Overlay Architectures for File Distribution: Fundamental Performance Analysis for Homogeneous and Heterogeneous Cases *

E. W. Biersack[1], D. Carra[2], R. Lo Cigno[2], P. Rodriguez[3] , P. Felber[4†]

[1]Institut EURECOM, Sophia Antipolis, FR

[2]Università di Trento, IT

[3]Microsoft Research, Cambridge, UK

[4]Université de Neuchâtel, CH

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 given a distribution architecture. Our results indicate that even for the case of heterogeneous client populations it is possible to achieve download times that is almost independent of the number of clients and very close to optimal.

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; 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 simultaneous distribution

*This work has been partly supported by the European Union under the E-NEXT project FP6-506869.

†This work was done while P. Felber was with Institut EURECOM

of bandwidth-intensive content to thousands of users both Internet-wide and in private networks.

Cooperative content distribution networks are inherently *self-scalable*, in that the overall 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 impact on performance. We consider the simple case of N peers that simultaneously 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, or distribution architecture.

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 both the homogeneous case, where all peers have the same bandwidth, and the heterogeneous case, where peers are divided in classes based on their access bandwidth. In the heterogeneous case we consider different cooperation schemes between the classes in order to understand how heterogeneity affects the performance of the distribution scheme. 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.

The model presented in this paper represents a high level description of different protocols that uses similar mechanisms to distribute the content. The file to be distributed, in fact, is broken into independent chunks and each chunk can be distributed individually. This methodology is a basic functionality of BitTorrent [1], so, leaving out details such as chunk selection strategies and peer selection strategies, our model is able to catch performance bounds that can be obtained using such protocols.

Moreover, we suppose to have collaborative peers, i.e., each peer, if necessary, help distributing the chunks. This principle is also known as swarming or hoarding: chunks reach all the peers through the other peers, using their resources. The different distribution architectures define different organizations of the paths that chunks can follow during the distribution process, and the topological properties of the distribution overlay network. Since we have complete knowledge of the network, and the peer bandwidths are stable, these paths can be considered deterministic, i.e., giving simple content distribution policies that are run locally by each peer, the content follows paths that can be calculated (giving the desired performance metrics we are interested in).

In this paper we disregard protocol details, focusing on general properties of the distribution organization. For instance, Avalanche [2] and BitTorrent differ in techniques for chunk selection and encodings. We do not consider such properties, while both Avalanche and BitTorrent can, with some modifications, be used to provide file distribution following the general organization schemes studied in this paper.

1.1 Related Work

The early work by Saroiu et al. [3] analyzes Gnutella and Napster traces; the aim of the study is to characterize end-user hosts, their connectivity and behavior. The results of the analysis show the presence of significant heterogeneity in peer capacity, availability and behavior; moreover, there is a fraction of peers that act primarily as clients (i.e., they only download) and other primarily as servers (i.e., they are altruists).

In [4] authors analyze the trace of BitTorrent P2P application [1]; in this case the aim is to assess the performance of the algorithm used in BitTorrent. They conclude that mechanisms used by BitTorrent allow efficient and rapid replication of contents, even in presence of flash crowd phenomena.

The work in [5] is among the first to propose an analytical model of a P2P system, evaluating its performance. The paper represents a P2P system as a multi-class closed queuing network and it shows the influence of the design parameters, like peer request rate and file popularity, on stationary performances.

In [6], authors use an age dependent branching process to model the transient evolution of a P2P system and a simple Markovian model to analyze the steady state regime. This paper introduces the concept of service capacity as the number of available copies in the network. 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.

Fluid models have been recently considered given their analytical tractability and their potential to describe dynamic and transient behavior. The work in [7] proposes a fluid model for the analysis of the Squirrel protocol [8]. The result is an accurate model that estimates the performance of the protocol. In [9] the authors study the BitTorrent protocol with a simple fluid model. The model is able to catch the transient and the steady state behavior of the system with a few simple parameters; moreover, an analysis of the different mechanisms of BitTorrent is provided. In [10] a stochastic fluid flow model is proposed to study the file distribution: the model computes the cumulative distribution of file transfer time and evaluates the impact of the system parameters, such as file popularity, bandwidth characteristics, concurrent downloads and uploads, on the performance.

Of the works above only [5] and [10] tackle the problem of presence of different access capacities among peers which is instead one of the focuses of this paper. In contrast to our work, both approaches do not consider file distribution process and do not take into account distribution architectures.

A related topic where distribution architectures are explicitly taken into account is the delivery of streaming services through overlay multicast. Narada [11], ALMI [12], NICE [13], and SplitStream [14] (from which the inspiration for PTree was taken), for instance, define a set of mechanisms to efficiently distribute the content to many overlay nodes. They build in different ways distribution trees and manage the dynamics of leaving and joining nodes. Nevertheless most of these studies are focused on protocol design and do not analyze the impact of distribution architectures on performance. Performance evaluation is only focused on the proposed protocol.

Other studies, [15] and [16], analyze file swarming but do not consider any particular architecture and are focused on other problems, like replication strategies and peer selection. The work in [17] studies how to build the tree topology, but it does not compare different topologies.

This paper extends the analysis made in [18] (partially replicated here) considering the presence of different access bandwidths.

2 Homogeneous Case

2.1 General Assumptions

We consider a scenario where each peer has the same upload and download bandwidth b. The case of asymmetric bandwidths (typically, with download greater than upload as in ADSL) corresponds to the case with symmetric bandwidth equal to the upload bandwidth: in all the presented schemes, in fact, the upload bandwidth is always saturated, so a greater download bandwidth cannot contribute in improving the performance.

The upload bandwidth 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 bandwidth b is referred to as *one round* or 1 unit of time. Thus, the time needed to download a single chunk given a bandwidth b is $1/C$.

We make the following assumptions:

- The server uploads the file indefinitely to one peer at time;
- Each peer starts serving the file once it has completely received the first chunk, and uploads the whole file once to k peers, $k = 1, 2, \ldots$

2.2 *Linear*: A Linear Chain Architecture

In this section, we study the evolution over time of the number of served peers for the *Linear* architecture. At any point in time, the server uploads the file to a single peer. Each peer has a bandwidth b and uploads the whole file to exactly one other peer ($k = 1$) 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. The same process repeats at each round, as shown in Fig. 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 $t-1$. 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. Including all parallel chains, the number of served peers within t rounds is given by

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

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 time needed to serve N peers:

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

N/C is the node to chunk ratio and we 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 $\frac{N}{C}\simeq 1$
3. $T_{Linear}(C,N)\approx\sqrt{\frac{N}{C}}$, for $\frac{N}{C}\gg 1$

Fig. 2 plots $T_{Linear}(C,N)$ as a function of the number of peers for different values of the number of chunks C. As expected, for a given number of peers N, the smaller the node to chunk ratio N/C, the shorter the time to serve all N peers. In fact, for $N/C\ll 1$ all peers are active uploading chunks for most of the time and T_{Linear} is 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.

2.3 *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 approximately $\log_k N$, as compared to N for the linear chain. We make the following assumptions:

- The server uploads to k peers in parallel, each with a bandwidth of b/k;
- Each peer downloads the whole file at bandwidth b/k;

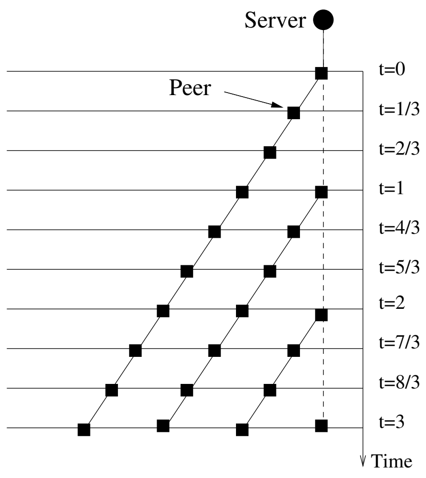


Figure 1: Evolution of the *Linear* architecture with time ($C = 3$).

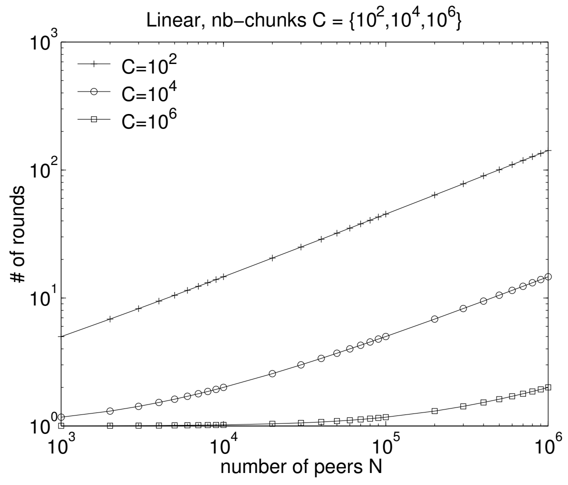


Figure 2: $T_{Linear}(C, N)$ as a function of N and C.

- A peer that is interior (i.e., non leaf) node of the distribution tree starts uploading the file to k other peers, each at a bandwidth b/k, as 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 bandwidth of b/k, a peer needs k/C units of time to receive a single chunk. To compute the download times in a tree architecture we need to know the number of levels l in a tree with N nodes. Without loss of generality, we assume that the tree is full. At the first level, the tree contains k nodes (the level 0 is the server itself), the second k^2 nodes and so on; therefore $N = \sum_{j=1}^{l} k^j = k\frac{k^l-1}{k-1}$ and $l = log_k\left(N\frac{k-1}{k}+1\right)$. The total download time is

$$T_{Tree}(C,k,N) = k + \left[log_k\left(N\frac{k-1}{k}+1\right)-1\right]\cdot\frac{k}{C}\ . \tag{3}$$

The term k/C is the transfer delay of each level in the tree. Leaf peers start receiving the first chunk after $\left(log_k\left(N\frac{k-1}{k}+1\right)-1\right)\cdot\frac{k}{C}$ units of time. They complete the download k units of time later. We subtract 1 to the number of levels because the time to upload the first chunk to the leaves is included in the time to upload the whole file.

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

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

It follows from Equation (3) and (4) that the performance of file distribution directly depends on the degree k of the tree.

The optimal outdegree k_{opt} depends on the peer to chunk ratio N/C (see Fig. 3). For $N/C \leq 1$, the optimal outdegree is 1, i.e., a linear chain, since 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).

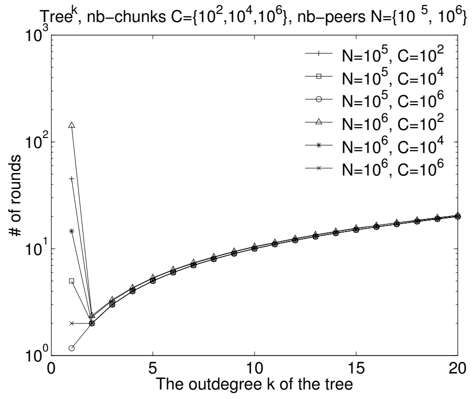


Figure 3: $T_{Tree}(C, k, N)$ as a function of k, N, and C.

In practice, the outdegree can only take integer values and we see that for $N/C > 1$ the binary tree yields lower download times than 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 ($(log_k\left(N\frac{k-1}{k}+1\right)-1)\cdot k/C$)

Trees, however, suffer from two important shortcomings:

1. While the maximum upload and download bandwidth is b, the peers in a tree download only at bandwidth b/k. As a consequence, the download time is at least *k-times* the time it takes if the file were downloaded at the maximum possible download bandwidth;

2. In a tree of outdegree k and height l, there are k^l leaf nodes and $\frac{k^l-k}{k-1}$ interior nodes. Since only the interior nodes upload chunks to other peers, this means that (in a binary tree) more than half of the peers will not upload even a single block. Also, the peers that upload must upload the entire file k times.

2.4 *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 upward 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, as described in [18], 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.[1] Such a distribution architecture was first proposed under the name of SplitStream [14] 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 bandwidth 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.

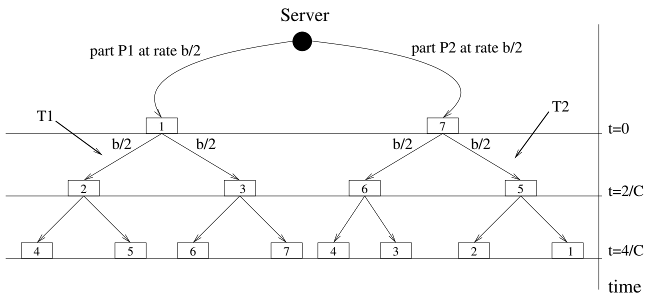


Figure 4: Evolution of the *PTree*$^{k=2}$ architecture with time.

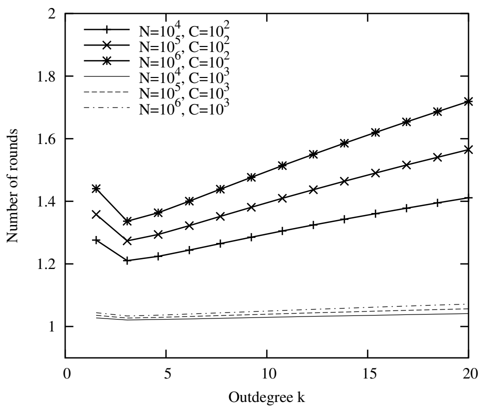


Figure 5: $T_{PTree}(C, k, N)$ as a function of k, N, and C.

Fig. 4 depicts the basic idea of *PTree*$^{k=2}$, where k denotes the outdegree of each 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.

The deterministic analysis of a *PTree*^{k} architecture begins observing that the server starts k different distribution trees and each tree contains all N nodes. The number of levels l in a tree with N nodes[2] can be found considering that the first level contains 1 node, the second k nodes, the third k^2 nodes, and so forth; so $N = \sum_{j=0}^{l} k^j = \frac{k^{l+1}-1}{k-1}$ and $l = log_k\left(\frac{N(k-1)+1}{k}\right)$.

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 $1 + (l-1)\cdot k/C$. We subtract 1 to the number of levels because the time to upload the first chunk to the leaves is included in the time to upload the whole file. The total download time is then

$$T_{PTree}(C, k, N) = 1 + \left[log_k\left(\frac{N(k-1)+1}{k}\right) - 1\right]\cdot\frac{k}{C}\,. \tag{5}$$

We derive from Equation (5) the number of peers served within t rounds as

$$N_{PTree}(C, k, t) \approx \left(k^{(t-1)\frac{C}{k}+1} - \frac{1}{k}\right)\frac{k}{k-1}\,. \tag{6}$$

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

[2]Again, we assume a full tree.

As for the tree architecture in Section 2.3, 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 bandwidth b/k at which each chunk is transmitted from one level to the next one decreases linearly with k. This bandwidth reduction can negate the benefits of having many peers reachable within few hops.

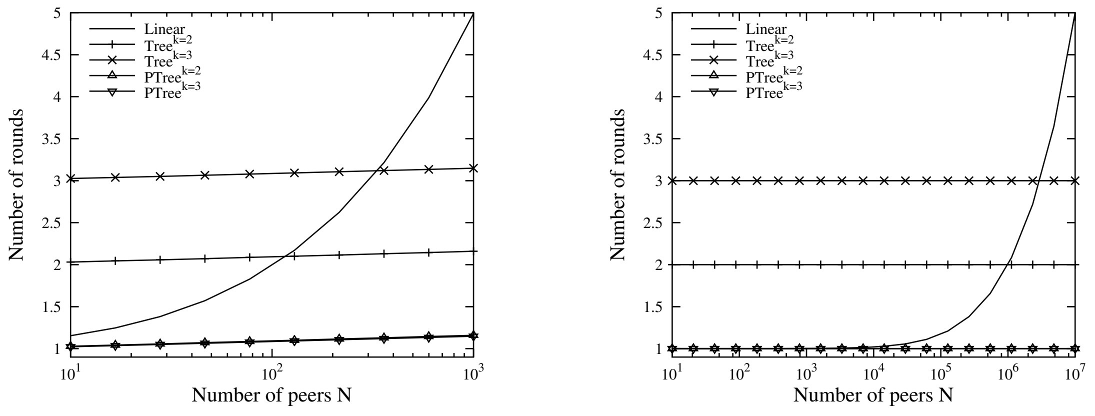
We can compute the optimal tree outdegree that provides the best *PTree*^{k} performance by taking the derivative of Equation (5) with respect to k and equating the result to zero.

Fig. 5 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*^{k} degrades: for $N/C \approx 1$ the degradation is very small while for $N/C \gg 1$ it is quite pronounced. It is interesting to notice that the optimal outdegree k for *PTree*^{k} architectures is different from that of a single *Tree*^{k}.

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 peers' 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.

2.5 Comparative Analysis for the Case of Homogeneous Peers

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) $C = 10^6$

Figure 6: Performance of *Linear*, *Tree*$^{k=\{2,3\}}$ and *PTree*$^{k=\{2,3\}}$ as a function of N.

From Fig. 6, we see that regardless of the number of nodes, chunks and outdegree k, *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)).

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 ($C \to \infty$), the number of peers is small, or the transmission bandwidth is very high. The pivotal point where *PTree*^{k} starts to significantly outperform *Linear* is around $N/C > 10^{-1}$ (see Fig. 6(b)).

3 Heterogeneous Case

3.1 General Assumptions

So far we have assumed that all peers have the same constant upload and download bandwidth. We now study the case where we have two classes of peers, referred to as fast peers and as slow peers. We look at different types of collaborative strategies among the two classes and are interested in how the fast peers can help the slow ones to improve their download time and how this will affect the download time of the fast peers. In addition to those described for the homogeneous case, the main assumptions are as follows:

- There are only two classes of peers in the network: class 1 (fast peers) with upload and download bandwidth b_1 and class 2 (slow peers) with upload and download bandwidth b_2; $b_1 > b_2$. In this case we define *one round* the time it takes to download the complete file at bandwidth b_2;

- The number of peers in class 1 and class 2 is equal to N_1 and N_2 respectively; the total number of peers is $N = N_1 + N_2$;

- The server has sufficient bandwidth to upload concurrently to the two classes, i.e., its upload bandwidth b_S is equal to $b_1 + b_2$; this hypothesis simplifies calculations and does not greatly influence the final results since the impact is only on peers that download from the server[3];

- Peers can be selfish, i.e., they disconnect as soon as they finish downloading the content, or altruistic, i.e., they remain in the system for a certain period of time after finishing the download (the time lapse is related to the specific used policy).

We consider four different variations for each of the architectures that differ in the way the peers cooperate. The most obvious policy is one where the peers in each class behave as if the peers in the other class do not exist. This scheme is referred to as *independent*. In the other three policies the fast peers always help the slow peers while the slow peers either do not upload any data at all or upload data to other slow peers. We analyze in detail the variations on the linear distribution architecture and then extend the results on the *Tree*^{k} and *PTree*^{k} architectures.

[3]We are interested in keeping the number of peers that download from the server as small as possible, otherwise the network departs from a P2P architecture and resembles more a traditional client/server one.

3.2 Linear Architecture and Independent Classes

In this case the server uploads chunks independently to each class; peers belonging to a class do not exchange contents with other class's peers. Fig. 7 shows the chunk distribution methodology.

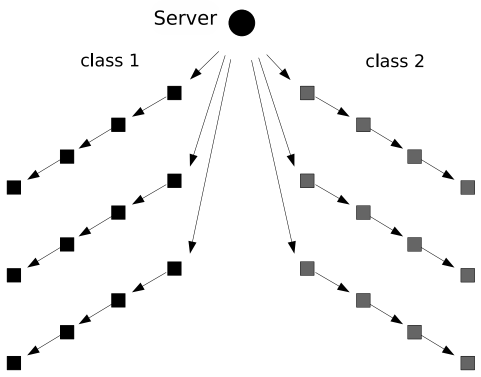


Figure 7: Chunk distribution with two independent classes

Generalizing the result of Eq. (2), it is straightforward to derive the time necessary to distribute the content to N_i peers with bandwidth b_i using a Linear scheme is

$$T^{\text{Class i}}_{\text{Lin, Ind}}(b_i, C, N_i) = \frac{F}{b_i} \cdot \frac{(C-2) + \sqrt{(C-2)^2 + 8N_iC}}{2C} \tag{7}$$

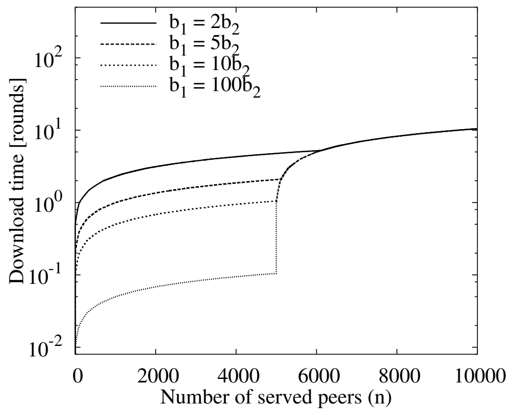
where F is the file size in bits and b_i the capacity of the class i in bit/s. Since the two classes evolve independently, the total time necessary to reach N peers depends, considering $N \gg C$, whether $N_2 > N_1 \left(\frac{b_2}{b_1}\right)^2$ (slow peers terminate after fast peers) or not (fast peer terminate after slow ones). The above threshold is obtained from the simplification of Equation (1).

Fig. 8 shows the total time against the number of peer n: in this example we have two classes with the same number of peers, i.e. $N_1 = N_2 = N/2$, where $N = 10^4$, and different bandwidth ratios: we assume that the bandwidth of class 2 is fixed, with $F/b_2 = 1$ and the bandwidth of class 1 is $2, 5, 10$ and 100 times greater.

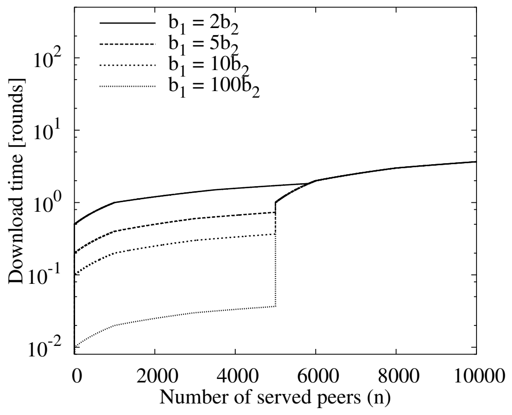
3.3 Linear Architecture with Generous Fast Peers

With this configuration slow peers do not upload any chunk; they only download from fast peers; fast peers upload in parallel the chunks to one fast and one slow peer. Each fast peer stops after it has completely served one slow peer. Fig. 9 shows the chunk distribution scheme in this case.

At the beginning, the evolution in time is equivalent to a single class with capacity $b_1^* = b_1 - b_2$. Each slow peer finishes to download $\frac{F}{b_2} + \frac{F}{Cb_1^*} - \frac{F}{b_1^*}$ rounds after the correspondent fast peer terminates. When all the peers of one class are reached by the content, the evolution of the system is different depending whether N_1 is larger or smaller than N_2. Let n be the number of peers that have already finished downloading at time t, $0 < n < N$. If $N_1 < N_2$ then fast peers finish before slow ones (see Fig. 9) and, when $n > 2N_1$, the remaining slow



(a) $C = 10^2$



(b) $C = 10^3$

Figure 8: Linear chain with independent classes: time necessary to complete the download ($N_1 = N_2 = N/2$, $N = 10^4$)

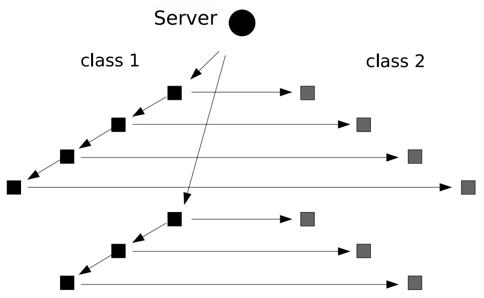


Figure 9: Chunk distribution with generous fast peers

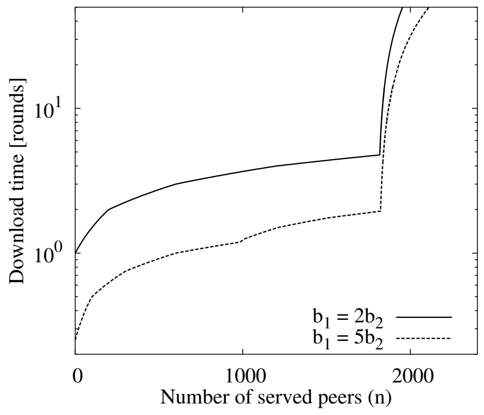
peers can only download from the server (they are not collaborative, so they do not upload to any other peers); in this case b_S/b_2 peers finish the download every F/b_2. If $N_1 > N_2$ then slow peers finish before fast ones and, when $n > 2N_2$, the remaining fast peers evolve with full bandwidth b_1.

Fig. 10 shows the behavior in two cases. When $N_1 < N_2$ (here $N_2 = 10N_1$, with $N_1 \simeq 900$) it is possible to see that, after $2N_1$, the system evolves very slowly, since only the server uploads the content. On the contrary, when $N_1 > N_2$ (here $N_1 = 10N_2$, with $N_2 \simeq 900$), only a small part of fast peers are involved in helping slow peers; after $2N_2$ peers are served, the system evolves faster. The figure shows how the system evolves: to see the difference between the phase when class 1 has a capacity equal to b_1^* and when it has a capacity equal to b_1, the dashed line represents the evolution if class had always a capacity b_1^*. In case of greater bandwidth ratios (not shown here), the difference becomes entirely negligible.

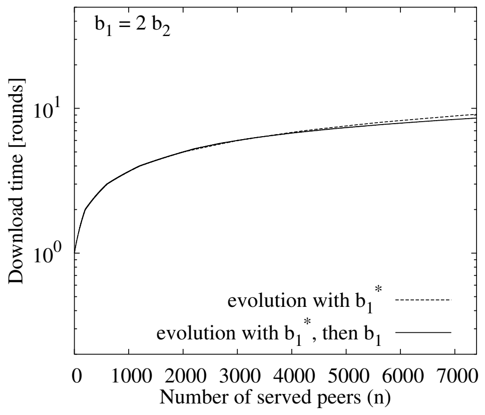
Considering the case $N_1 < N_2$ (the results for $N_1 > N_2$ are trivial), the total download time for class 1 is

$$T_{\text{Lin, Gen}}^{\text{Class1}}(b_1^*, C, N_1) = \frac{F}{b_1^*} \cdot \frac{(C-2) + \sqrt{(C-2)^2 + 8N_1C}}{2C} \,. \tag{8}$$

For class 2 we have to distinguish between the two phases: let n_2 be the number of slow



(a) $N_1 < N_2$



(b) $N_1 > N_2$

Figure 10: Linear chain with generous fast peer: time necessary to complete the download ($N = 10^4$, $C = 10^2$); (a) $N_1 < N_2$, when fast peers have completed, slow peers can download only from the server; (b) $N_1 > N_2$, after helping slow peers, fast peers evolve with full bandwidth

peers that have completed the download, we have

$$T^{\text{Class2}}_{\text{Lin, Gen}}(b_2, C, N_2) = \begin{cases} T^{\text{Class1}}_{\text{Lin, Gen}}(b_1^*, C, n_2) + \frac{F}{b_2} + \frac{F}{Cb_1^*} - \frac{F}{b_1^*} & \text{if } n_2 < N_1 \\ T^{\text{Class1}}_{\text{Lin, Gen}}(b_1^*, C, N_1) + \frac{F}{b_S} n_2 & \text{if } n_2 > N_1 \end{cases} \tag{9}$$

3.4 Linear Architecture with Generous Fast Peers and Collaborating Slow Peers

In this configuration the system evolves as in the previous case, except that each slow peer served by a fast peer starts a new chain of slow peers. In this case each fast peer serves one fast peer and one slow peer, and each slow peer serves another slow peer. Fig. 11 shows the chunk distribution methodology in this case.

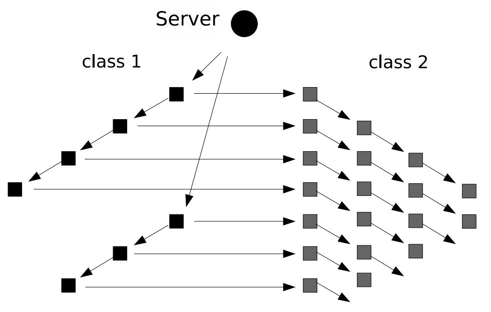


Figure 11: Chunk distribution with generous fast peers and collaborative slow peers

With this scheme we try to exploit the unused capacity of slow peers. While fast peers continue to upload chunks to slow peers, each slow peer starts a new chain. The rate of slow chain creation is equal to the rate new fast peers are involved in the distribution process.

Looking at fast peers, the download time can be found, as in the previous case, simply considering a Linear evolution with capacity $b_1^* = b_1 - b_2$. When class 2 completes before class 1, the remaining fast peers evolve with full bandwidth b_1. We can approximate the total download time of class 1 using an upload bandwidth b_1^* and obtain

$$T^{\text{Class1}}_{\text{Lin, GenColl}}(b_1^*, C, N_1) = \frac{F}{b_1^*} \cdot \frac{(C-2) + \sqrt{(C-2)^2 + 8N_1 C}}{2C} . \tag{10}$$

As slow peers are concerned, in order to find the total download time we first find the number of slow peers that have completed the download at time t and then it is possible to derive the formula of total download time against the number of served slow peers.

Consider the first chain of fast peers. A new fast peer is reached by a chunk every $F/(Cb_1^*)$, so the number of fast peers in the first fast chain is $\frac{t_{\text{class1}} - F/b_1^*}{F/(Cb_1^*)} + 1$, where, for notation simplicity, we use t_{class1} instead of $T(b_1, C, N_1)$. For each fast peer a new slow chain is started. The number of slow peers in a slow chain at time t is $1 + \frac{t - t_{\text{start}} - F/b_2}{F/(Cb_2)}$, where t_{start} is the time when the chain is started. The number of slow peers contained in all the slow chains generated by the first fast chain is then

$$\sum_{j=1}^{\frac{t_{\text{class1}} - F/b_1^*}{F/(Cb_1^*)} + 1} \max\left(0, \left\lfloor 1 + \frac{t - j\frac{F}{Cb_1^*} - \frac{F}{b_2}}{F/(Cb_2)} \right\rfloor\right) \tag{11}$$

where t_{class1} is the time necessary to class 1 to complete.

The second fast chain generates a number of slow chains equal to $\frac{t_{\text{class1}} - 2F/b_1^*}{F/(Cb_1^*)}$, so the number of slow peers contained in all the slow chains generated by the second fast chain is

$$\sum_{j=1}^{\frac{t_{\text{class1}} - 2F/b_1^*}{F/Cb_1^*} + 1} \max\left(0, \left\lfloor 1 + \frac{t - \left(j\frac{F}{(Cb_1^*)} + \frac{F}{b_1^*}\right) - \frac{F}{b_2}}{F/(Cb_2)} \right\rfloor\right) . \tag{12}$$

Applying the calculus for every fast peer chain, we obtain the total number of slow peer reached at time t

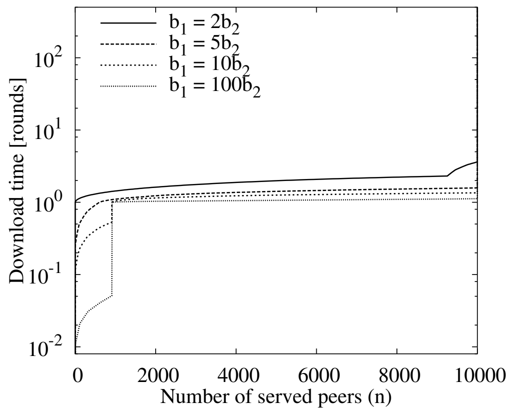
$$n_2(t) = \sum_{k=0}^{\frac{t_{\text{class1}}}{F/b_1^*}} \sum_{j=1}^{\frac{t_{\text{class1}} - (k+1)F/b_1^*}{F/(Cb_1^*)} + 1} \max\left(0, \left\lfloor 1 + \frac{t - \left(j\frac{F}{Cb_1^*} + k\frac{F}{b_1^*}\right) - \frac{F}{b_2}}{F/(Cb_2)} \right\rfloor\right) . \tag{13}$$

From this relation, it is possible to find

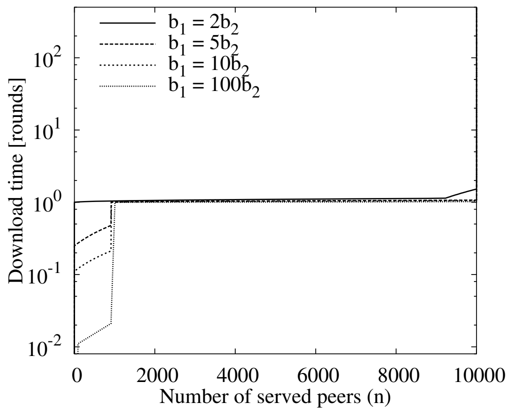
$$T^{\text{Class2}}_{\text{Lin, GenColl}}(b_2, C, N_2) \qquad \text{such that} \qquad n_2(T^{\text{Class2}}_{\text{Lin, GenColl}}) = N_2 . \tag{14}$$

It is important to note that equation (13) does not take into account the possible contribution of the server if it becomes available (this happens if class 1 terminates before all slow peers finish): nevertheless we consider the contribution not significant, since there is a slow chain started every fast peer.

Fig. 12 shows the total time versus the number of peers n: in this case $N_2 = 10N_1$, where $N_1 + N_2 = N = 10^4$. The collaboration of slow peers ensures that the whole capacity of the system is well exploited even when the number of fast peers is relatively small.



(a) $C = 10^2$



(b) $C = 10^3$

Figure 12: Linear architecture: system evolution with Generous scheme with Collaboration ($N_2 = 10N_1$, $N = 10^4$)

3.5 Linear Architecture with Altruistic Fast Peers

In this configuration each fast peer, after uploading the content to a fast peer, stays on-line and serves slow peers; when $b_1 \gg b_2$, parallel upload is employed to fully exploit the upload capacity of the fast peers and b_1/b_2 slow peers start to download from a single fast peer to finish their download F/b_2 time later. For simplicity, we suppose no collaboration of slow peers, i.e., slow peers do not start new chains. We suppose that fast peers are able to serve all the slow peers, i.e., $\frac{b_1}{b_2}N_1 > N_2$. Fig. 13 shows the chunk distribution scheme in this case.

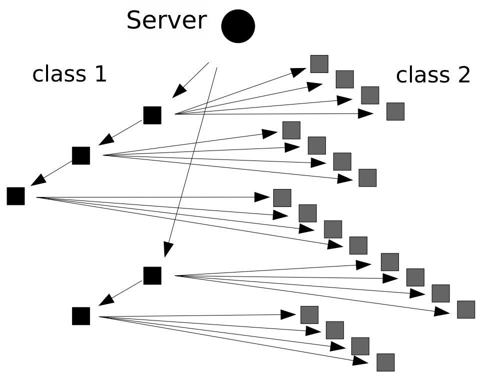


Figure 13: Chunk distribution with altruistic fast peers

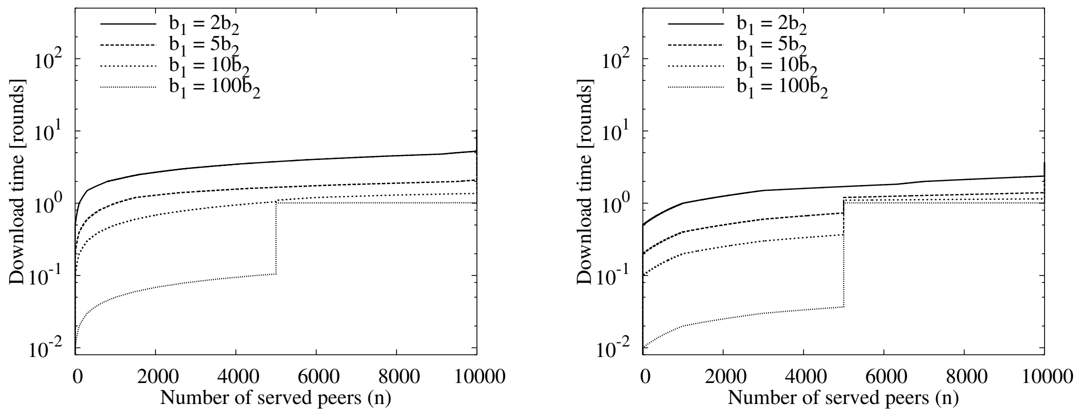
Considering an instant t, the number of slow peers that has completed is equal to the number of fast peers that have completed at time $t - F/b_2$ multiplied by a factor b_1/b_2. The total download time for class 1 is equal to the total download time found for the Independent case, i.e.,

$$T^{\text{Class1}}_{\text{Lin, Altr}}(b_1, C, N_1) = \frac{F}{b_1} \cdot \frac{(C-2) + \sqrt{(C-2)^2 + 8N_1C}}{2C}\ . \tag{15}$$

The total time for class 2 to complete is

$$T^{\text{Class2}}_{\text{Lin, Altr}}(b_2, C, N_2) = \tfrac{F}{b_2} + T^{\text{Class1}}_{\text{Lin, Altr}}(b_1, C, \tfrac{b_2}{b_1}N_2) \quad \text{given that } \ N_1 > \tfrac{b_2}{b_1}N_2 \tag{16}$$

where the last term is the time necessary to reach a number of fast peers that is able to serve all the slow peers. Fig. 14 shows the total time against the number of peers n, with $N_1 = N_2 = N/2$, where $N = 10^4$.



(a) $C = 10^2$ (b) $C = 10^3$

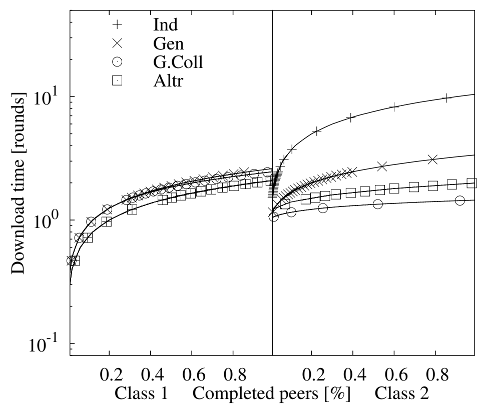
Figure 14: Linear chain architecture: system evolution with Altruistic scheme ($N_1 = N_2$, $N = 10^4$)

3.6 Comparative Analysis for Linear Architecture

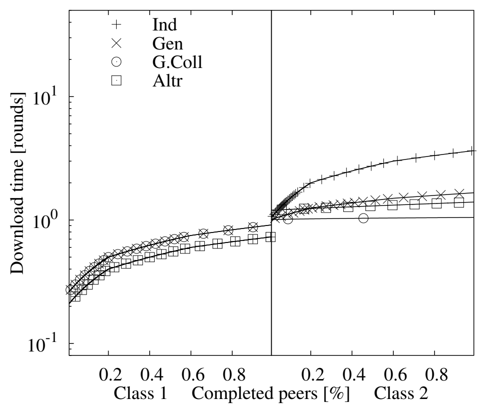
We consider the four cases obtained combining $b_1 = 5b_2$, $b_1 = 10b_2$ and $N_2 = N_1$, $N_2 = 10N_1$. In particular we focus on a total number of peers equal to 10^4: the analyzed cases correspond to (i) $N_2 = N_1 = 5000$ and (ii) $N_1 \simeq 900$ and $N_2 = 10N_1 \simeq 9000$. We suppose a number of chunks equal to 10^2 or 10^3. We consider the file size and the bandwidth b_2 such that $F/b_2 = 1$ round. Other combinations of parameters yield results that confirm the insight achieved with these simple cases.

Figs. 15 and 16 show how the system evolves in terms of the percentage of completed peers for each class. Fig. 15 shows the case $b_1 = 5b_2$, with $N_1 = N_2$, whereas Fig. 16 shows the case of $b_1 = 10b_2$, with $N_2 = 10N_1$. We see that when the fast peers help the slow ones, the performance of slow peers is greatly improved, especially for $N_2 = 10N_1$. The Generous scheme with Collaboration achieves near optimal performance, regardless of bandwidth ratio and number of peers ratio. In particular, in Fig. 15, class 2 finishes even before class 1. The reason is that a lot of slow chains are started almost at the same time (i.e., one slow chain every $F/(Cb_1)$, whereas only one fast chain every F/b_1 is created). Each of these slow chains, after the whole file is uploaded, adds a new slow peer every $F/(Cb_2)$, so in few slots of time (each slot is $F/(Cb_2)$) a lot of slow peers are reached.

When we have many more slow peers than fast ones (Fig 16) the download time with the Generous scheme is very high for most of the slow peers that will be served by the server.

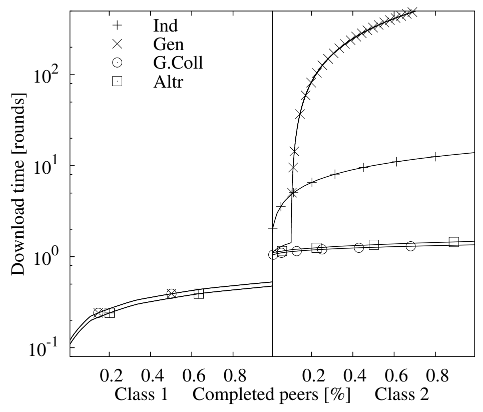


(a) $C = 10^2$

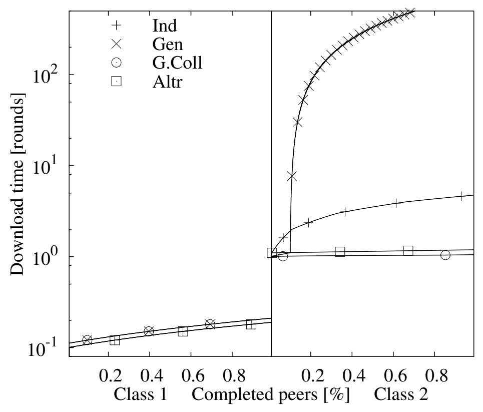


(b) $C = 10^3$

Figure 15: System evolution with Linear architecture ($b_1 = 5b_2$, $N_1 = N_2$, $N = 10^4$)



(a) $C = 10^2$



(b) $C = 10^3$

Figure 16: System evolution with Linear architecture ($b_1 = 10b_2$, $N_2 = 10N_1$, $N = 10^4$)

3.7 *Tree*^{k} and *PTree*^{k} Architectures and Overall Comparison for Heterogeneous Case

In order to compare different cooperation schemes compactly, we introduce a new metric that measures how much longer it takes to download the file to **all** peers of a class as compared to the time it would take a single peer of that class to download the file directly from the server.

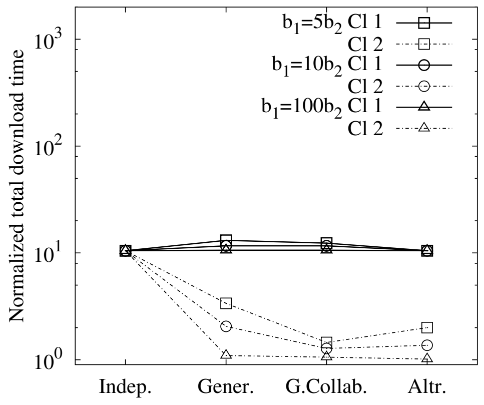
$$\text{Normalized Total Download Time for class i} = \frac{\text{Total Download Time for class i}}{F/b_i}$$

Regardless of the architecture, scheme or bandwidth ratios, a value of one for the *Normalized Total Download Time* is an optimal lower bound.

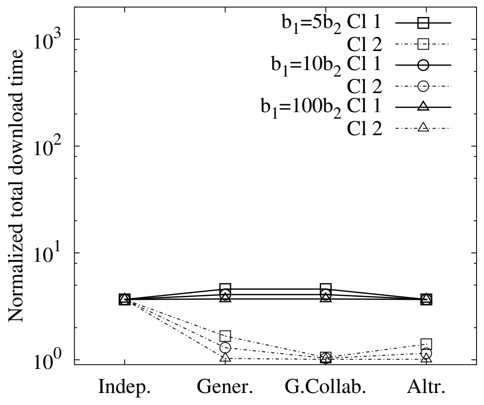
Figs. 17 and 18 show the complete set of results for $N_1 = N_2$ and $N_2 = 10N_1$ respectively, for two different values of C. For each scheme (reported on the x-axis) the normalized total download time of each class for different bandwidth ratios is shown.

Independently from the ratio between fast and slow peers ($N_1 = N_2$ and $N_2 = 10N_1$), the impact of a collaboration policy on the total download time of fast peers is negligible.

For class 2 (slow peers), the Generous scheme with Collaboration performance is close to optimal. The altruistic scheme performs nearly as well, with normalized download times that are slightly higher than for the Generous scheme with Collaboration. Notice that the collaboration of slow peers becomes fundamental when they outnumber fast ones.

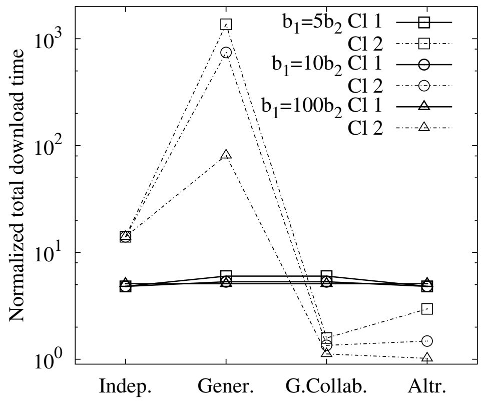


(a) $C = 10^2$

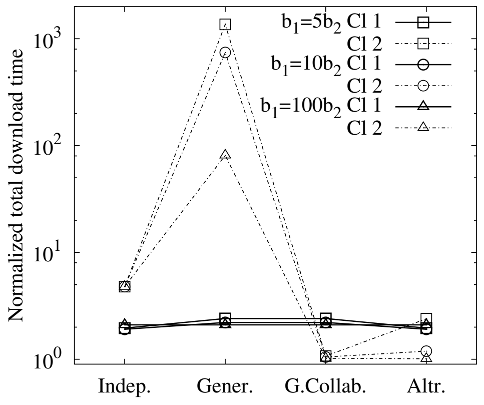


(b) $C = 10^3$

Figure 17: Normalized total download time with Linear architecture achieved by each class with different schemes ($N_1 = N_2$, $N = 10^4$)



(a) $C = 10^2$



(b) $C = 10^3$

Figure 18: Normalized total download time with Linear architecture achieved by each class with different schemes ($N_2 = 10N_1$, $N = 10^4$)

We want to compare the different cooperation schemes of the linear architecture and introduce results and

Completing the analysis requires the derivation of results for the *Tree*^{k} and *PTree*^{k} architectures. To avoid cluttering the paper we do not derive here the formulas for *Tree*^{k} and *PTree*^{k} for the heterogeneous case, since most of the calculations are cumbersome and the basic ideas used to find the final results are the same to those we adopt for the *Linear* architecture. The only new parameter introduced here is f: the outdegree used from fast peers to slow peers (while k is the outdegree within the same class). We present in Table 1 the final results that indicate how the different collaboration schemes perform for *Tree*^{k} and *PTree*^{k}, and refer to the technical report [19] for more details on how these formulas are obtained. Note that the Independent scheme is equivalent to he homogeneous case with bandwidth b_1 or b_2. As the other schemes are concerned, fast peers evolve as in the independent scheme, with a modified bandwidth; slow peers have a term equal to $\frac{F}{b_2}$ (that is the minimum time to upload the file F to a peer with bandwidth b_2) plus the time necessary for the fast peer to reach the leaves (Generous scheme) or to finish the download (Altruistic scheme).

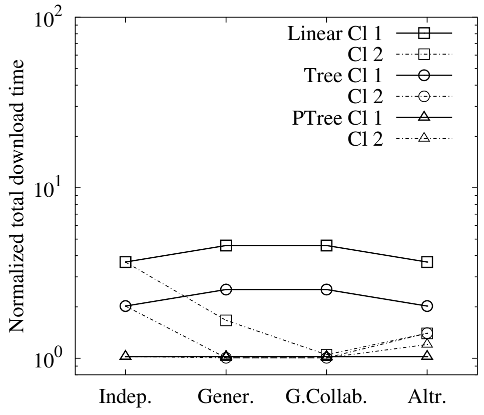
Table 1: *Tree*^{k} and *PTree*^{k} architectures performance comparison.

Architecture	Scheme	Dwld Time Class 1	Dwld Time Class 2
Linear	Indep.	$\frac{F}{b_1} \cdot \frac{(C-2)+\sqrt{(C-2)^2+8N_1C}}{2C}$	$\frac{F}{b_2} \cdot \frac{(C-2)+\sqrt{(C-2)^2+8N_2C}}{2C}$
	Gener.	same as Indep., with $b_1^* = b_1 - b_2$	$\frac{F}{b_2} + \frac{F}{Cb_1^*} - \frac{F}{b_1^*} + t^{\text{class1}}$
	G.+Coll	same as Indep., with $b_1^* = b_1 - b_2$	not explicitly reversible
	Altr.	same as Indep.	$\frac{F}{b_2} + t^{\text{class1}}$
Tree^k	Indep.	$\frac{F}{b_1}\left\{k + \frac{k}{C}\left[log_k\left(N_1\frac{k-1}{k}+1\right)-1\right]\right\}$	$\frac{F}{b_2}\left\{k + \frac{k}{C}\left[log_k\left(N_2\frac{k-1}{k}+1\right)-1\right]\right\}$
	Gener.	same as Indep., with $b_1^* = b_1 - fb_2$	$\frac{F}{b_1^*}\frac{k}{C}log_k\left(N_1\frac{k-1}{k}+1\right)+\frac{F}{b_2}$
	G.+Coll	same as Indep., with $b_1^* = b_1 - fb_2$	not explicitly reversible
	Altr.	same as Indep.	$\frac{F}{b_2} + t^{\text{class1}}$
PTree^k	Indep.	$\frac{F}{b_1}\left\{1 + \frac{k}{C}\left[log_k\left(N_1\frac{k-1}{k}+\frac{1}{k}\right)-1\right]\right\}$	$\frac{F}{b_2}\left\{1 + \frac{k}{C}\left[log_k\left(N_2\frac{k-1}{k}+\frac{1}{k}\right)-1\right]\right\}$
	Gener.	same as Indep., with $b_1^* = b_1 - fb_2/k$	$\frac{F}{b_1^*}\frac{k}{C}log_k\left(N_1\frac{k-1}{k}+\frac{1}{k}\right)+\frac{F}{b_2}$
	G.+Coll	same as Indep., with $b_1^* = b_1 - fb_2/k$	$\frac{F}{b_1^*}\frac{k}{C}log_k\left(N_1\frac{k-1}{k}+\frac{1}{k}\right)+\frac{F}{b_2}$
	Altr.	same as Indep.	$\frac{F}{b_2} + t^{\text{class1}}$

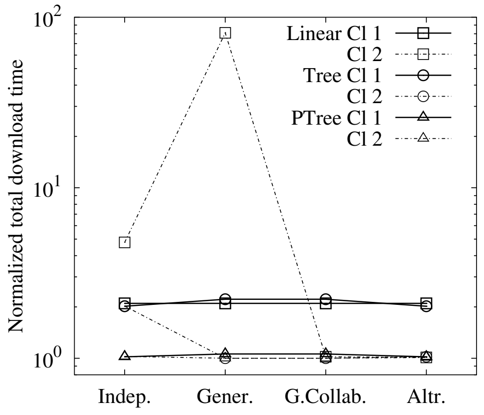
Fig. 19(a) shows the results for all three architectures when $b_1 = 5b_2$ and $N_1 = N_2$. Looking at class 1, as we already observed for the homogeneous case, the more sophisticated the architecture, the closer the total download time to the optimum. The adopted collaboration scheme has almost no impact on the performance of peers in class 1. The results obtained for class 2 can be quite sensitive with respect to the collaboration scheme: A Generous scheme can greatly improve the performance of slow peers and allows them to obtain, regardless of the architecture, a near-optimal download time.

These results are confirmed when we have many more slow peers than fast ones (see Fig. 19(b)). A small number of very fast peers can make the total download time of slow peers come close to the optimal one.

As the *PTree*^{k} architecture is concerned, near optimal performance for peers in both classes is achieved even if the peers in the two classes operate independently: this means that it is not possible to find a collaboration scheme that will significantly improve the performance, as it can be expected since the *PTree*^{k} scheme fully exploit all resources.



(a) $N_1 = N_2, b_1 = 5b_2$



(b) $N_2 = 10N_1, b_1 = 100b_2$

Figure 19: Normalized total Download Time for different architectures and collaboration schemes ($N = 10^4$, $C = 10^3$)

The most remarkable observation, however, is that heterogeneity does not necessarily jeopardize the performance. Indeed, in presence of heterogeneous peers, some simple forms of cooperation between different peer classes can help improving the performance of the slow peers without significantly affecting fast ones, even with simple distribution architectures.

4 Conclusions and Perspectives

The *self-scaling* and *self-organizing* properties of peer-to-peer networks allow to quickly and efficiently distribute content to huge client populations. 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 a wide range of parameters the *PTree*^{k} allows to serve a large population of peers in a time that is just slightly above the time it would take a single peer to download the file from the server, and this also happens independently from the presence of different speed in access links.

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.

Our results for a heterogeneous peer populations indicate that for the linear architecture when the fast peers help the slow peers, the slow peers can achieve close to optimal download times while the download time of the fast peers will not suffer. It is worth noticing how different organization schemes are affected by heterogeneity. Linear architecture, that in the homogeneous case is shown to have poor performances, in the heterogeneous case with helping fast peers obtains performance near to optimality. In the more sophisticated *Tree*^{k}

and *PTree*^{k} organization schemes, finding appropriate ways for fast peers to help slow ones is more difficult. In the PTree architecture in particular almost any devised helping scheme leads to poorer performances, at least for fast peers.

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 [1]). 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 *Tree*$^{k=2}$; and so on. Failures are 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 *PTree*^{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 [20], whose topology dynamically evolves according to predefined peer and chunk selection strategies, offer service times as low as the ones of *PTree*^{k} and adjust dynamically to bandwidth fluctuations, bandwidth heterogeneity, and node failures.

References

[1] B. Cohen, Incentives to build robustness in BitTorrent, in: Proceedings of P2P-Econ, June 2003.

[2] C. Gkantsidis, P. Rodriguez, Network coding for large scale content distribution, in: Proceedings of IEEE Infocom, Miami, FL, USA, Mar. 13-17, 2005.

[3] S. Saroiu, P. K. Gummadi, S. D. Gribble, A measurement study of peer-to-peer file sharing systems, in: Proceedings of Multimedia Computing and Networking (MMCN), San Jose, California, Jan. 2002.

[4] M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. Al Hamra, L. Garcés-Erice, Dissecting BitTorrent: five months in a torrent's lifetime, in: Proceedings of Passive and Active Measurements, Antibes Juan-les-Pins, France, April 2004.

[5] D. T. Zihui Ge, D. R. Figueiredo, S. Jaiswal, J. Kurose, Modeling Peer-Peer file sharing systems, in: Proceedings of IEEE Infocom, pp. 210–217, San Francisco, California, USA, Mar. 2003.

[6] X. Yang, G. de Veciana, Service capacity of Peer-to-Peer networks, in: Proceedings of IEEE Infocom, pp. 22422252, Hong Kong, Mar. 2004.

[7] L. Clevenot, P. Nain, A simple fluid model for the analysis of SQUIRREL, in: Proceedings of IEEE Infocom, pp. 85 - 95, Hong Kong, Mar. 2004.

[8] S. Iyer, A. Rowstron, P. Druschel, Squirrel: a decentralized, peer-to-peer web cache, in: ACM Symposium on Principles of Distributed Computing (PODC 02), pp. 213 - 222, Monterey, California, 2002.

[9] D. Qiu, S. Srikant, Modeling and performance analysis of BitTorrent-like Peer-to-Peer networks, in: Proceedings of ACM SIGCOMM, pp 367 - 378, Portland, OR, Sept. 2004.

[10] R. Gaeta, M. Gribaudo, D. Manini, M. Sereno, Analysis of resource transfer in Peer-to-Peer file sharing applications using fluid models, Performance Evaluation, Volume 63, Issue 3, pp. 149-174, March 2006.

[11] Y.-H. Chu, S. G. Rao, H. Zang, A case for end system multicast, in: Proceedings of of ACM SIGMETRICS, pp. 1 - 12, June 2000.

[12] D. Pendarakis, S. Shi, D. Verma, M. Waldvogel, ALMI: an application level multicast infrastructure, in: Proceedings of the 3rd Usenix Symposium on Internet Technologies & Systems (USITS), Mar. 2001.

[13] S. Banerjee, B. Bhattacharjee, C. Kommareddy, Scalable application layer multicast, in: Proceedings of ACM SIGCOMM, Pages 205 - 217, Aug. 2002.

[14] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, A. Singh, Split-Stream: high-bandwidth content distribution in a cooperative environment, in: Proceedings of ACM Symposium on Operating Systems Principles (SOSP), pp. 298 - 313, The Sagamore, New York, USA, Oct. 2003.

[15] L. Massoulie, M. Vojnovic, Coupon replication systems, in: Proceedings of ACM Sigmetrics, pp. 2 - 13, Banff, Alberta, Canada, June 2005.

[16] D. Stutzbach, D. Zappala, R. Rejaie, The scalability of swarming Peer-to-Peer content delivery, in: Proceedings of Networking, Waterloo, Ontario, Canada, May 2005.

[17] S.-W. Tan, A. G. Waters, J. Crawford, Meshtree: a selay optimised Overlay Multicast Tree Building Protocol, Univ. of Kent, Tech. Rep. 5-05, April 2005.

[18] E. W. Biersack, P. Rodriguez, P. Felber, Performance analysis of Peer-to-Peer networks for file distribution, in: Proceedings of 5th International Workshop on Quality of Future Internet Services (QofIS), Barcelona, Spain, Sept. 2004.

[19] D. Carra, R. L. Cigno, E. W. Biersack, Introducing heterogeneity in the performance analysis of P2P networks for file distribution, Tech. Rep. DIT-04-113, University of Trento, December 2004.

[20] P. A. Felber, E. W. Biersack, Self-scaling networks for content distributions, in: O. Babaoglu et al., editors, Self-Star Properties in Complex Information Systems, volume 3460 of Lecture Notes in Computer Science, Springer-Verlag, 2005.