performance by dynamically selecting the best available data source from the set of helpers and other peers in the Bittorrent network using the default mechanisms of Bittorrent, which prefer peers that upload at higher rates. As helpers give priority to collector requests, they are preferred as data sources.

We have implemented and tested 2Fast in a real environment. For this we have selected a middle-sized swarm of around 1,900 peers with only 6% seeds, distributing a 1.2 GB file. These numbers remained almost unchanged during our experiments. We have performed tests for three download-to-upload bandwidth ratios from standard Internet package offerings: low-end ADSL with a ratio of 512:128 Kbps, high-end ADSL with a ratio of 2048:512 Kbps, and ADSL-2 with a ratio of 8:1 Mbps. It should be noted that we could only impose these bandwidths on the collector and the helpers, which were under our control, but not on the peers in the Internet that they selected as their bartering partners.

As a performance metric of our system, we use the ratio between the download time achieved by a peer obtaining a file all by itself versus the corresponding time for a collaborative group (*speedup*). The theoretical maximum speedup is achieved when a peer can fully use its download bandwidth, and so it is equal to the ratio between the download and upload link capacities. Thus, for ADSL and ADSL-2 the maximum achievable speedup is 4 and 8, respectively.

Figure 5 shows the obtained speedups for the number of helpers in the range from 0 to 32. The total download time was decreased with a factor of almost 2 for low-end ADSL, of more than 3 for high-end ADSL, and of almost 6 for ADSL-2. The difference between the theoretical and achieved speedups is mainly due to the influence of the seeders (when there are many seeders to begin with, the potential

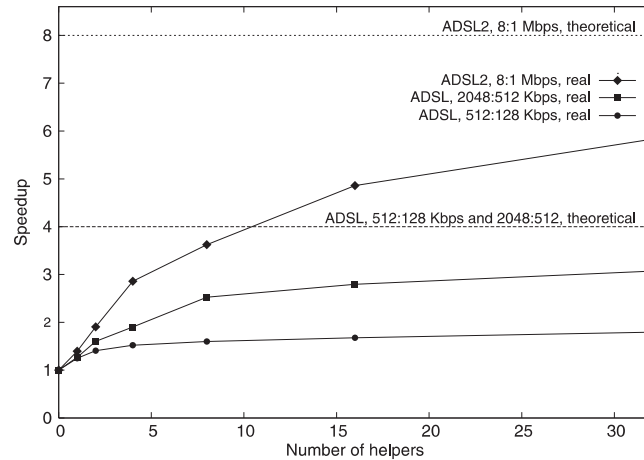


Figure 5. The impact of the number of helpers on the download speedup.

speedup is restricted) and of the delays experienced by helpers when requesting unique file chunks from peers. The more helpers are involved, the more restrictive the unique file chunk selection method is, and consequently, the longer the time needed to find a bartering partner for such a chunk. This time is further increased in the case of low-end ADSL by the fact that then the collector and the helpers were discriminated as peers with an upload bandwidth which is below average [15].

7. Related work

The idea of exploiting the natural social connections between the humans behind the computers in large-scale P2P networks is starting to become a major research topic. To date, methods based on social clustering were applied in P2P networks to limited aspects of content distribution [8], user communities formation [4], and collaborative service provisioning [5]. In [8], a system which uses knowledge discovery techniques for overlay network creation is presented. By automatically clustering users based on their preferences, the system enables content location and improves the performance of content sharing. In [4], a simple general-purpose system is proposed which groups peers based on the similarity of their keyword searches. The authors give evidence on how their system can be used to form and maintain communities of users. An extensive experimental analysis of several collaborative filtering methods is given in [5]. TRIBLER is the first system which exploits social phenomena to address all the research challenges in P2P file-sharing networks mentioned in Section 2. The BTSlave project [2] extends the Bittorrent protocol by introducing special-purpose peers called *repeaters* that use their idle bandwidth to increase content replication in the swarm. In contrast to Tribler, in BTSlave a peer is not aware of the help it is receiving, and so it cannot coordinate its helpers as the collector



in 2Fast does. Due to the lack of coordination, BTSslave repeaters can end up competing for the same content instead of collaborating.

8. Conclusion and future work

In this paper we have presented a novel paradigm for the design of P2P file-sharing networks based on social phenomena such as friendship and trust. Following the paradigm's main concepts of taste buddies and friends, we have designed and implemented the TRIBLER P2P file-sharing system. We have described how TRIBLER can help to automatically build a robust semantic and social overlay on top of Bittorrent, one of the most popular P2P file-sharing systems. We have shown how various TRIBLER components can yield good performance with respect to existing solutions. In particular, we have presented evidence that collaborative downloading yields a significant speedup when used in a real Bittorrent swarm. Last but not least, we have shown how with TRIBLER we have made a start in addressing the five major P2P research challenges introduced in Section 2.

As future work, we intend to extend TRIBLER with a reputation system, with tag-based content navigation, with video-on-demand, and with application-level multicasting for video streaming. In addition, we are planning a much more detailed analysis of the Buddycast algorithm.

REFERENCES

1. <http://sf.net/projects/pingpong-abc>.
2. <http://btslave.sourceforge.net>.
3. E. Adar and B. A. Huberman. Free Riding on Gnutella. Technical report, Xerox PARC, August 2000.
4. N. Borch. Social Peer-to-Peer for Social People. In *The Int'l Conf. on Internet Technologies and Applications*, Sep 2005.
5. J. S. Breese, D. Heckerman, and C. Kadie. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proc. of UAI*, 1998.
6. A. Broder and M. Mitzenmacher. Network Applications of Bloom Filters: A Survey. In *40th Conference on Communication, Control, and Computing*, 2002.
7. N. Christin, A.S. Weigand, and J. Chuang. Content Availability, Pollution and Poisoning. In *ACM E-Commerce Conference*. ACM, June 2005.
8. A. Fast, D. Jensen, and B. N. Levine. Creating Social Networks to Improve Peer-to-Peer Networking. In *11th ACM SIGKDD*, Aug 2005.
9. P. Garbacki, A. Iosup, D. H. J. Epema, and M. van Steen. 2Fast: Collaborative Downloads in P2P Networks. In *6-th IEEE Int'l Conference on Peer-to-Peer Computing*, pages 23–30. IEEE Computer Society, Sept 2006.
10. M. Jelasity and M. van Steen. Large-scale newscast computing on the Internet. Technical Report IR-503, 2002.
11. R.C. Merkle. A Digital Signature Based on a Conventional Encryption Function. In *CRYPTO '87*, pages 369–378. Springer-Verlag, 1988.
12. S. J. Nielson, S. A. Crosby, and D. S. Wallach. A Taxonomy of Rational Attacks. In *4th Int'l Workshop on Peer-to-Peer Systems (IPTPS)*, volume 3640 of *LNCIS*, pages 36–46. Springer-Verlag, Feb 2005.
13. E. Pennisi. How Did Cooperative Behavior Evolve? *Science*, 309(5731):93, July 2005.
14. L. Plissonneau, J.-L. Costeux, and P. Brown. Analysis of Peer-to-Peer Traffic on ADSL. In C. Dovrolis, editor, *Passive and Active Measurements Conference*, volume 3431 of *LNCIS*, pages 69–82. Springer, 2005.
15. J. A. Pouwelse, P. Garbacki, D. H. J. Epema, and H. J. Sips. The Bittorrent P2P File-sharing System: Measurements and Analysis. In *4th Int'l Workshop on Peer-to-Peer Systems (IPTPS)*, volume 3640 of *LNCIS*, pages 205–216. Springer-Verlag, Feb 2005.
16. J. Roozenburg. Secure Decentralized Swarm Discovery in Tribler. Master's thesis, Delft University of Technology, 2006.
17. S. Voulgaris and M. van Steen. Epidemic-style Management of Semantic Overlays for Content-Based Searching. In *Euro-Par 2005*, volume 3638 of *LNCIS*, pages 1143–1152. Springer-Verlag, Aug 2005.
18. J. Wong. Enhancing Collaborative Content Delivery with Helpers. Master's thesis, The University of British Columbia, 2004.