Collabory: A Collaborative Throughput Stabilizer & Accelerator for P2P Protocols

Shay Horovitz and Danny Dolev
Hebrew University of Jerusalem
{horovitz,dolev}@cs.huji.ac.il

Abstract

Common peer-to-peer (P2P) file sharing clients usually download at an unstable rate and hardly exploit the available bandwidth offered by low rate sources. The characteristic fluctuational throughput of the source peers might be caused by user behavior factors such as running other bandwidth consuming tasks, throttling of download speed by P2P software or even termination of the source.

In this paper we propose Collabory - a solution for stabilizing and accelerating the download speed rate in existing P2P networks. We introduce a new role: "Feeders" - peers that collaboratively aggregate the downloads from multiple sources into a single, stable stream served to the downloading peer. We show that the solution utilizes source nodes with an extremely low and unstable throughput without reducing the download rate of the downloading peer.

Measurements in a test suite expressed a major increase in download rate and stability. Upgraded & stabilized throughput is demonstrated on eMule.

1. Introduction

Following the increased popularity of P2P, came a corresponding increase in the amount of research on the lookup problem. Yet, the common method being used for the data transfer between peers still relies on traditional client-server techniques and ignore the unique characters of P2P networks. We noticed a significant difference in download rate stability on the client side as it downloads from P2P peers compared to downloading from a server. In a home machine the throughput of the internet connection is directly affected by the user's behavior - running different bandwidth and cpu demanding tasks in an unpredictable manner resulting in frequently sudden changes in the effective throughput and instability in the download rate.

In this paper we address the problem of throughput stability in P2P networks. We discuss the causes for such instability and offer a design that can be easily adapted to any P2P network for creating a much more stable download rate and respectively gain higher throughput. Instability of the download rate stems from several reasons, which we categorize as network related, P2P network design and user behavior aspects.

Network infrastructure aspects that affect the stability of a peer are not under the control of the user. Large delay variation (jitter), excessive packet delays and round trip time reduce the ability to ensure a stable connection. Packet loss might also contribute to the download rate instability especially between different regions. In addition, ISP enforced policy controlled by traffic shapers downgrade P2P download rate at peak hours [6].

P2P network design aspects are connected to how the client searches for potential peers. In many networks, ping RTT measurements are used as a metric for better sources. However our experiments showed that this technique does not produce a stable behavior. Also, if a client has a technique for finding peers with maximal available upload rate, it does not guarantee that those peers are stable and the resulting throughput to be higher. Moreover, when such a source is heavily used, the disruption in case of a failure during the download session is much higher. In addition, P2P networks embed fairness rules for preventing free riders. This might affect the stability during the download phase as there is a certain ratio between the download rate and the amount of data a source uploads.

Yet, the aspect that we found having the greatest impact on download rate stability is the behavior of users at source peers. More specifically, actions that the user of the uploading peer machine occasionally takes might directly affect the upload rate of the machine. The most obvious occurrence is the case where the user at the source peer invokes applications that heavily use bandwidth such as Email clients, online games or other P2P applications. By doing so, the bandwidth available for the client connected to that machine may be drastically reduced and becomes significantly unstable.

In this paper we present Collabory, a collaboration tool utilizing helper peers for stabilizing P2P clients' download

rates. Helper peers are peers that voluntarily participate in the file download by pre-fetching and caching parts of the file for the client. The idea of utilizing helper peers was mentioned in [12, 4, 7, 11], however, they all offered using helper peers (in addition to normal source peers) for bypassing the limitations of fairness rules that are embedded in P2P networks [3, 1]. Moreover, the helpers selection criteria in these papers does not guarantee better stability compared to the stability of normal source peers i.e., the fluctuational rate of the sources and helpers is still reflected in the rate of the client. Therefore, the client is still exposed to stability problems due to network infrastructure, P2P network design and user behavior aspects.

Existing P2P protocols tend to ignore source peers that have relatively low bandwidth to offer. This dramatically reduces the amount of available sources and eventually the downloading peer ends up with insufficient available sources to maximize its throughput. Our solution enables utilizing extremely unstable peers and low bandwidth peers for maximizing the final throughput of the downloading client without being exposed to the sources instability. The above problems motivate the need for a new data transfer approach, adjusted to the P2P network characteristics and the behavior pattern of end users.

Collabory employs a helper-like architecture. Unlike previous works, it intentionally selects the helpers to be optimal for availability and throughput stability *with the client* by constantly measuring stability factors. We will term such helpers as *feeder* peers. The Feeders negotiate with potential source peers and aggregate the downloads from multiple unstable sources into a single, stable stream served to the downloading peer. The Feeders are employed exclusively as a means of delivering data to the client.

For estimating the level of fitness of a potential feeder we first optimize the availability of a feeder by matching a client with feeders that share similar online periods yet dissimilar in activity periods. This way we reduce the chance that feeders will disconnect intentionally while the client is downloading. Among the feeders that match, we will prefer those that are most predictable i.e., have repetitive overlapping online sessions with high probability. Furthermore, we strive to match feeders that have good network connectivity with the client. We constantly measure the jitter, RTT and packet losses and accordingly select those with the best connectivity measurements. In addition, we place a high priority on the effect of user behavior on the stability of a potential feeder. Thus we would prefer feeders with a fair amount of predictable spare bandwidth and if possible, low CPU usage. The incentive for serving as a feeder is by contributing machine resources in return to a higher throughput when one wishes to download.

To evaluate the concept, we built a test suite that monitors the download rate and its stability given different scenarios of network infrastructure problems as well as user behavior patterns at the source peers. The results of our experiments confirmed the following contributions of Collabory:

1. Ensuring a stable download rate resistant to source throughput instability with higher probability than in current techniques;
2. Allowing weak - low bandwidth sources to participate in the download process without compromising the client's download rate stability, thus increasing the number of potential sources;
3. Ability to locate potential sources and measure their throughput without compromising the client's download rate stability.

The remainder of this paper examines these issues both analytically and empirically and presents Collabory.

2. Related Work

The idea of utilizing helper peers was first introduced by Wong [12]. Guo et. al. [5] described a trade mechanism where peers exchange file portions. "2Fast" [4, 7] embeds helper peers for bypassing fairness rules. Wong offered a similar solution without the limit of choosing the helpers out of a social group and [11] recruited helper peers for increasing the number of sources on BitTorrent [3]. [9] offered a way to prevent the *last chunk problem* by cooperating between source peers.

The above papers offered techniques to increase the download rate of a peer by either bypassing fairness rules or by sharing fairness credit among a social group of peers. However, the helpers selection criteria in these papers do not guarantee better stability compared to the stability of normal source peers. Thus, the client is still exposed to stability problems due to network infrastructure, P2P network design and user behavior aspects. Moreover, the above solutions do not address low rate sources, frequent joins and leaves and unstable sources - which are characteristic to P2P networks and ignoring them opens a substantial gap between theory and reality.

3. The Problem of Throughput Stability in P2P

While performing some throughput tests for various P2P applications, we encountered a major difference in download rate stability of the client when downloading from peers using P2P software compared to downloading from a file server.

In a typical P2P network, the user connects to the network and searches for a file using keywords. Then the P2P

network returns a list of available files according to the keywords provided earlier. The user selects a file to be downloaded and in return, the P2P network provides a list of IP addresses, *L*, of peers that share portions of the requested file (this part is done automatically and is hidden from the user). Some P2P networks somewhat differ from the above description, yet provide a list of potential source peers. The part of the process that was described so far is also known as the Lookup problem. The Lookup problem was extensively researched and many publications presented innovative techniques for managing a directory of files in a P2P network [10, 8].

In this paper we will not discuss the mechanisms for creating the list of sources *L*. We will focus on various techniques that improve stability for a client that is interested in a file that is shared by peers listed in *L*. Obviously, there are current solutions to complete the download process as soon as *L* was created, yet the common techniques being used today present a download rate instability.

Examining file transfer analytically, we'd like to match the client with a set of sources that provide a stable and consistent rate of data. Since current P2P clients download portions of the file directly from the source peers that share the file, the stability of the download rate is bounded by the stability of the most stable sources in *L*. The source's stability might be degraded for many reasons such as packet loss, jitter, excessive delays or any other concurrently running software that compete on the same bandwidth and CPU resources.

Another issue that might affect the download rate of the client is the availability of sources. A source might become unstable or unavailable and might increase the total time required to download the file. Existing file transfer systems limit the probability to find stable sources even more, as they manage fairness rules to prevent selfish free-riders in addition to using queueing techniques; thus, in such systems *L* is much smaller than its theoretical potential size of all peers that share the file. In systems that allow downloading from sources that received only portions of the file, as in eDonkey [1], the client might need to generate several requests until the missing file portions are found; this increases the total time required to download the file.

In systems where peers can serve packets also before completing the download of the file [3, 1], the excessive use of upload bandwidth will not only harm the download performance of the source peer, but will also have an effect on P2P uploads, since the peer will not be able to download new content at a satisfying rate. In addition, in some ISPs [2] P2P sources are subject to traffic shaping rules [6], which again degrade the throughput.

All of the above issues are typical to P2P based file transfer stem from the direct link between the potential source available upload rate and the user's behavior; This motivates the need for a new data transfer approach, adjusted to the characteristics of P2P networks and the behavior pattern of the end user.

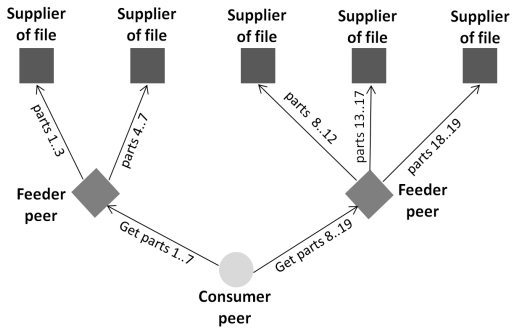

Figure 1. A one-level feeding structure

4. Collabory

Feeding Networks Model:
We present a model where the file transfer rate for subscribers is higher and the consumed bandwidth is more stable. Since we provide a generic solution it can be easily embedded in other works that present social groups or other bartering techniques such as tit-for-tat.

In our basic model, a peer that shares files (source) is called a supplier peer; a peer that requests files from supplier peers is called a consumer peer. Each consumer holds a list of feeders, which is a group of peers that can download the requested packets from supplier peers or other feeder peers and transfer them to the consumer peer. In general, when a consumer peer wishes to download a file, it will ask each of his feeder peers to supply him with different parts of that file. Each feeder will open one or more connections with supplier peers that share the requested file (see Figure 1).

The list of feeder peers for each consumer can be built manually or automatically by tracing the stability metrics between the feeder and the consumer during past network activities. There are three major parameters used in order to analyze the compatibility of a potential feeder to a given consumer peer: availability, capability and infrastructure properties.

Availability: We'd like the potential feeder peers to be online and have limited network and CPU consumption when the consumer is about to start a new download process. Therefore, we look for feeders that have a matching pattern of availability, meaning that they are likely to stay online and have low network and CPU consumption while the consumer is downloading. We'll use the term *fit* to address the above demands.

In order to find fitting feeders, we log feeders' online periods (sessions) and the relevant network use and CPU utilization measurements within these sessions.

We term *Feedability* as the ability of a feeder to feed a consumer peer at a specific point in time i.e., the feeder is online and has low network use and CPU consumption.

Denote a Feedability function FA of feeder f, in session s at time t (time units after session initiation time) as:

$$FA_{f,s}(t) = \begin{cases} 1 & f_{s_{cpu}}(t) < Th_{cpu} \wedge f_{s_{bw}}(t) < Th_{bw} \\ 0 & otherwise. \end{cases}$$

where $f_{cpu}(t)$ and $f_{bw}(t)$ are the measurements of cpu utilization and consumed upload bandwidth at after t time units from the beginning of session s (when the feeder went online). Th_{cpu} and Th_{bw} are the thresholds of cpu utilization and consumed bandwidth enabling the feeder to serve a consumer peer.

A potential feeder p is the most fitting feeder to a consumer peer (among all online feeders that have small RTT and low jitter with the consumer peer) if the average of its Feedability function $FA_{f,s}(t)$ over all of its sessions and a given time period starting now (when the consumer requested to start a new download) is maximized over all other feeders:

$$p = \underset{f}{\operatorname{argmax}} \int_{t}^{t+k} \sum_{i=1}^{n_f} \frac{FA_{f,s}(t)}{n_f} \ ,$$

where n_f is the number of sessions that were logged by feeder f. We choose k as the length of a minimal time period for feeding before looking for alternative feeders.

Capability: We would prefer to choose feeders with lower bandwidth consumption and low CPU utilization as this gives us higher confidence in their ability to supply the client's network demands. Furthermore, we limit the bandwidth provided by the feeder to its consumer in order to increase our immunity to unexpected bandwidth consuming events at the feeder.

For measuring these parameters, we offer to embed a special agent software that constantly logs the bandwidth consumed by different processes as well as the relative CPU utilization. We can use the same agent to alert consumer peers when an unpredicted resource consuming task was initiated - threatening the stability of the throughput between the feeder and the consumer peer. In this scenario, the consumer will replace the feeder with a different one.

Infrastructure: Network infrastructure aspects that affect the stability of a peer are not under the control of the user. Large delay variation (jitter), excessive packet delays and round trip time reduce the ability of TCP to ensure a stable connection. Packet loss might also contribute to the download rate instability especially between different regions. Our proposed agent software will log the round trip time and jitter parameters of each feeder it connects to. In time, it will construct a list of potential feeders that have the most stable download rate.

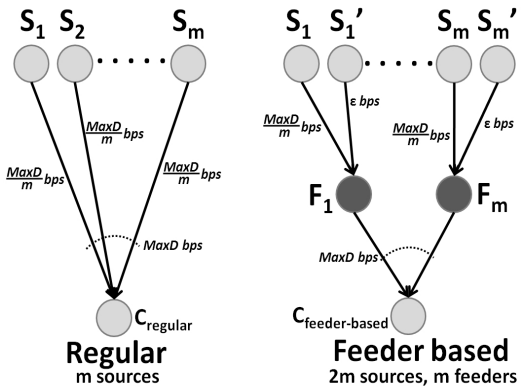

Figure 2. Schematic view of the general case

In addition, as some ISPs employ traffic shapers that [6] downgrade P2P download rate for external connections, by preferring feeder peers that reside in the same ISP as the consumer, it is possible to bypass the traffic policy and increase the throughput.

Analysis:

In Figure 2, $C_{regular}$ represents the case of normal file transfer - downloading from m sources, each supplying $\frac{MaxD}{m} bps$, where $MaxD$ is the maximum download rate of the client peer.

In $C_{feeder-based}$ however, the client downloads a file from m feeders, each of them dowloads from two sources: the 1st source supplies $\frac{MaxD}{m} bps$ and the second source supplies up to ϵ bps. We use the sources that supply ϵ as for short-term caching to ensure that the feeder peer can always supply $\frac{MaxD}{m} bps$ for its client.

In a working system, ϵ will dynamically change during the download process depending on the bandwidth supplied by the source peer. Following is the analysis of the simple model described above, comparing the Effective Download Rate (EDR) of each case, where p_x is the probability that a peer x (source or feeder) will deliver packets at full speed - the capacity of the modem:

$$\begin{aligned} EDR(C_{regular}) = \\ & m \cdot (\text{EUB of each } \textbf{source}) = \\ & m \cdot p_s \cdot (\text{capacity of each source}) = \\ & m \cdot p_s \cdot \frac{MaxD}{m} = p_s \cdot MaxD \ , \end{aligned}$$

where EUB is the effective upload bandwidth.

$$\begin{aligned} EDR(C_{feeder-based}) = \\ & m \cdot (\text{EUB of each } \textbf{feeder}) = \\ & m \cdot (p_f \cdot (\text{UB of feeder depended on its' sources})) = \\ & m \cdot (p_f \min(\frac{MaxD}{m}, \sum \text{EUB (feeder's sources)})) = \\ & \min(p_f MaxD, p_f p_s MaxD + m p_f p_s \epsilon) \ , \end{aligned}$$

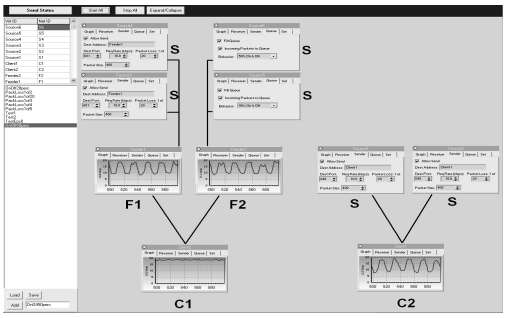

Figure 3. Collabory test suite

where UB is the upload bandwidth and EUB is the effective upload bandwidth. Thus:

$$\frac{EDR(C_{feeder-based})}{EDR(C_{regular})} = \frac{\min\left(p_f MaxD, p_f p_s MaxD + m p_f p_s \epsilon\right)}{p_s MaxD} = \min\left(\frac{p_f}{p_s}, p_f + \frac{m p_f \epsilon}{MaxD}\right),$$

meaning that $C_{feeder-based}$ will download at higher speed than $C_{regular}$ if $p_f + \frac{m p_f \epsilon}{MaxD} > 1 \Rightarrow \epsilon > \frac{(1-p_f)MaxD}{p_f m}$. Notice that as m grows, a smaller ϵ will satisfy the benefit of the feeder-based solution. Likewise, if we allow a bigger ϵ, we can use less feeders to gain the same results. This shows a great benefit of the feeder-based model over the regular model as it is possible to move the "risk" of a non-stable download bandwidth from the client to the feeder - that has potentially much more available download bandwidth than the client. Upon selecting stable feeders it is possible to reach better download stability while using even less stable sources, since the feeder has available download bandwidth that can be used for short-term caching - meaning that we use a bigger ϵ to make sure that the feeder will be able to supply the requested bandwidth to the supplier. The asymmetric upload and download bandwidth does not affect our solution, since a feeder can theoretically download at full download speed to ensure the small upload bandwidth that it should supply the source.

Since we can adjust ϵ dynamically during the download phase, we can afford using extremely weak and unstable sources from the P2P network and still not influence the stability of the download rate at the client, as long as the feeder manages to gather enough cache to be able to provide the requested rate by the consumer. Since it's possible to employ weak sources we estimate that Collabory enhances existing networks' scalability as it increases the total number of potential sources because nowadays existing P2P applications tend to neglect weak sources.

For reinforcing feeders' redundancy, at the beginning of the download process we initiate an extra feeder that downloads packets from the end of the file and does not deliver them to the consumer until either the consumer already downloaded the rest of the file or in case a feeder has failed.

5. Results

Test Suite:
For conducting our tests, we built a special test suite that allows defining structures of connections between 3 types of nodes: source peers, feeders and clients. The suite is composed of a monitoring utility and a set of agents that can emulate the behavior of any of the 3 types of nodes. The agents may run in different machines and submit their performance metrics to the monitoring utility.

As can be seen in Figure 3 the test suite is configured to compare the feeder based model with the regular one.The client/consumer peer $C1$ is connected to 2 feeders $F1$ and $F2$ and each feeder is connected to 2 sources, marked with S. The client $C2$ represents the regular case thus it is only connected to 2 sources. The graphs attached to each feeder and client peers shows the download rate measured at that peer. In this specific experiment we examined burst scenarios i.e., we set all source peers to behave in a repeating pattern of sending at 50% of their maximal upload bandwidth for 10 seconds followed by additional 10 seconds of sending at full speed. It can be noticed that the download rate at the feeders and at $C2$ behave the same, due to using sources with the same behavior. However the client that downloads from the feeders $C1$ gained a stable and higher download rate.

Measurements:
We conducted our tests in two different settings. In the first one we set the maximum download throughput of peers to 920Kb/Sec - common to broadband. Upload is limited to 460Kb/Sec. Packet loss is set to 1 to 20 for all source peers. By employing the behavior of 50% mentioned above we received 866Kb/Sec at $C1$ (feeder based) and 648Kb/Sec at $C2$ (regular). Notice Figure 4 for the results of 80%, 50% and 0% (full transmission for 10 seconds on and off) where the extreme behavioral settings of the sources had only slight effect on the download rate of $C1$ compared to a major decrease in $C2$. The rest of the results are shown for the setting where we set the maximum download throughput of all peers to 20Kb/Sec and the upload is bounded by 10Kb/Sec. This was chosen to show the benefit of Collabory on extremely weak peers that are hardly being used in existing networks because of their unstable nature and low bandwidth.

In Figure 5 we examine different values of ϵ to see how it affects the performance of feeders. All sources in the system are set to the 80% settings of the previous test, including the sources that transmit ϵ which means that they will repeatedly transmit 0.8 ϵKb/Sec for 10 seconds and then ϵKb/Sec for the following 10 seconds. Given larger values of ϵ allows the feeders to hold a cache for a longer period of

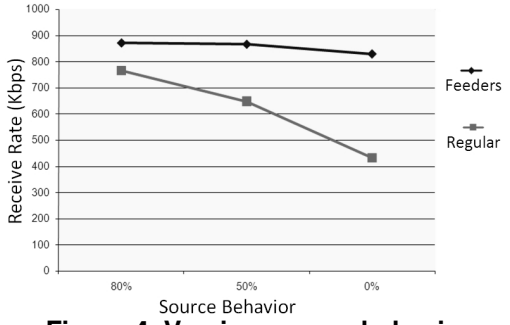

Figure 4. Varying source behavior

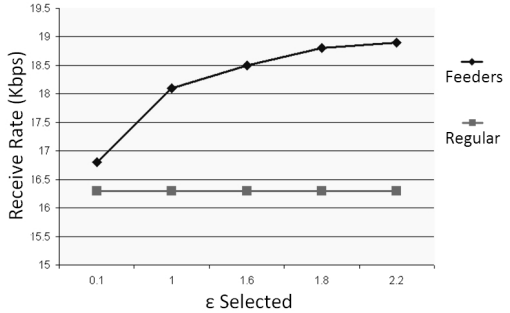

Figure 5. Different ϵ values

time and this way be able to transmit the cache content to $C1$ accordingly. Notice that when we set ϵ to 2.2 the cache content was increasing consistently thus allows the feeder to transmit $C1$ as if it was a stable source. In this scenario $C1$ received stable download rate of 18.9Kb/Sec.

We also tested the case of using weak source peers for the feeder (Figure 6). For the regular method we set 2 sources of 10Kb/Sec with the behavior of 80% mentioned above. For the feeder method we set the following different test settings- A: 4 sources of 6.0Kb/Sec under 80% behavior. B: 8 sources of 3.0Kb/Sec under 80% behavior. C: 8 sources of 4.0Kb/Sec under 50% behavior. In all of our tests we gained stable increased rate in the feeder case compared to unstable rate in the regular case.

eMule/eDonkey Hooking Experiment:

We implemented our solution on a set of 3 machines connected to ADSL broadband connection through different ISPs using 1.5 Mbps ADSL links. Two computers served as feeders for the third machine. We managed to hook into eMule by providing it a set of file portions and generating on-demand maps of the portions we wanted to be downloaded (*.part.met* files), allowing us to "order" any piece of file we wanted at any period of time. Each feeder managed multiple copies of eMule, and opened multiple requests for specific file portions on demand.

We let the experiment run for 10 periods of 4 hours at different parts of the day covering peak hours as well. We constantly logged the download rate of the client. Every 30 minutes, we switched the experiment between Collabory and the normal eMule client. As shown in Figure 7, We measured an average of 1.41 Mbps for the Collabory supported client and an average of 0.92 Mbps for the normal eMule client.

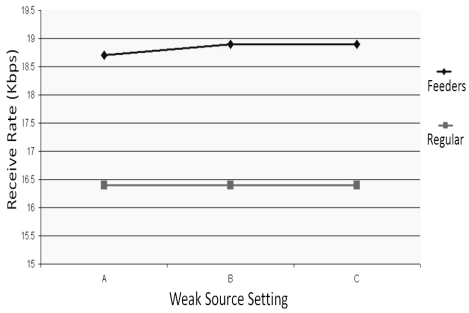

Figure 6. Different settings of weak sources

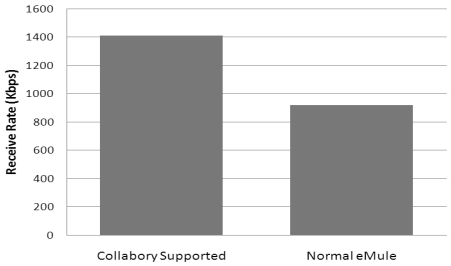

Figure 7. Collabory on eMule

References

[1] Emule web site. http://www.emule-project.net.

[2] A list of isps that use traffic shapers for limiting azureus p2p client. http://www.azureuswiki.com/index.php/bad_isps.

[3] B. Cohen. Incentives Build Robustness in BitTorrent. *Workshop on Economics of Peer-to-Peer Systems*, 6, 2003.

[4] P. Garbacki, A. Iosup, D. Epema, and M. van Steen. 2fast: Collaborative downloads in p2p networks. *p2p*, 2006.

[5] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang. Measurements, Analysis, and Modeling of BitTorrent-like Systems. *Internet Measurement Conference*, 2005.

[6] A. Halme. Peer-to-peer Traffic: Impact on ISPs and Evaluation of Traffic Management Tools.

[7] J. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. Epema, M. Reinders, M. van Steen, and H. Sips. Tribler: A social-based peer-to-peer system. *IPTPS06*.

[8] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329, 2001.

[9] D. Schlosser, T. Hossfeld, and K. Tutschku. Comparison of Robust Cooperation Strategies for P2P Content Distribution Networks with Multiple Source Download. *Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing*, pages 31–38, 2006.

[10] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[11] J. Wang, C. Yeo, V. Prabhakaran, and K. Ramchandran. On the role of helpers in peer-to-peer file download systems: Design, analysis and simulation. *IPTPS07*.

[12] J. Wong. Enhancing collaborative content delivery with helpers. Master's thesis, Univ of British Columbia, 2004.