Performance Analysis of Peer-to-Peer Networks for File Distribution

Ernst W. Biersack[1], Pablo Rodriguez[2], and Pascal Felber[1]

[1] Institut EURECOM, France
{erbi,felber}@eurecom.fr
[2] Microsoft Research, UK
pablo@microsoft.com

Abstract. Peer-to-peer networks have been commonly used for tasks such as file sharing or file distribution. We study a class of cooperative file distribution systems where a file is broken up into many chunks that can be downloaded independently. The different peers cooperate by mutually exchanging the different chunks of the file, each peer being client and server at the same time. While such systems are already in widespread use, little is known about their performance and scaling behavior. We develop analytic models that provide insights into how long it takes to deliver a file to N clients. Our results indicate that the service capacity of these systems grows exponentially with the number of chunks a file consists of.

1 Introduction

Peer-to-peer systems, in which peer computers form a cooperative network and share their resources (storage, CPU, bandwidth), have attracted a lot of interest lately. They provide a great potential for building cooperative networks that are self-organizing, efficient, and scalable.

Research in peer-to-peer networks has so far mainly focused on content storage and lookup, but fewer efforts have been spent on content distribution. By capitalizing on the *bandwidth* of peer nodes, cooperative architectures offer great potential for addressing some of the most challenging issues of today's Internet: the cost-effective distribution of bandwidth-intensive content to thousands of simultaneous users both Internet-wide and in private networks.

Cooperative content distribution networks are inherently *self-scalable*, in that the bandwidth capacity of the system increases as more peers arrive: each new peer requests service from, but also provides service to, the other peers. The network can thus spontaneously adapt to the demand by taking advantage of the resources provided by every peer.

We present a deterministic analysis that provides insights into how different approaches for distributing a file to a large number of clients compare. We consider the simple case of N peers that arrive simultaneously and request to download the same file. Initially, the file exists in a single copy stored at a node called *source* or *server*. We assume that the file is broken up into *chunks* and that peers cooperate, i.e., a peer that has completely received a chunk will offer to upload this chunk to other peers. The time it takes to download the file to all peers will depend on *how* the chunks are exchanged among the peers, which is referred to as peer organization strategy.

To get some insights into the performance of different peer organization strategies, we analytically study three different distribution models:

- A linear chain architecture, referred to as *Linear*, where the peers are organized in a chain with the server uploading the chunks to peer P_1, which in turn uploads the chunks to P_2 and so on.
- A tree architecture, referred to as $Tree^k$, where the peers are organized in a tree with an outdegree k. All the peers that are not leaves in the tree will upload the chunks to k peers.
- A forest of trees consisting of k different trees, referred to as $PTree^k$, which partitions the file into k parts and constructs k spanning trees to distribute the k parts to all peers.

We analyze the performance of these three architectures and derive an upper bound on the number of peers served within an interval of time t. We consider a scenario where each peer has equal upload and download rates of b. The upload rate of the server is also b. We focus on the distribution of a single file that is partitioned into C chunks. The time needed to download the complete file at rate b is referred to as *one round* or 1 unit of time. Thus, the time needed to download a single chunk at rate b is $1/C$. For the sake of simplicity, we completely ignore the bandwidth fluctuation in the network or node failures. We assume that the only constraint is the upload/download capacity of peers.

Several systems have been recently proposed to leverage peer-to-peer architectures for application-layer multicast. Most of these systems target streaming media (e.g., [1–4]) but some also consider bulk file transfers (e.g., [5, 6]). Experimental evaluation and measurements have been conducted on real-world several peer-to-peer systems to observe their properties and behavior [7–10] but, to the best of our knowledge, there has been scarcely any analytical study of distribution architectures for file distribution. We are only aware of one other paper that evaluates the performance and scalability of peer-to-peer systems by modeling the propagation of the file as a branching process [11]. However, no particular distribution architecture is assumed. The results of this paper indicate that the number of clients that complete the download grows exponentially in time and are in accordance with our results.

The rest of the paper is organized as follows. Section 2 introduces the *Linear* architecture. In Section 3 we study $Tree^k$ and we evaluate $PTree^k$ in Section 4. We then presents a comparative analysis of the three distribution models in Section 5 and conclude the paper in Section 6.

2 *Linear*: A Linear Chain Architecture

In this section, we study the evolution over time of the number of served peers for the *Linear* architecture. We make the following assumptions:

- The server serves sequentially and infinitely the file at rate b. At any point in time, the server uploads the file to a single peer.
- Each peer starts serving the file once it receives the first chunk.

We consider the case where each peer uploads the whole file at rate b to exactly one other peer before it disconnects. Thus, each peer contributes the same amount of data to the system as it receives from the system. At time 0, the server starts serving a first peer. At time $1/C$, the first peer has completely received the first chunk and starts serving a second peer. Likewise, once the second peer has received the first chunk at time $2/C$, it starts serving a third peer and so on. As a result, peers are connected in a chain with each peer receiving chunks from the previous one and serving the next one. The length (i.e., the number of peers) of the chain increases by one peer each $1/C$ unit of time. At time 1, the server finishes uploading the file to the first peer. If there are still peers left that have not even received a single chunk, the server starts a new chain that increases also by one peer each $1/C$ unit of time. The same process repeats at each round, as shown in Figure 1 (the black circle represents the server, the black squares are peers that start downloading the file, and the lines connecting the peers correspond to active connections). This makes $(t+1)$ chains within t rounds. The number of served peers at time t over all those chains includes only the peers that have joined the network on or before time $t-1$. Clients that arrive after time $t-1$ will take one unit of time to download the file and will be done after time t. Given a chain initiated at time 0, its length at time t is $(1+t\cdot C)$ and the number of served peers in that chain is $1+(t-1)C$ peers. Over all chains, the number of served peers within t rounds is given by

$$N_{Linear}(C,t)=\sum_{i=1}^{t}(1+(i-1)C)=t+\frac{C\cdot t(t-1)}{2}\approx C\cdot t^2 \qquad (1)$$

We see that the number of peers served grows linearly with the number of chunks C and quadratically with the number of rounds t. From Equation (1) we derive the formula for the time needed to completely serve N peers as

$$T_{Linear}(C,N)=\frac{(C-2)+\sqrt{(C-2)^2+8\cdot N\cdot C}}{2\cdot C}\approx\frac{1}{2}+\sqrt{\frac{1}{4}+\frac{2\cdot N}{C}} \qquad (2)$$

If N/C denotes the node to chunk ratio, we can distinguish the following cases:

1. $T_{Linear}(C,N)\approx\frac{1}{2}+\sqrt{\frac{1}{4}}=1$, for $\frac{N}{C}\ll 1$
2. $T_{Linear}(C,N)\approx\frac{1}{2}+\sqrt{2}\approx 2$, for $\frac{N}{C}=1$
3. $T_{Linear}(C,N)\approx\sqrt{\frac{N}{C}}$, for $\frac{N}{C}\gg 1$

Figure 2 plots $N_{Linear}(C,t)$ as a function of the number of rounds for different values of the number of chunks C. It appears clearly that, for a given number of peers N, the smaller the node to chunk ratio N/C, the shorter the time to serve

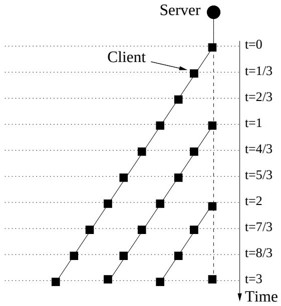


Fig. 1. Evolution of the *Linear* architecture with time ($C = 3$).

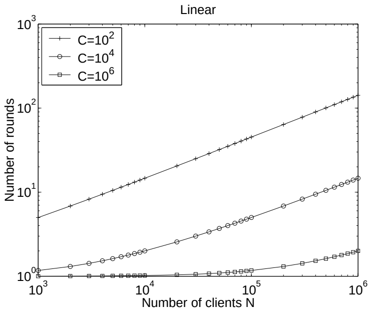


Fig. 2. $T_{Linear}(C, N)$ as a function of N and C.

these N peers. In fact, for $N/C \ll 1$ all peers will be active uploading chunks for most of the time and T_{Linear} will be approximately one round. On the other hand, for $N/C > 1$ only C out of the N peers will be uploading at any point in time, while the other $N - C$ peers have either already forwarded the entire file or not yet received a single chunk.

3 *Tree*^{k}: A Tree Distribution Architecture

As we have just seen, for $N/C > 1$ the linear chain fails to keep all the peers working most of the time. To alleviate this problem we now consider *Tree*^{k}, a tree architecture with outdegree k where the number of "hops" from the server to the last peer is $\log_k N$, as compared to N for the linear chain. We make the following assumptions:

- The server serves k peers in parallel, each at rate b/k.
- Each peer downloads the whole file at rate b/k.
- A peer that is interior (i.e., non leaf) node of the distribution tree starts uploading the file to k other peers, each rate b/k, soon as it has received the first chunk. This means that interior nodes upload an amount of data equivalent to k times the size of the file, while leaf nodes do not upload the file at all.

Given a download rate of b/k, a peer needs k/C units of time to receive a single chunk. Note that *Tree*$^{k=1}$ is equivalent to *Linear*. We first explain the evolution for $k = 2$. At time 0, the server serves 2 peers each at rate $b/2$. Each of those peers starts serving 2 new peers $2/C$ units later, which will need to wait another $2/C$ units before they have completely received a chunk and can in turn serve other peers. The two peers served by the server become each a root of a tree with an outdegree 2. The height of each tree increases by one level every $2/C$ units of time (see Figure 3).

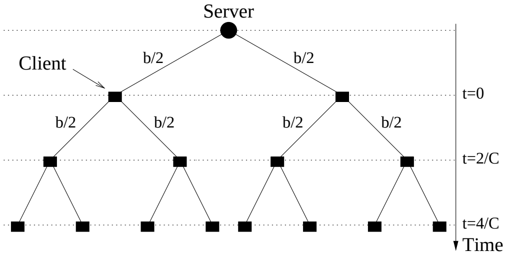


Fig. 3. Evolution of the *Tree*$^{k=2}$ architecture with time.

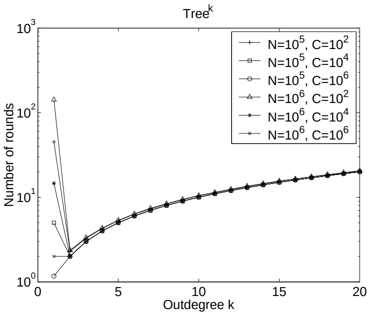


Fig. 4. $T_{Tree}(C, k, N)$ as a function of k, N, and C.

In the Tree^k architecture, k identical trees are initiated by the server at time 0, each of which will include N/k peers. The time needed to serve N peers is

$$T_{Tree}(C, k, N) = k + \lfloor \log_k(\frac{N}{k}) \rfloor \cdot \frac{k}{C} \tag{3}$$

where k/C represents the delay induced by each level in the tree. Leaf peers start receiving the first chunk $\lfloor \log_k(N/k) \rfloor \cdot k/C$ units of time after the root peer and complete the download k units of time later.

We derive from Equation (3) the number of peers served within t rounds as

$$N_{Tree}(C, k, t) \approx k^{(\frac{t}{k}-1)C+1} = k^{(t-k)\frac{C}{k}+1} \tag{4}$$

It follows from Equation (3) and (4) that the performance of file distribution directly depends on the degree k of the tree. We can compute the optimal value of k by taking the derivative of T_{tree} with respect to k. This gives

$$k_{opt} = e^{\frac{-\log N + \sqrt{(\log N)^2 + 4\cdot(C-1)\cdot \log N}}{2\cdot(C-1)}} \qquad \text{given that} \quad N < k \cdot \frac{k^{(C+1)} - 1}{k - 1} \tag{5}$$

The optimal outdegree k_{opt} depends on the peer to chunk ratio N/C. For $N/C \leq 1$, the optimal outdegree is 1, i.e., a linear chain, since the linear chain assures that the peers are uploading most of the time at their full bandwidth capacity. For $N/C > 1$, an increase in N/C leads to an increase in the optimal outdegree as the linear chain becomes less and less effective (remember that only C out of the N peers are uploading simultaneously).

In practice, the outdegree can only take integer values and we see from Figure 4 that for $N/C > 1$ the binary tree yields lower download times that the linear chain. The binary tree is also the optimal tree. Remember that in T_{tree} the outdegree k appears as an additive constant that is typically much larger than the other term ($\lfloor \log_k(N/k) \rfloor \cdot k/C$)

While the binary tree improves the download time as compared to the linear chain when $N/C > 1$, it suffers from two important shortcomings. First, although the maximum upload and download rate is b, the peers in a binary tree download only at rate $b/2$. As a consequence, the download time is at least *twice* the time it takes if the file were downloaded at the maximum possible download rate.

Second, in a binary tree of height h, there are 2^h leaf nodes and $2^h - 1$ interior nodes. Since only the interior nodes upload chunks to other peers, this means that half of the peers will not upload even a single block. As the outdegree k of the tree increases, the percentage of peers uploading data keeps decreasing, with only about one out of k peers uploading data. Also, the peers that upload must upload the entire file k times.

4 *PTree^k*: An Architecture Based on Parallel Trees

The overall performance of the tree architecture would be significantly improved if we could capitalize on the unused upload capacity of the leaves to utilize the $b - b/k$ unused download capacity at each of the peers. It is not possible, however, for a leaf to serve other peers upwards its tree because it only holds chunks that its ancestors already have. Given a tree architecture with k trees rooted at the server, the basic intuition underlying the *PTree^k* architecture is to "connect" the leaves of one of the trees to peers of the other $k - 1$ trees to ultimately produce k spanning trees, and have the server send distinct chunks to each of these trees.

More specifically, the *PTree^k* architecture organizes the peers in k different trees such that each peer is an interior peer in at most one tree and a leaf peer in the remaining $k - 1$ trees. The file is then partitioned into k parts, where each part is distributed on a different tree: tree T^k for part P^k. All k parts have the same size in terms of number of bytes. If the entire file is divided into C chunks, each of the k parts will comprise C/k disjoint chunks.[3] Such a distribution architecture was first proposed under the name of SplitStream [3] to increase the resilience against churn (i.e., peers failing or leaving prematurely) in a video streaming application.

In *PTree^k*, a peer receives the k parts in parallel from k different peers, each part at rate b/k, while the peer helps distributing at most one part of the file to k other peers. Therefore, the total amount of data a peer uploads corresponds exactly to the amount contained in the file, regardless the outdegree k of the trees.

Figure 5 depicts the basic idea of *PTree$^{k=2}$*, where k denotes the outdegree of the tree. Each peer, except for peer 4, is an interior peer in one tree and a leaf peer in another tree. It is easy to show that, independent of the outdegree k, there will always be one peer in *PTree^k* that is leaf in all k trees. Each tree includes all N peers. A tree with outdegree k has $1 + \lfloor log_k N \rfloor$ levels and a height of $\lfloor log_k N \rfloor$. Since peers transmit at a rate b/k, each level in the tree induces a delay of k/C units of time.

Consider a leaf peer C_0 in tree T^k that is located $\lfloor log_k N \rfloor$ levels down from the root of T^k. C_0 starts receiving part P^k at time $\lfloor log_k N \rfloor \cdot k/C$ and the time

[3] For the sake of simplicity, we assume that the number of chunks C is a multiple of the number of parts k.

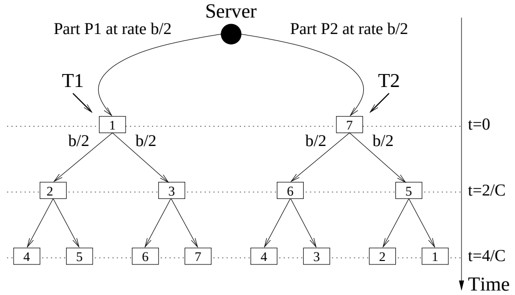


Fig. 5. Evolution of the $PTree^{k=2}$ architecture with time.

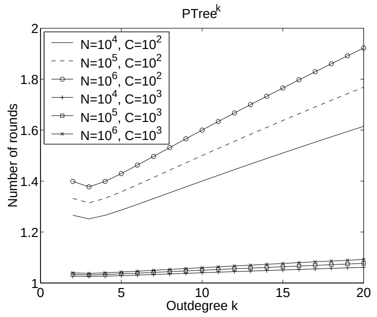


Fig. 6. $T_{PTree}(C, k, N)$ as a function of k, N, and C.

to receive P^k entirely, once reception has started, is 1. Therefore, C_0 will have completely received part P^k at time $1 + \lfloor log_k N \rfloor \cdot k/C$.

A peer has completed its download after it has received the last byte of each of the k parts of the file. A $PTree^k$ peer is a leaf node in $k-1$ trees and an interior node in one tree, and it receives all k parts in parallel. This means that all peers complete their download at the same time $1 + \lfloor log_k N \rfloor \cdot k/C$. We therefore have

$$T_{PTree}(C, k, N) = 1 + \lfloor log_k N \rfloor \cdot \frac{k}{C} \tag{6}$$

We derive from Equation (6) the number of peers served within t rounds as

$$N_{PTree}(C, k, t) \approx k^{(t-1)\frac{C}{k}} \tag{7}$$

Similarly to the tree distribution architecture in Section 3, there is an optimal value k for $PTree^k$ that minimizes the service time. Intuitively, a very deep tree should be quite inefficient in engaging peers early since leaves are quite far from the source. In fact, $PTree^{k=1}$ is equivalent to $Linear$, which is very inefficient in engaging peers for $N/C > 1$. On the other hand, when the outdegree of the tree is large, leaf peers are only a few hops from the source and can be engaged fast. However, this intuition is not completely correct: flat trees with large outdegrees suffer from the problem that, as the outdegree k increases, the rate b/k at which each chunk is transmitted from one level to the next one decreases linearly with k. This rate reduction can negate the benefits of having many peers reachable within few hops.

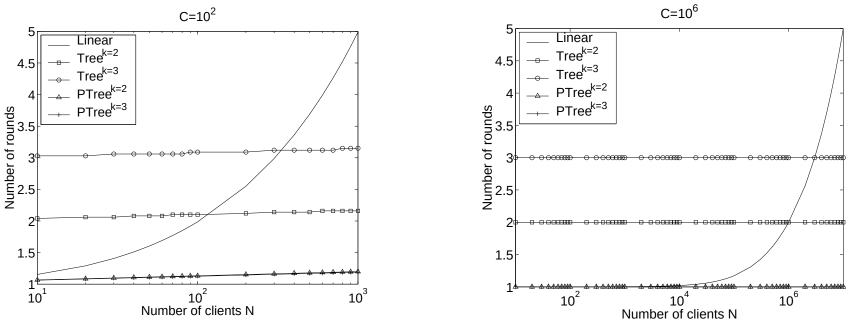
We can compute the optimal tree outdegree that provides the best $PTree^k$ performance by taking the derivative of Equation (6) with respect to k and equating the result to zero. We find T_{PTree} to be minimal for $k = e$, independant of the peer to chunk ratio N/C.

Figure 6 depicts the performance of $PTree^k$ as a function of the outdegree. We see that the optimal $PTree^k$ performance is obtained for trees with an outdegree $k = 3$. However, the performance for $k = 2$ and $k = 4$ is almost the same as for $k = 3$. As the outdegree increases the performance of PTree degrades: for $N/C \approx 1$ the degradation is very small while for $N/C \gg 1$ it is quite pronounced.

By striping content across multiple trees, $PTree^k$ can ensure that the departure of one peer causes only a minimal disruption to the system, reducing the peer's throughput only by b/k. Given that the overhead caused by churn can be minimized by striping content across a higher number of trees, one can consider slightly higher outdegrees than the optimal value (e.g., 5) to minimize the impact of churn at the expense of a minimal increase in transfer time.

5 Comparative Analysis

In this section, we compare the performance of the *Linear*, $Tree^k$ and $PTree^k$ architecture. We first investigate how the time needed to serve N peers varies as a function of the number of peers N and the number of chunks C.



(a) $C = 10^2$

(b) $C = 10^6$

Fig. 7. Performance of *Linear*, $Tree^{k=\{2,3\}}$ and $PTree^{k=\{2,3\}}$ as a function of N.

From Figure 7, we see that independent of the number of nodes and chunks, $PTree^k$ is able to offer download times close to 1. On the other hand, as already pointed out, the download times for $Tree^k$ are always larger than k units of time (see Equation (3)). $PTree^k$ with optimal outdegree $k = 3$ provides clear benefits over *Linear* for $N/C \gg 1$ since peers are engaged much faster into uploading chunks than with a linear chain.

When the propagation delay of the first chunk is very small compared to the transmission time of the file, the peers stay engaged most of the time in the linear chain and the benefit of $PTree^k$ diminishes. This is the case when the number of chunks is very large ($C \rightarrow \infty$), the number of peers is small, or the transmission rate is very high. The pivotal point where $PTree^k$ starts to significantly outperform *Linear* is around $N/C > 10^{-1}$ (see Figure 7(b)).

The comparison of the different approaches would be incomplete if we did not address aspects such as robustness and ease of deployment. Cooperative file distribution relies on the collaboration of individual computers that are not subject to any centralized control or administration. Therefore, the failure or departure of some computers during the data exchange are most likely to occur and should also be taken into account when comparing approaches. For the linear chain and the tree, the departure of the node will disconnect all the nodes "downstream" from the data forwarding. With $PTree^k$, the impact of the departure of a node effects only affects one out of k trees, which makes parallel trees the most robust of the three approaches.

The *Linear* and $PTree^k$ architectures both assume that the upload and download rates are the same. In practice, this is often not the case (e.g., the upload rate of ADSL lines is only a fraction of the download rate). In such cases, the performance will be limited by the upload rate and some of the download bandwidth capacity will remain unused. The $Tree^k$ architecture assumes that nodes can *upload* data at a rate k times higher than the download rate, which is exactly the opposite to what ADSL offers. The tree approach is therefore particularly ill-suited for such environments.

From these results we can conclude that, for most typical file-transfer scenarios, $PTree^k$ provides significant benefits over the other two architectures studied in this paper.

6 Conclusions and Perspectives

The *self-scaling* and *self-organizing* properties of peer-to-peer networks offer the technical capabilities to quickly and efficiently distribute large or critical content to huge populations of clients. Cooperative distribution techniques capitalize on the bandwidth of every peer to offer a service capacity that grows exponentially, provided the blocks among the peers are exchanged in such a way that the peers are busy most of the time. The architecture that best achieves this goal among those studied in the paper, independently of the peer to chunk ratio N/C, is $PTree^k$. For both $Tree^k$ and $PTree^k$ there is an optimal outdegree that minimizes the download time.

Our analysis provided some important insights as to how to choose certain key parameters such as C and k. First, the file should be partitioned into a large number of chunks C, since the performance scales exponentially with C (but not too many as each chunk adds some coordination and connection overhead). Second, each peer should limit the number k of simultaneous uploads to other peers. We saw that for $PTree^k$ a good value for k is between 3 and 5.

The results of our study also guide the design of cooperative peer-to-peer file sharing applications that do not organize the peers in a such a static way as do the linear chain or tree(s) but use a *mesh* instead (e.g., BitTorrent [6]). Here, a peer must decide how many peers to serve simultaneously (the outdegree k) and what chunks to serve next (the "chunk selection strategy"). For each chunk, a peer selects the peer it wants to upload that chunk to (the "peer selection strategy").

Consider a peer selection strategy that gives preference to the peers that are closest to completion (among those that have the fewest incoming connections), and a chunk selection strategy that favors the chunks that are least widely held in the system. Assume that each peer only accepts a single inbound connection. With 1 outbound connection per peer, we trivially obtain a linear chain; with 2 outbound connections, we obtain a binary tree $\mathit{Tree}^{k=2}$; and so on. Failures are a handled gracefully as the parent of a failed peer automatically reconnects to the next peer in the chain or tree.

If we now allow each peer to have k inbound and k outbound connections, we obtain a configuration equivalent to $P\mathit{Tree}^k$. Indeed, the source will fork k trees to which it will send distinct chunks (remember that we give preference to the rarest chunks). The leaves of the trees, which have free outbound capacity, will connect to the peers of the other trees to eventually create k parallel spanning trees. Such mesh-based systems, whose topology dynamically evolves according to predefined peer and chunk selection strategies, offer service times as low as the ones of $P\mathit{Tree}^k$ and adjust dynamically to bandwidth fluctuations, bandwidth heterogeneity, and node failures [12].

References

1. Chu, Y.H., Rao, S., Zhang, H.: A case for end system multicast. In: Proceedings of ACM SIGMETRICS. (2000)
2. Jannotti, J., Gifford, D., Johnson, K.: Overcast: Reliable multicasting with an overlay network. In: Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI). (2000)
3. Castro, M., Druschel, P., Kermarrec, A.M., Nandi, A., Rowstron, A., Singh, A.: SplitStream: High-bandwidth multicast in a cooperative environment. In: Proceedings of the ACM Symposium on Operating Systems Principles (SOSP). (2003)
4. Tran, D., Hua, K., Do, T.: ZIGZAG: An efficient peer-to-peer scheme for media streaming. In: Proceedings of IEEE INFOCOM, San Francisco, CA, USA (2003)
5. Sherwood, R., Braud, R., Bhattacharjee, B.: Slurpie: A cooperative bulk data transfer protocol. In: Proceedings of IEEE INFOCOM. (2004)
6. Cohen, B.: Incentives to build robustness in bittorrent. Technical report, http://bitconjurer.org/BitTorrent/bittorrentecon.pdf (2003)
7. Saroiu, S., Gummadi, K., Gribble, S.: A measurement study of peer-to-peer file sharing systems. In: Proceedings of Multimedia Computing and Networking (MMCN). (2002)
8. Ripeanu, M., Foster, I., Iamnitchi, A.: Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. IEEE Internet Computing Journal **6** (2002)
9. Gummadi, K., Dunn, R., Saroiu, S., Gribble, S., Levy, H., Zahorjan, J.: Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In: Proceedings of the ACM Symposium on Operating Systems Principles (SOSP). (2003)
10. Izal, M., Urvoy-Keller, G., Biersack, E., Felber, P., Hamra, A.A., Garces-Erice, L.: Dissecting BitTorrent: Five months in a torrent's lifetime. In: Proceedings of the 5th Passive and Active Measurement Workshop (PAM). (2004)
11. Yang, X., de Veciana, G.: Service capacity of peer-to-peer networks. In: Proceedings of IEEE INFOCOM. (2004)
12. Felber, P., Biersack, E.: Self-scaling networks for content distribution. In: Proceedings of the International Workshop on Self-* Properties in Complex Information Systems. (2004)