PhD Dissertation

International Doctorate School in Information and Communication Technologies

DIT - University of Trento

Performance Evaluation of Overlay Content Distribution Systems

Damiano Carra

Advisor:

Prof. Renato Lo Cigno

Università degli Studi di Trento

Co-Advisor:

Prof. Ernst W. Biersack

Institut Eurécom

March 2007

to Matteo

to my mother, to my father

Abstract

The peer-to-peer (P2P) networking paradigm received a lot of attention in recent years. P2P systems construct an overlay network at the application level and do not require any modification to the existing networking infrastructure, which is in contrast to other technologies such as IPv6 or IP-level multicast that do require modifications inside the network. Therefore, P2P systems are very attractive for supporting new communications paradigms such as 'application level multicast' or 'distributed publish/subscribe'.

Research in peer-to-peer networks has so far mainly focused on content storage and lookup, but few efforts have been spent to analyze and optimize the distribution phase. Cooperative content distribution networks are inherently self-scalable, in that the 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 spontaneously adapt to the demand by taking advantage of the resources provided by every peer, but the organization of the distribution process, the network characteristics, and the peer behavior can greatly influence the overall performance.

In this work we analyze the performance that can be obtained by different distribution architectures in a heterogeneous environment, where peers have different access link bandwidths.

We start form a deterministic *case, where the peers know exactly the bandwidth of their neighbors and can apply smart distribution policies. We then consider the case when peers do not know their neighbor bandwidth, e.g., when peers participating in the distribution process might dedicate only part of the bandwidth and this fraction can change over time. The analysis of such case can be done for simple architectures, with specific constraints, like chains or tree where we fix the maximum depth.*

In order to consider general distribution architectures, we introduced a graph based approach in the analysis of the systems. In particular, we analyze the overly distribution building process as a Markov process, called Stochastic Graph Processes (SGP): with the help of SGP we find the main performance metrics, such as the download time distribution or the distance from the source. We are able to analyze trees and meshes with different system constraints, such as the number of neighbors, the minimum and maximum outdegree, or the lifetime of the nodes. This approach gives us a great flexibility in deciding the level of detail of the system we are analyzing.

We successfully apply the SGP formalism to file distribution and also to streaming services. In such a case, we are able to analyze mesh based streaming systems and to evaluate their performance, also in a comparative perspective. The performance metrics in this case are the delay of the received stream and the number of parents from which a node receives.

SGP represents a powerful analytical tool that allows for detailed comparisons between different systems of very large size that were previously not feasible. Our technique allows the precise quantification of the performance of different solutions and the evaluation of the impact of design parameter.

Keywords

Peer-to-Peer, Content Distribution, Modeling techniques, Stochastic Processes

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Distributing Content

In the last decade the problem of content distribution over the Internet has attracted much attention: providing a content wherever a user is, and whenever she/he wants it, represents a big challenge. The growth of the user's demand, driven by the spread of broadband access connections such as xDSL, cable and wireless, has been tackled by different approaches.

At a first stage, content providers increased the service capacity by deploying additional mirror servers: given a network of mirror servers, users can be automatically and transparently redirected to the optimal server. This solution was considered sufficiently efficient by many content providers. However, the cost of maintaining such an infrastructure may be high: the focus of the content provider should be to produce contents, rather than distributing them.

In this scenario, new companies started to offer content delivery services to content providers. The basic idea is building a private infrastructure, mainly on top of the Internet, dedicated to spread the content as nearly as possible to the end user. Such a Content Distribution Network (CDN) can be shared by multiple content providers, thus, reducing the cost of operation.

Nevertheless, the business model where the content providers produce the content and CDNs deliver it may not be applied to every cases. Small content providers may find the cost of CDNs too high. Moreover, we have witnessed recently to an increasing phenomenon where the users themselves become content providers. The success of YouTube or Google Video, where thousands of users publish their own videos, the proliferation of independent composers, songwriters or filmmakers that publish their contents on the web, are examples of a trend where everyone may potentially be interested in distributing a content. In such a case, these *content providers* may not exploit the advantages of CDNs due to their cost.

Another attempt to provide tools for content distribution is represented by the IP Multicast protocol. In this case the network itself is responsible for efficiently reducing the load on the network to the minimum necessary for the transmission, avoiding duplicated content over the same link. This model can be applied when many users ask for the same content at the same time. However, a problem with IP Multicast is the lack of general deployment, without collaboration among Internet Service Providers (ISPs) for creating a working infrastructure.

Given this scenario, a huge challenge is finding a solution for the content distribution problem that can be deployed today, with the existing infrastructure, at affordable cost.

1.2 Overlay Systems

One of the solution suitable to solve the content distribution problem is building the CDN as cooperative overlay, like peer-to-peer (P2P) networking. P2P systems construct an overlay at the application layer and offer a great flexibility in creating different services, without requiring any modification to the existing Internet. The novelty of the P2P paradigm relies on

the concept of *cooperation among users*. Cooperation, with users providing services to the community, has several beneficial effects on the global system performance: it permits to improve capacity, network reliability, and it makes the network more adaptive to the users needs.

The P2P paradigm has become very popular essentially for file-sharing: Napster, Gnutella, Kazaa, and are well known examples of P2P file-sharing applications. After the first wave of these applications, which often reaches the attention of mass media for they were used to exchange illegal contents, the P2P communication paradigm was used in different applications. The introduction of BitTorrent has moved the focus to an efficient content download. The primary aim of BitTorrent is the distribution of very large popular files (bulk data), such as operating systems updates, patches, antiviruses, to large communities of users, and for its characteristics it was considered as a candidate for efficient overlay content distribution.

Live media streaming and on-demand streaming services are expected to benefit from the P2P approach too. These applications relies on *Application Level Multicast*, an implementation of the multicast concepts to the application level that can be realized without changing the existing infrastructure. The distribution of streaming media content represents one of the most bandwidth intensive services available on the Internet.

P2P for content distribution has the appealing feature of self-scaling: each peer can play both roles, the one of a client and the one of a server, which implies that the amount of resources scales with the demand for service. The issue that needs to be addressed is how to *efficiently* use the resources of the overall P2P system. Most P2P systems were born with a make-it-do approach, applying heuristics and common good sense to most parts of the system. Nevertheless, many performance issues must be taken into account when an application with a potential impact, as P2P applications have, is deployed.

1.3 Motivations of the Thesis

1.3.1 Overlay System Performance

Despite the success of P2P system and many proposed systems, a lot of issues related to distribution of the content anywhere, anytime, anyone, remain open. Most of the research effort was focused on efficient lookup mechanisms (e.g., structured systems based on Distributed Hash Tables, DHTs), considering the download phase just a matter of time. Nevertheless, the search phase requires the exchange of a relatively small amount of data[1], while the download phase (content retrieval) represents a demanding aspect for the network.

Many studies set the P2P traffic as the dominant traffic over the Internet [1]. In such a scenario, the network resources are becoming scarce and application designers should start considering a re-engineering of the systems, or provide new systems with a performance oriented design. The change of the overlay protocols is simple, so each content provider can theoretically create its ad-hoc content distribution protocol.

Application layer has a limited view of the physical topology (mainly an approximated and indirect view through measures of the delay) and cannot rely on IP Multicast for efficient delivery. Thus, a content distribution application should be carefully designed taking into consideration the impact of such application on the network, considering all the possible design degrees of freedom and comparing the different solutions in order to provide systems as much efficient as possible, i.e., those efficiency can be compared to the IP Multicast.

The design process of such complex systems requires multiple iterations: the number of nodes involved in a content distribution network, in fact, is

[1] Even in an unstructured system, as Kazaa, smart policies, like hierarchical organization of the nodes, result in a limited traffic due to queries.

high and it is not possible to completely predict the behavior of a protocol, i.e., the effects of interactions among different nodes or between nodes and the network. The evaluation of the performance needs specific tools that are able to account for such a complex environment.

1.3.2 Performance Evaluation Tools

The application designer should evaluate, besides the formal correctness, the protocol performance given a set of performance indexes. There are essentially three available tools for such an evaluation: analytical modeling, simulation and prototypes. Even if the direct implementation and testing (using a simulator or a prototypes) may seem the simplest way, with P2P networks it is not easy to follow this approach. Simulations (or measures on prototypes) have scalability problems, i.e., they can be controlled and performed in a small environment, where it is possible to analyze correctly the behavior of the protocol. P2P networks represent the opposite of this situation: the typical size of a P2P community starts from thousands of users (for streaming services) to millions of users (e.g., file sharing). How this huge network size can be simulated?

If, on the one hand, analytical tools do not have scalability problems, on the other hand they deal with a simplified model of the system. Building an analytical model represents a challenging task: such a model should represent the essential features of the system we are studying, assuming that the behavior of some elements of the system have a small impact on the total performance. The problem with modeling is to find an analytical tool that is able to reduce as much as possible these simplifications.

1.4 Content of the Thesis and Contributions

1.4.1 Analysis of File Distribution Architectures: Deterministic Case

In this thesis we first consider the specific problem of how to distribute in the shortest possible time a file to a community of users organized as an overlay of peers. We assume the presence of two classes of peers (according to their bandwidth) and we consider an ideal situation where each peer knows the addresses and the bandwidths of all the other peers. We define three distribution architectures: linear chain, tree and a forest of trees called PTree. 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 different cooperation schemes between the classes in order to understand how heterogeneity affects the performance of the distribution scheme.

The analysis is carried out using basic combinatorial techniques: in fact, once the parameters are set — the number of nodes in each class, the ratio between bandwidths, the distribution architecture, the collaborations scheme — the distribution time is completely deterministic.

The results of the analysis shows the importance of basic design parameters, such as the number pieces into which the file is broken or the outdegree of the distribution trees. We can also evaluate which is the best cooperation scheme for the different situations, e.g., for different values of the ratio between the number of peers in each class and their bandwidth. These results can be applicable in controlled contexts, such as a network of a company, where the hypotheses we made are met. If the network administrator has to upgrade all the PCs of the company with a new software patch, he/she can use a file distribution application during the nighttime and set the distribution application and cooperation scheme suited for his

problem. In such a context the peer bandwidths are stable and each peer can maintain a database with all the peers.

Nevertheless, in *open* context, where peers may use their bandwidth for other applications, or they may be selfish we must resort other analytical tools.

1.4.2 Stochastic Analysis of File Distribution

For the case of unknown neighbor bandwidth, we develop a model that consider the transmission bandwidth between logically adjacent nodes as the only 'un-determined' of the problem, i.e., the only quantity that has a stochastic description. We also consider that peers can leave the system during the distribution process. We derive an approximated analytical model that yields the stochastic distribution of the file delivery performance as a function of the distribution of the peer bandwidth. The model allows to assess the impact of slow peers on the delivery and gives enough insight in the problem to devise dynamic distribution strategies to overcome the impact of slow peers.

We develop an overlay network simulator in order to validate the model and to analyze distribution architectures where the tree degree can vary dynamically based on locally available resources. In particular, we analyze the effect of bounds of the node outdegree (the tree degree local to a given node) on global performance. We also assessed the performance of the distribution in presence of selfish or malfunctioning peers as a function of the distribution architecture. Results shows that selfish peers have no great impact and the results on performance we obtain considering no selfish peer still hold.

The distribution architectures that we are able to analyze with the analytical model we developed have some constraints: for instance, there must be a maximum number of hops from the source, i.e. all the leaves of the

distribution process (peers that download only and do not find any peer without the content to upload to) are at the same distance (in terms of number of hops) from the source. These constraints limits the distribution protocols that is possible to analyze, thus we consider a different analytical tool.

We formalize the problem of building a content distribution overlay as a Constrained Stochastic Graph Process (CSGP), which is a discrete time Markov chain whose states are graphs with additional constraints describing the features of the distribution system. The graphs we are interested in are directed acyclic graphs, such as trees or meshes. Depending on how we define the transition rules from one state of CSGP to the next one, we get different content distribution overlays. We consider three different overlays, two of which are trees (hop-driven and cost-driven trees) and the third one is a mesh.

The transient analysis of a Markov process is done considering the formulation of the problem in terms of Master Equations (MEs), i.e., the Kolmogorov forward (or backward) equations, coupled with the Chapman-Kolmogorov Equations. The structure of the transition matrix that describes the stochastic process, and the correspondent MEs, is extremely suited for an efficient numerical solution based on Monte Carlo techniques — basically a random walk in the state space of the process. Monte Carlo integration converges very quickly allowing to compute the metrics of interest for the content distribution — such as the file download times, and the percentage of upload bandwidth left unused —for very large overlays with up to millions of nodes, where standard event-driven simulations would fail for lack or time or memory.

1.4.3 Stochastic Analysis of Overlay Streaming Systems

The formalism of CSGP represents a flexible tool and it can be applied to other content distribution problem different form file distribution. Based on the same principles that guided the analysis of file distribution, we develop a mathematical model based on graph theory that can be used to analyze fundamental performance issues of overlay streaming services. We model such systems with a high level of abstraction that allows the study of fundamental behavior under different conditions.

We derive the master equations that define the evolution of the streaming system in time, based on the basic characteristics of the streaming protocol as well as the bandwidth available at nodes for the streaming application. The model allows the assessment of the impact of different protocol choices, and of bandwidth heterogeneity on the delivery process and it gives enough insight in the problem to formulate improved streaming strategies.

The model is solved with the Monte Carlo integration methodology used for CSGP. The solution provided by this method is compared with other modeling and solution techniques to show the flexibility of the approach. The solution of the model yields the entire probability distribution of the results (not only the mean value), as well as the temporal (transient) dynamics.

1.5 Topics not Covered in the Thesis

There are important issues in Overlay Content Distribution that are not treated in this thesis:

- *Specific system details*: We do not attempt to model a specific file-distribution system, although this task may be possible using the

proposed methodology. We are also not proposing yet-another file swarming application or a specific file distribution protocol. Rather, we propose a methodology to capture general properties of distribution protocols and algorithms in the attempt of providing guidelines for the protocol and system design.

- *Security*: we always consider collaborative nodes without malicious peers. Content distribution may be affected by badly behaving peers, that block the distribution process or provide fake content (pollution). System should consider these cases and introduce policies that are able to handle such nodes.

- *Overlay management*: building an overlay network that is closely related to the physical topology represents a crucial aspects for increasing the performance. We always assume an overlay management that take care of this aspect, working on the resulting overlay graph. The close mapping between overlay and underlay layers is a general requirement that do not depend on the actual information exchanged and should be followed by any P2P application.

- *Network related issues*: the actual distribution of the content pass through transport and network protocols that introduce many dynamics during the content exchange. The throughput achievable depends not only on the access link bandwidth, but also on TCP window or congestion inside the network. During the evaluation of a content distribution protocol, these aspects should be taken into account (e.g., for a streaming application, by sizing accordingly the playout buffer).

1.6 Related Works

Performance analysis of P2P systems is still in its infancy, and there are only few works on the argument. Most of the analytical works analyzes a specific P2P system, and often the problem is stated very differently from the perspective of this thesis. The work in [2] is among the first to evaluate the performance of a P2P system, through the representation as a multi-class closed queuing network. In [3], 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. Fluid models have been also considered since they can efficiently describe such an amount of transferred data. The work in [4] proposes a fluid model for the analysis of the Squirrel protocol [5]. The result is an accurate model that estimates the performance of the protocol. In [6] 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 few simple parameters; moreover, an analysis of the different mechanisms of BitTorrent is provided. Of the works above only [2] tackles the problem of presence of different access bandwidths among peers which is instead the one of the purposes of this thesis.

A related topic where distribution architectures are explicitly taken into account is the delivery of streaming services through overlay multicast, also known as Application Level Multicast (ALM).

Systems such as ALMI [7], NICE [8] and Zigzag [9] organize the nodes following a tree structure. The stream is received from a single father and uploaded to a set of children (if any). The differences among these systems concern algorithms used for managing the structure in case of node dynamics. Systems such as Narada [10], Coolstreaming [11] PRIME [12] and PULSE [13] leverage on mesh structure. Nodes download from a set

of fathers that can change over time. Problems related to delay and synchronization are handled according to different heuristics. Other systems adopt an hybrid approach (e.g., SplitStream [14], from which the inspiration for the architecture we use as reference comes), where the stream is distributed using multiple trees obtaining a structured mesh.

The above distribution protocols were not designed with a performance oriented approach as far as delivery is concerned. Many proposals use heuristic methods to improve performance, but these heuristics are verified a posteriori and protocol parameters are tuned according to these results. Most of the analytical works focus on tree based structures or on a specific system, but, to the best of author's knowledge, no study has been done on modeling general mesh-based streaming systems. Only [12] starts analyzing such systems, but with a simulative approach and assuming a homogeneous access bandwidth.

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

Very few models have been proposed that allow comparative studies of different distribution architectures. In [18], inspired by SplitStream, the authors have defined and analyzed linear chain and tree-based architectures in a completely deterministic setting. The work in [19] defines a model for chain-based and tree-based architectures, uses max-plus algebra and considers an infinite number of packets to calculate the long term average throughput; our analysis instead considers a finite file size and calculates either the download time of peer i or the mean (and total) download time of all the peers involved in the distribution process.

Stochastic graph processes, the analytical tool we introduce in Chapter 4 to model overlay content delivery networks, were defined in [20] with

the same notation we use here and they are based on well known random graphs [21]. The focus of the analysis in [20][21] is the topological properties of random graphs, whereas our aim is to take into account not only connections among nodes, but also their weights given by the bandwidths of the nodes involved (this concept is clarified in the chapter), which give rise to the state reward structure that allows the computation of completion times.

1.7 Structure of the Thesis

The remaining of the thesis is composed by the following research chapters.

Chapter 2 presents an extension of the model for file distribution proposed in [18]: in this case we introduce multiple classes of peers and we propose different collaboration policies among classes. We consider the best case, i.e., a fully connected overlay graph with all peers that know the bandwidth of their neighbors. In this case, the deterministic problem can be solved with traditional combinatorial techniques.

In Chapter 3 we remove the hypothesis that peers are able to know the neighbor bandwidths: in such a scenario we consider the distribution architectures defined in the previous chapter and we analyze the download time. We compare our results with the deterministic case, showing the effect of limited knowledge on the performance.

Chapter 4 introduces a new performance evaluation tool based on Stochastic Graph Processes. With such a tool, we are able to analyze and compare general file distribution architectures and schemes.

In Chapter 5 we successfully apply SGP tool to the streaming case. We model mesh based distribution architecture and evaluate the performance under different node behavior.

Chapter 6 summarizes the results and the contribution of the thesis.

Chapter 2

Deterministic Analysis of File Distribution

This chapter considers the deterministic analysis of a heterogeneous P2P network with two classes of peers. The analysis introduces a set of schemes that can be used with different distribution architectures and evaluates the performances that can be obtained with each scheme.

2.1 Introduction

In this chapter we start from the analysis of the file distribution architectures made in [18] and consider different scenarios. The main hypothesis that we remove with respect to the previous analysis is the assumption of homogeneous sources: we suppose that peers in the network can have different available bandwidths.

In particular we consider the presence of two classes of peers: for each proposed architecture – *Linear, Tree and PTree* – we present four different schemes – *Independent, Generous, Generous with Collaboration and Altruistic* – that can be applied for the distribution of files. Each class is structured in the above mentioned architectures and the only degree of freedom is how a class can help the other class in order to maximize the

overall performance.

We focus our attention on three performance indexes: the total download time necessary to reach N peers, the mean download time and the amount of work done (how many file copies each peer uploads). The results we obtain show that, when we consider heterogeneity, slow peers may not negatively affect the system. In general, the simpler the adopted delivery architecture is, the more improvement can be obtained with a intelligent distribution policy; the more complex the delivery architecture is, the less the gain we have.

The rest of the chapter is organized as follows. Section 2.2 describes the distribution architectures, the collaboration schemes and the system characteristics with two classes of peers. Sections 2.3, 2.4 and 2.5 analyzes the three different architectures, Linear, Tree and PTree respectively; for each architecture a set of distribution scheme is defined. Each section concludes with a summary of the results. Section 2.6 presents the comparison among different architectures and schemes. Section 2.7 summarizes the conclusions of this chapter.

2.2 Architectures, Schemes and General Assumptions

2.2.1 Architectures

Chains

The simplest architecture that can be defined is a single chain, where each node downloads from exactly one node and uploads to exactly one node. The distribution process generates different, parallel, independent chains, and a peer can be a node in one chain only. Even if this architecture seems too simple to be of practical interest, it serves as a basis for comparative analysis. Moreover, it can be used as simple building block that can form

more complex distribution architectures, and in some cases it might suffice for the application purposes. It is worth noticing that chains are fair in the sense that each peer (excluded the last one in the chain) uploads exactly the same amount of information it downloads.

Tree

In tree structures, each node downloads from exactly one node and uploads to k nodes, where k is the *outdegree* of the tree. The choice of k has two opposite effects: On the one side, by increasing k it is possible to reach the leaf nodes in fewer steps. On the other side, the bandwidth used in transferring the content from a peer to its children is divided among k peers. Therefore, increasing k will increase the total download time from one level of the tree to the next one. Considering a theoretical model where all peers have the same bandwidth and the tree is balanced, the binary tree is the best compromise between these two effects [18].

Tree based distribution architectures are an efficient way to distribute contents, given performance and simplicity. Nevertheless they suffer from two main shortcomings: (i) only interior nodes upload to other nodes, whereas leaf nodes do not upload at all and (ii) nodes use only part their download bandwidth (if the tree is balanced and peers are homogeneous, each node receives $1/k$ of the uploading bandwidth of its parent node). The first aspect results in *unfairness* among nodes: some nodes upload k times the amount of data they receive, while other nodes (leafs) only receive data. Since the number of leaf nodes is greater than the number of interior nodes[1], this also implies a great waste of upload bandwidth, since the majority of the nodes do not contribute to the distribution of the content. The second aspect implies that the time necessary to complete the download is k times

[1]In a balanced tree with l levels and outdegree k, $k \geq 2$, the number of leaf nodes and interior nodes is respectively k^l and $\frac{k^l-1}{k-1}$.

larger than the time it would take if the file were downloaded using the full download bandwidth of the parent node (this is strictly true for the homogeneous case, both with symmetric and asymmetric bandwidths).

PTree

The overall performance of a tree based architecture is significantly improved by capitalizing the unused upload bandwidth of the leaf nodes and the unused download bandwidth of internal peers. A solution in this direction is given by SplitStream [14]. The SplitStream architecture was defined for streaming services in a structured P2P system (based on Pastry). In the general case where we do not consider any particular structure for the overlay network, we call this architecture PTree.

In PTree the content is divided into r different stripes and each stripe is distributed by a different tree. A peer participates in all distribution trees, but with different roles. Precisely, a peer is an interior node in at most one tree, while it is a leaf node in the remaining $r-1$ trees. In this situation, when the node is interior, it uploads one stripe of the file k times, while it is receiving the other $r-1$ stripes in parallel since it is a leaf node of the other trees. If we choose to divide the file in exactly k stripes, i.e., $r = k$, we obtain complete fairness (see Fig. 2.1): each peer uploads the same amount of data it receives. There is only one exception: independent of the outdegree k, there will always be one peer that is leaf node in all r trees (the shaded circle in Fig. 2.1).

If we choose $r < k$, we obtain a situation where some peers are interior nodes of one tree and leaves of the other trees and they upload k/r times what they receive, whereas other peers are only leaf nodes, so they only receive the whole file. This sort of mild unfairness can be considered acceptable, since nodes behind firewalls and NAT devices can not participate actively in the distribution process. We do not consider the case $r > k$:

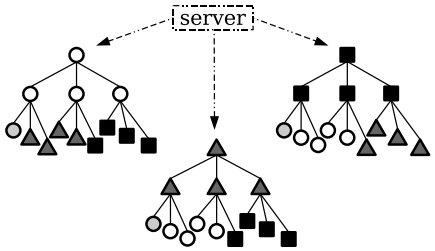


Figure 2.1: Example of a PTree distribution architecture with 13 nodes and 3 stripes, each node appears once in every distribution tree.

the number of leaf nodes of one tree is not sufficient to cover the interior nodes of the other $r-1$ trees, and a subset of nodes must act as interior node in more than one tree. We consider only the case $r \leq k$.

2.2.2 Schemes

Given a distribution architecture, we consider four different schemes of cooperation among peers belonging to different classes. Based on the access link bandwidth, we identify a *slow peer* class and a *fast peer* class.

Independent: the server uploads the content independently to each class; peers belonging to one class do not exchange packets with peers of the other class. This scheme, with no interaction between classes, serves as a reference.

Generous: slow peers do not upload; they only download from fast peers; fast peers upload in parallel to a fast peer and to a slow peer.

Generous with Collaboration: the system evolves as with the Generous scheme, but here each served slow peer collaborates in distributing the content to other slow peers. With this scheme we try to exploit the unused capacity of slow peers.

Altruistic: each fast peer, after uploading the content to a fast peer, stays on-line and serves in parallel as many slow peers as possible.

2.2.3 General Assumptions

We consider a P2P system where each class is symmetric, i.e., peers that belong to a class have the upload capacity equal to the download capacity. In case of asymmetric capacities[2], our analysis could give some insights if we consider as reference bandwidth the upload capacity (that is typically the smallest). Nevertheless, we leave for future works the detailed comparison of the different scenarios.

The main metrics of interest are (i) the total download time necessary to N users to complete the download of a file, (ii) the average download time and (iii) the amount of work done by each peer, i.e., how much they contribute to upload (this metric can be considered as a fairness index).

The main hypotheses of our model are the following:

- 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; $b_1 > b_2$;
- 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 $N = N_1 + N_2$;
- all the peers know all the other peers, including their bandwidths;
- there is exactly one server that has the original content; it is always on-line and it can indefinitely upload the file to new peers;
- we focus on the distribution of a single file $\mathcal{F}$; the file is divided in C identical pieces called "chunks" and each chunk can be distributed independently;
- 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

[2]For instance, ADSL users have the download capacity greater than the upload capacity.

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).

Even if the hypothesis of global knowledge can be unrealistic, it is necessary in order to understand the optimal case. In Chapter 3 we will remove this hypothesis analyzing the effect of limited knowledge. On the contrary, the hypothesis of knowing to some extent the capacity of each link is realistic, since most of the current systems can estimate the bandwidth of a peer during previous exchanges (and each peer can also communicate its own capacity).

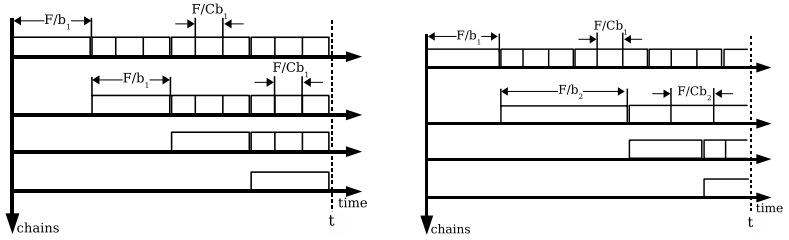
Aim of this study is to find optimal policies (schemes) for file distribution in such environment; depending on the file distribution policy, the performance can be greatly different.

2.3 Linear

In this particular architecture, parallel upload should not be considered; nevertheless, in order to exploit the capability of the system it is necessary to allow some special configuration with parallel upload. For instance, it is possible for a fast peer to upload in parallel to exactly one slow peer, with bandwidth b_2, and one fast peer, with bandwidth $(b_1 - b_2)$. Another case is when fast peers are altruistic, so they use their capacities to help slow peers.

[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.

Under the hypotheses made above it is possible to identify a set of cases. In the following paragraphs we analyze the cases considered most interesting.



(a) Single class (b) Two classes

Figure 2.2: Linear chain: number of peers per chain versus time with Linear architecture (C=3): the server upload to a single class (a) and alternatively to two classes (b)

Figure 2.2 shows a basic scheme to calculate the number of peers in time. Figure 2.2(a) shows the evolution when there is a single class: each row corresponds to a single chain. The first peer finishes to download after $\mathcal{F}/b_1$ rounds; the second peer finishes at $\mathcal{F}/b_1 + \mathcal{F}/Cb_1$ since it has to wait that the first peer has downloaded the first chunk before starting the upload to the second peer. A new chain is started when the file is completely downloaded from the first peer. Figure 2.2(b) considers the case with two classes when a server alternatively serves a fast peer (with bandwidth b_1) and a slow peer (with bandwidth b_2).

If we observe the system at certain time t it is possible to calculate the number of chains and the number of peers per chain. In general, through this kind of scheme we can evaluate the number of peers that have completed the download versus time for all the policies we define.

2.3.1 Independent

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. Figure 2.3 shows the chunk distribution methodology.

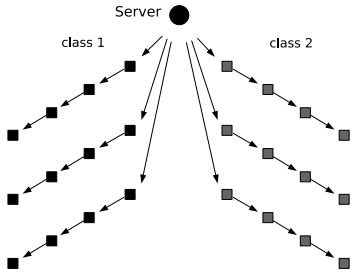
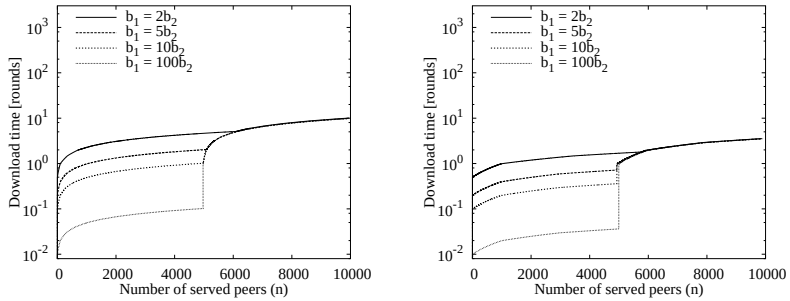


Figure 2.3: Chunk distribution with two independent classes

Total download time. Generalizing the method used in [18], it is simple to find that the time necessary to distribute the content to N_i peers with bandwidth b_i with a Linear scheme is

$$T^{\text{Class i}}_{\text{Lin, Ind}}(b_i, C, N_i) = \frac{\mathcal{F}}{b_i} \cdot \frac{(C-2) + \sqrt{(C-2)^2 + 8N_iC}}{2C} \tag{2.1}$$

where $\mathcal{F}$ 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 is dominated by slow peers. Figure 2.4 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 $\mathcal{F}/b_2 = 1$ and the bandwidth of class 1 is two, five, ten and 100 times greater.



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

Figure 2.4: Linear chain with independent classes: time necessary to complete the download ($N_1 = N_2 = N/2$, $N = 10^4$)

Average download time. Another important metric that can be useful to understand the performance is the average download time. We consider the sum of the times necessary to complete the download of each peer and then we divide it by the number of peers.

Referring to Fig. 2.2(a), consider the first chain. The first peer finishes to download at time $\frac{\mathcal{F}}{b_i}$, the second peer at time $\frac{\mathcal{F}}{b_i} + \frac{\mathcal{F}}{Cb_i}$, the third at time $\frac{\mathcal{F}}{b_i} + 2\frac{\mathcal{F}}{Cb_i}$ and so forth. So at time t in the first chain the sum of each single download time is

$$\sum_{j=0}^{\frac{t-\mathcal{F}/b_i}{\mathcal{F}/Cb_i}} \frac{\mathcal{F}}{b_i} + j\frac{\mathcal{F}}{Cb_i} \ . \tag{2.2}$$

The second chain is equal to the first, with a delay of $\frac{\mathcal{F}}{b_i}$, and the third chain is delayed $2\frac{\mathcal{F}}{b_i}$. Let $t_{\text{class i}}$ the time necessary to complete all the downloads of a specific class i, i.e., $t_{\text{class i}} = T^{\text{Class i}}_{\text{Lin, Ind}}(b_i, C, N_i)$ (we will use the simplified form $t_{\text{class i}}$ where it is clear which architecture and scheme we are using, otherwise we will use the complete form). Since in the system there are

$\frac{t_{\text{class i}}}{\mathcal{F}/b_i}$ chains, the sum of the total download times D_i for each class is

$$D_i = \sum_{k=1}^{\frac{t_{\text{class i}}}{\mathcal{F}/b_i}} \sum_{j=0}^{\frac{t_{\text{class i}}-k\mathcal{F}/b_i}{\mathcal{F}/Cb_i}} k\frac{\mathcal{F}}{b_i} + j\frac{\mathcal{F}}{Cb_i} \ . \tag{2.3}$$

The value of $t_{\text{class i}}$ can be found with (2.1). The average download time is D_i/N_i for each class and $(D_1 + D_2)/(N_1 + N_2)$ globally.

Amount of work. The number of copies distributed by the server is equal to the number of started chains for each class. In this case we have $\sum_i \frac{t_{\text{class i}}}{\mathcal{F}/b_i}$.

As regards fast and slow peers, each peer uploads once. Only the last peer of each chain does not upload; we can consider that the impact on the index is not significant.

2.3.2 Generous

With this configuration slow peers do not upload any chunk; they only download from fast peers; fast peers upload in parallel the chunks to a fast peer and to a slow peer. Each fast peer stops after it has completely served one slow peer. Figure 2.5 shows the chunk distribution methodology in this case.

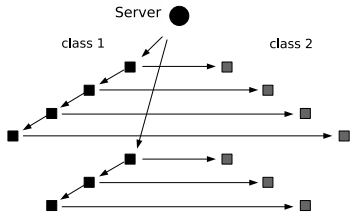


Figure 2.5: Chunk distribution with generous fast peers

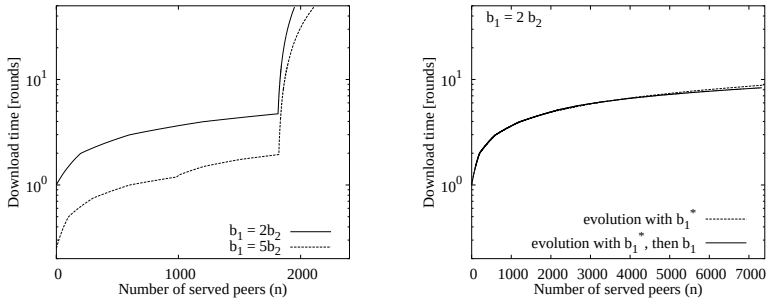
Total download time. The system at the beginning evolves as if there were only a single class with capacity $b_1^* = b_1 - b_2$. Every helped slow peer finishes to download $\frac{\mathcal{F}}{b_2} + \frac{\mathcal{F}}{Cb_1^*} - \frac{\mathcal{F}}{b_1^*}$ rounds after the correspondent fast peer terminates. When the peers of one class finish downloading, 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. 2.5) and, when $n > 2N_1$, the remaining slow peers can only download form the server (they are not collaborative, so they don't upload to any other peers); in this case b_S/b_2 peers finish to download every $\mathcal{F}/b_2$. If $N_1 > N_2$ then slow peers finish before fast ones and, when $n > 2N_2$, the remaining fast peers evolves with full bandwidth b_1.

Figure 2.6 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 $N_2 = 10N_1$, with $N_2 \simeq 900$), only a small part of fast peer 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 fully disappear.

The total download time for class 1 is then

$$T^{\text{Class1}}_{\text{Lin, Gen}}(b_1^*, C, N_1) = \frac{\mathcal{F}}{b_1^*} \cdot \frac{(C-2) + \sqrt{(C-2)^2 + 8N_1C}}{2C} \ . \tag{2.4}$$

For class 2 we have to distinguish between the two phases: let n_2 the



(a) $N_1 < N_2$ (b) $N_1 > N_2$

Figure 2.6: Linear chain with generous fast peer: time necessary to complete the download ($N = 10^4$, $C = 10^2$). With $N_1 < N_2$ (a), when fast peers have completed, slow peers can download only from the server. With $N_1 > N_2$ (b), after helping slow peers, fast peers evolve with full bandwidth

number of slow peers that have completed the download, we have

$$T^{\text{Class2}}_{\text{Lin, Gen}}(b_2, C, N_2) = \begin{cases} T^{\text{Class1}}_{\text{Lin, Gen}}(b_1^*, C, n_2) + \frac{\mathcal{F}}{b_2} + \frac{\mathcal{F}}{Cb_1^*} - \frac{\mathcal{F}}{b_1^*} & \text{if } n_2 < N_1 \\ T^{\text{Class1}}_{\text{Lin, Gen}}(b_1^*, C, N_1) + \frac{\mathcal{F}}{b_S} n_2 & \text{if } n_2 > N_1 \end{cases} \tag{2.5}$$

Average download time. The average download time has to be calculated in the two different cases.

When $N_1 < N_2$, the average download time of the N_1 fast peers can be calculated with (2.3), where the bandwidth is $b_1^* = b_1 - b_2$. For class 2, each of the N_1 slow peers finishes with a delay of $\frac{\mathcal{F}}{b_2} + \frac{\mathcal{F}}{Cb_1^*} - \frac{\mathcal{F}}{b_1^*}$ with respect to each fast peer, so we can use again (2.3), with $b_i = b_1^*$ and considering that delay by simply adding at the end $N_1 \left(\frac{\mathcal{F}}{b_2} + \frac{\mathcal{F}}{Cb_1^*} - \frac{\mathcal{F}}{b_1^*} \right)$. The remaining $N_2 - N_1$ slow peers complete at the rate of b_S/b_2 peers every $\mathcal{F}/b_2$. So the

sum of all the delays experimented by these peers is

$$\sum_{k=1}^{\frac{N-2N_1}{b_S/b_2}} t_{\text{class1}} + k\frac{\mathcal{F}}{b_2} \tag{2.6}$$

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

When $N_1 > N_2$ we can approximate the average download time of class 1 with (2.3) using the bandwidth b_1^*. For class 2, we can calculate the sum of the delays with (2.3), with bandwidth b_1^*; the value of $t_{\text{class i}}$ can be calculated from (2.1) using a number of peers equal to N_2; finally the term $N_2 \left(\frac{\mathcal{F}}{b_2} + \frac{\mathcal{F}}{Cb_1^*} - \frac{\mathcal{F}}{b_1^*} \right)$ must be added in order to consider the delay necessary to complete the download.

Amount of work. The number of copies distributed by the server is equal to the number of started fast chains. If the number of fast peer is not sufficient to serve all the slow peers, the server must upload to the remaining slow peers, so we have $\frac{t_{\text{class i}}}{\mathcal{F}/b_i^*} + \max(0, (N_2 - N_1))$. This represent an approximation since we do not distinguish between $N_1 > N_2$ and $N_1 < N_2$; nevertheless this approximation has not a great impact on final results.

As regards peers, each fast peer uploads at most twice (depending on the number of slow peers), whereas slow peers do not upload.

2.3.3 Generous, with Collaboration

In this configuration the system evolves as in the previous case, but here each slow peer served by a fast peer starts a new chain. In this case each fast peer serves one fast peer and one slow peer, and each slow peer serve one slow peer. Figure 2.7 shows the chunk distribution methodology in this case.

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

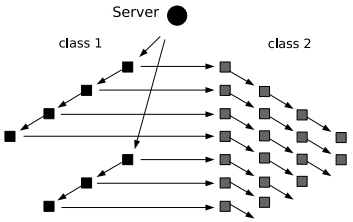


Figure 2.7: Chunk distribution with generous fast peers and collaborative slow peers

starts a new chain. The rate of slow chain creation is equal to the rate new fast peers are involved in the distribution process.

Total download time. Looking at fast peers, the download time can be found, as in the previous case, simply considering a Linear evolution with capacity $b_1^* = b_1 - b_2$. In case of class 2 complete before class 1, the remaining fast peers evolve with full bandwidth b_1. We can approximate the total download of class 1 considering the capacity b_1^*, obtaining

$$T^{\text{Class1}}_{\text{Lin, GenColl}}(b_1^*, C, N_1) = \frac{\mathcal{F}}{b_1^*} \cdot \frac{(C-2) + \sqrt{(C-2)^2 + 8N_1C}}{2C} \ . \tag{2.7}$$

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 $\mathcal{F}/Cb_1^*$, so the number of fast peers in the first fast chain is $\frac{t_{\text{class1}} - \mathcal{F}/b_1^*}{\mathcal{F}/Cb_1^*} + 1$. For each fast peer a new slow chain is started. The number of slow peers in a slow chain at time t is $1 + \frac{t - t_{\text{start}} - \mathcal{F}/b_2}{\mathcal{F}/Cb_2}$, 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

$$\sum_{j=1}^{\frac{t_{\text{class1}} - \mathcal{F}/b_1^*}{\mathcal{F}/Cb_1^*}+1} \max\left(0, \left\lfloor 1 + \frac{t - j\frac{\mathcal{F}}{Cb_1^*} - \frac{\mathcal{F}}{b_2}}{\mathcal{F}/Cb_2} \right\rfloor\right) \tag{2.8}$$

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

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

$$\sum_{j=1}^{\frac{t_{\text{class1}} - 2\mathcal{F}/b_1^*}{\mathcal{F}/Cb_1^*}+1} \max\left(0, \left\lfloor 1 + \frac{t - \left(j\frac{\mathcal{F}}{Cb_1^*} + \frac{\mathcal{F}}{b_1^*}\right) - \frac{\mathcal{F}}{b_2}}{\mathcal{F}/Cb_2} \right\rfloor\right) \ . \tag{2.9}$$

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

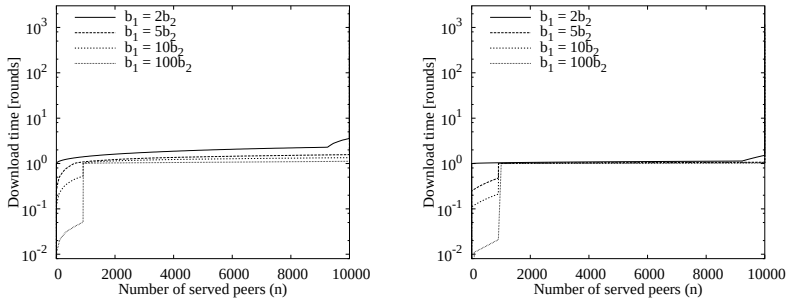
$$n_2(t) = \sum_{k=0}^{\frac{t_{\text{class1}}}{\mathcal{F}/b_1^*}} \sum_{j=1}^{\frac{t_{\text{class1}} - (k+1)\mathcal{F}/b_1^*}{\mathcal{F}/Cb_1^*}+1} max\left(0, \left\lfloor 1 + \frac{t - \left(j\frac{\mathcal{F}}{Cb_1^*} + k\frac{\mathcal{F}}{b_1^*}\right) - \frac{\mathcal{F}}{b_2}}{\mathcal{F}/Cb_2} \right\rfloor\right) \ . \tag{2.10}$$

From this relation, it is possible to find

$$T^{\text{Class2}}_{\text{Lin, GenColl}}(b_2, C, N_2) \qquad \text{such that} \qquad n_2(T^{\text{Class2}}_{\text{Lin, GenColl}}) = N_2 \tag{2.11}$$

It is important to note that equation (2.10) 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 are a lot of slow chain started (a slow chain for every fast peer). The additional chains started by the server can only be negligible for any interesting number N_1, N_2.

Figure 2.8 shows the total time against the number of peer n: in this case $N_2 = 10N_1$, where $N_1 + N_2 = N = 10^4$. With collaboration, with few fast peers it is possible to distribute the content to a lot of slow peers.



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

Figure 2.8: Linear architecture: system evolution with Generous with Collaboration scheme ($N_2 = 10N_1$, $N = 10^4$)

Average download time. For the average download time we have to find the sum of the delays experimented by the N peers. For class 1 the delays can be found with the approximation that in both the cases $t_{\text{class1}} < t_{\text{class2}}$ and $t_{\text{class1}} > t_{\text{class2}}$ the bandwidth used by fast peer chains is b_1^* and then apply (2.3).

For class 2, consider the slow chains born from the first fast chain: the first peer of the first slow chain finishes at $\frac{\mathcal{F}}{Cb_1^*} + \frac{\mathcal{F}}{b_2}$, the second at $\frac{\mathcal{F}}{Cb_1^*} + \frac{\mathcal{F}}{b_2} + \frac{\mathcal{F}}{Cb_2}$, the third at $\frac{\mathcal{F}}{Cb_1^*} + \frac{\mathcal{F}}{b_2} + 2\frac{\mathcal{F}}{Cb_2}$; the first peer of the second slow chain finishes at $2\frac{\mathcal{F}}{Cb_1^*} + \frac{\mathcal{F}}{b_2}$, the second at $2\frac{\mathcal{F}}{Cb_1^*} + \frac{\mathcal{F}}{b_2} + \frac{\mathcal{F}}{Cb_2}$, the third at $2\frac{\mathcal{F}}{Cb_1^*} + \frac{\mathcal{F}}{b_2} + 2\frac{\mathcal{F}}{Cb_2}$; generalizing for all the $\frac{t_{\text{class1}} - \mathcal{F}/b_1^*}{\mathcal{F}/Cb_1^*} + 1$ slow chains derived from the first fast chain we obtain:

$$\sum_{k=1}^{\frac{t_{\text{class1}} - \mathcal{F}/b_1^*}{\mathcal{F}/Cb_1^*}+1} \; \sum_{j=0}^{\frac{t - k\frac{\mathcal{F}}{Cb_1^*} - \frac{\mathcal{F}}{b_2}}{\mathcal{F}/Cb_2}} \left(k\frac{\mathcal{F}}{Cb_1^*} + \frac{\mathcal{F}}{b_2} + j\frac{\mathcal{F}}{Cb_2} \right) \tag{2.12}$$

where t is the instant that we choose to observe the system. If we jointly

consider also fast peer chains we obtain:

$$D_2 = \sum_{l=0}^{\frac{t_{\text{class1}}}{\mathcal{F}/b_1^*}} \; \sum_{k=1}^{\frac{t_{\text{class1}} - (l+1)\frac{\mathcal{F}}{b_1^*}}{\mathcal{F}/Cb_1^*}+1} \; \sum_{j=0}^{\frac{t - l\frac{\mathcal{F}}{b_1^*} - k\frac{\mathcal{F}}{Cb_1^*} - \frac{\mathcal{F}}{b_2}}{\mathcal{F}/Cb_2}} \left(l\frac{\mathcal{F}}{b_1^*} + k\frac{\mathcal{F}}{Cb_1^*} + \frac{\mathcal{F}}{b_2} + j\frac{\mathcal{F}}{Cb_2} \right) . \tag{2.13}$$

The result of this formula is cumbersome but easy to evaluate via software implementation.

Amount of work. The number of copies distributed by the server is equal to the number of started fast chains. Since slow peers upload to others, even if fast peers complete before slow peers, it is not necessary that the server continue to upload to slow peers.

Each fast peer uploads at most twice, whereas each slow peer uploads at most once (the last peer of each chain does not upload, and, since here the number of chains is high, this term becomes significant).

2.3.4 Altruistic

In this configuration each fast peer, after uploading the content to a fast peer, stays on-line and serves slow peers; in this case, since the capacity is big, parallel upload is necessary to exploit all the fast peers upload capacity. This means that b_1/b_2 slow peers start to download from a single fast peer and finish to download after $\mathcal{F}/b_2$ rounds. For simplicity, we suppose no collaboration of slow peers, i.e., they do not start new chains. We suppose that fast peers are able to serve all the slow peers (i.e., $\frac{b_1}{b_2}N_1 > N_2$), so they upload only once. Figure 2.9 shows the chunk distribution methodology in this case.

Total download time. Considering an instant t, the number of slow peers that has completed is equal to the number of fast peer that have completed

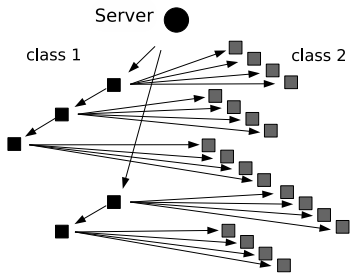


Figure 2.9: Chunk distribution with altruistic fast peers

at time $t - \mathcal{F}/b_2$ 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.,

$$T^{\text{Class1}}_{\text{Lin, Altr}}(b_1, C, N_1) = \frac{\mathcal{F}}{b_1} \cdot \frac{(C-2) + \sqrt{(C-2)^2 + 8N_1C}}{2C} \ . \tag{2.14}$$

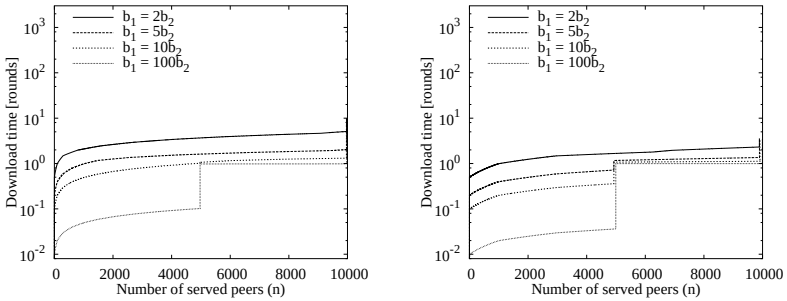
The total time for class 2 to complete is

$$T^{\text{Class2}}_{\text{Lin, Altr}}(b_2, C, N_2) = \tfrac{\mathcal{F}}{b_2} + T^{\text{Class1}}_{\text{Lin, Altr}}(b_1, C, \tfrac{b_2}{b_1}N_2) \quad \text{given that} \quad N_1 > \tfrac{b_2}{b_1}N_2 \tag{2.15}$$

where the last term is the time necessary to reach a number of fast peers that is able to serve all the slow peers (since every fast peer serves $\frac{b_1}{b_2}$ slow peers, $\frac{b_2}{b_1}N_2$ fast peer are needed).

Figure 2.10 shows the total time against the number of peer n, with $N_1 = N_2 = N/2$, where $N = 10^4$.

Average download time. For class 1 we can use (2.3). For class 2, following the steps made to find (2.3), considering that each fast peer uploads the file to b_1/b_2 slow peers and they finish with a delay of $\mathcal{F}/b_2$ with respect



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

Figure 2.10: Linear chain architecture: system evolution with Altruistic scheme ($N_1 = N_2$, $N = 10^4$)

to the fast peer, the mean download time is

$$D_2 = \sum_{k=1}^{\frac{t_{\text{class2}} - \mathcal{F}/b_2}{\mathcal{F}/b_1}} \ \sum_{j=0}^{\frac{t_{\text{class2}} - \mathcal{F}/b_2 - k\mathcal{F}/b_1}{\mathcal{F}/Cb_1}} \frac{b_1}{b_2}\left(k\frac{\mathcal{F}}{b_1} + j\frac{\mathcal{F}}{Cb_1} + \frac{\mathcal{F}}{b_2}\right) \ . \tag{2.16}$$

The Altruistic case is the most interesting case because it removes the hypothesis that peers leave the system as soon as they complete, so they can help slow peers. Of course this comes at a cost: each fast peer replicates the file b_1/b_2 times. In a specific context, this could be not acceptable (private users); in other context (file distribution in a company) this is not a problem.

Amount of work. The number of copies distributed by the server is equal to the number of started fast chains. As regards peers, each fast peer uploads $1 + \frac{N_2}{N_1}$ times, whereas slow peers do not upload.

2.3.5 Comparative Analysis for Linear Architecture

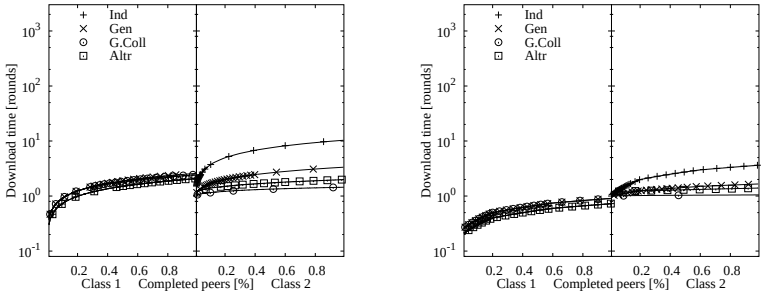
As numerical example we consider a set of bandwidth ratios, with two possible numbers of slow peers with respect to fast peers. In particular we focus on a total number of peers equal to 10^4: the analyzed cases correspond to (i) $N_2 = N_1 = 5000$ and (ii) $N_1 \simeq 900$ and $N_2 = 10N_1 \simeq 9000$. We suppose a number of chunks equal to 10^2 and 10^3. We consider the file size and the bandwidth b_2 such that $\mathcal{F}/b_2 = 1$ round and the bandwidth b_1 such that $\mathcal{F}/b_1 = 0.5,\ 0.2,\ 0.1$ and 0.01. The last case corresponds to $b_1 = 100b_2$ and can be considered as a limit to which the system evolves: with such a high bandwidth, in fact, fast peers complete the download very quickly and can help, with high capacity, low peers. The inferior limit that can be reached is a download time that tends to zero for fast peers and a download time that tends to 1 for slow peers (by construction $\mathcal{F}/b_2 = 1$).

Total Download Time

Figures 2.11 and 2.12 show how the system evolves in case of Linear architecture[4]. Each plot shows the percentage of completed peers for each class with the four different schemes (Independent, Generous, ...). Figure 2.11 shows the case $b_1 = 5b_2$, with $N_1 = N_2$ and different values of C, whereas Fig 2.12 shows the case of $b_1 = 10b_2$, with $N_2 = 10N_1$ and different values of C. It is possible to see that a helping policy greatly improves the class 2 performance, especially when $N_2 = 10N_1$, and the Generous with Collaboration reaches near optimal performance, regardless of bandwidths ratio and number of peers ratio. In particular, in Fig. 2.11, 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 $\mathcal{F}/Cb_1$, whereas only one fast chain every $\mathcal{F}/b_1$ is created). Each of these slow chains, after the whole

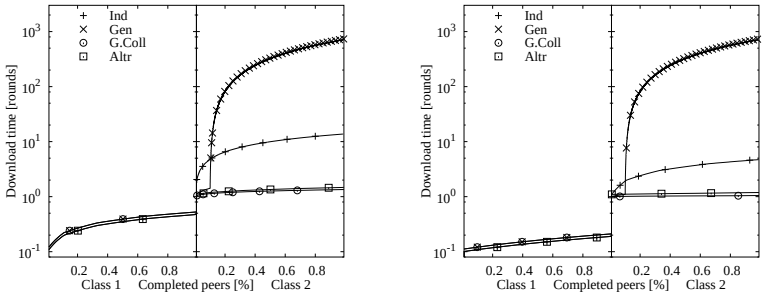
[4]Note that plots show each line with different point types: the shown points do not represent all the samples, but only a subset necessary to distinguish among different lines

file is uploaded, adds a new slow peer every $\mathcal{F}/Cb_2$, so in few slots of time (each slot is $\mathcal{F}/Cb_2$) a lot of slow peers are reached.



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

Figure 2.11: System evolution with Linear architecture ($b_1 = 5b_2$, $N_1 = N_2$, $N = 10^4$)



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

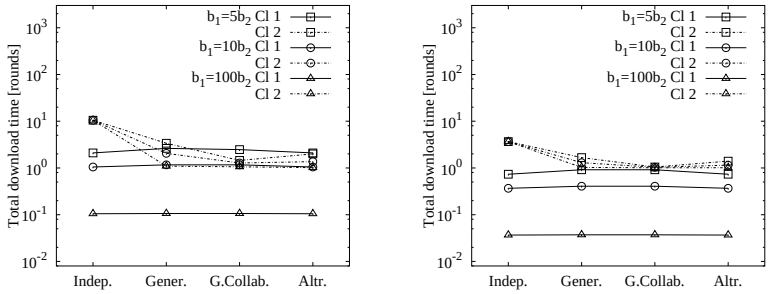
Figure 2.12: System evolution with Linear architecture ($b_1 = 10b_2$, $N_2 = 10N_1$, $N = 10^4$)

In Fig 2.12 the download time with the Generous scheme grows rapidly since slow peers do not upload and, when fast peers have completed the download, only the server can upload to slow peers.

Figures 2.13 and 2.14 shows the complete set of results for $N_1 = N_2$ and $N_2 = 10N_1$ respectively, with different values of C. For each scheme

(reported on x-axis) the total download time of each class with specific bandwidth ratios is shown.

In both cases, $N_1 = N_2$ and $N_2 = 10N_1$, for class 1, there are no big differences among used schemes, so the impact of a helping policy on results is negligible. For class 2 the Generous with Collaboration scheme performs near optimally; also Altruistic scheme obtains good results, but with higher bandwidth ratios.

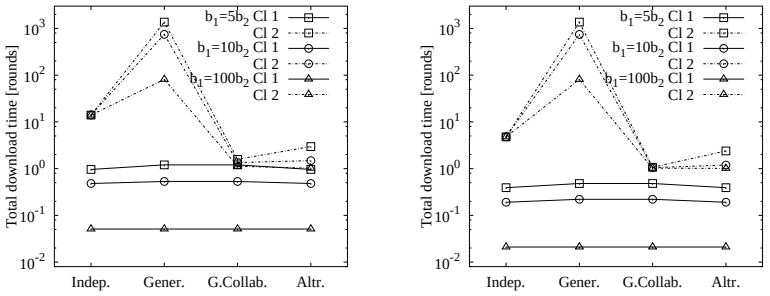


(a) $C = 10^2$ (b) $C = 10^3$

Figure 2.13: Total download time with Linear architecture achieved by each class with different schemes ($N_1 = N_2$, $N = 10^4$)

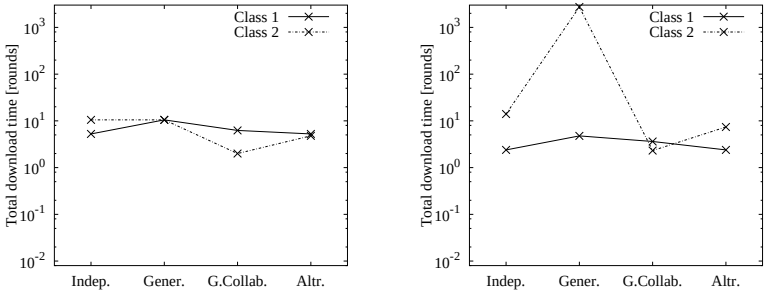
The case with $b_1 = 2b_2$ deserves a separated analysis. In this case the bandwidth for class 1 becomes $b_1^* = b_1 - b_2 = b_2$ and class 2 may complete the download before class 1 (Fig. 2.15). For this reason, helping schemes should be used only if $b_1 > 2b_2$. Nevertheless, when $b_1 = 2b_2$, the Generous with Collaboration scheme, particularly with $N_2 = 10N_1$, can be used to improve the performance of class 2.

Concluding the analysis of the total download time with Linear architecture, for fast peers any helping policy can be adopted without significantly affecting the class performance, provided that $b_1 > 2b_2$. For slow peers, the Generous with Collaboration scheme works best in most of the cases



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

Figure 2.14: Total download time with Linear architecture achieved by each class with different schemes ($N_2 = 10N_1$, $N = 10^4$)



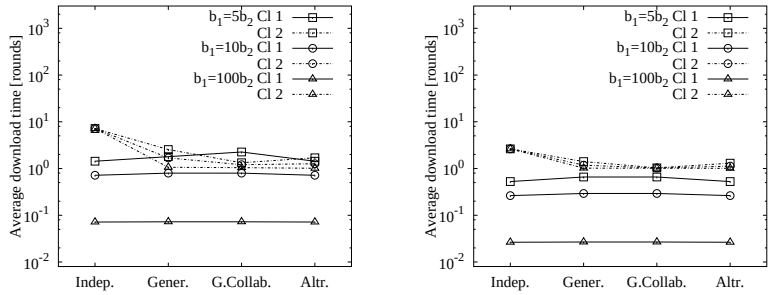
(a) $N_1 = N_2$ (b) $N_2 = 10N_1$

Figure 2.15: Total download time with Linear architecture in case of $b_1 = 2b_2$ ($N = 10^4$, $C = 10^2$)

we considered. Note that this scheme is robust to the different values of $\frac{N_2}{N_1}$, so it is applicable in scenarios where this ratio is not known a priori.

Average Download Time

Figure 2.16 shows the average download time with Linear architecture, when $N_1 = N_2$, with different values of C. The behavior with the helping schemes is similar to that we found with total download time, so the conclusions we have drawn before still hold.



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

Figure 2.16: Average Download Time with Linear architectures using different schemes ($N_1 = N_2$, $N = 10^4$)

Amount of Work

Table 2.1 summarizes the amount of work for each scheme. Linear architecture maintains the amount of work of both classes low. The only exception is class 1 work with Altruistic scheme. Nevertheless, the Generous with Collaboration scheme, that obtains the best total download time in most of the cases, has a little impact on amount of work done by each class.

Table 2.1: Amount of work with Linear architecture

Schemes	Server	Class 1	Class 2
Independent	$\sum_i \frac{t_{\text{class i}}}{\mathcal{F}/b_i}$	1	1
Generous	$\frac{t_{\text{class 1}}}{\mathcal{F}/b_1} + \max(0, N_1 + N_2)$	2	0
Gen. Collab.	$\frac{t_{\text{class 1}}}{\mathcal{F}/b_1}$	≤ 2	≤ 1
Altruistic	$\frac{t_{\text{class 1}}}{\mathcal{F}/b_1}$	$1 + N_2/N_1$	0

2.4 Tree

With Tree architecture it is possible to have parallel uploads. In the general case, it is possible to specify different outdegrees: k, the outdegree of a fast peer toward fast peers; f, the outdegree of a fast peer toward slow peers; s the outdegree of a slow peer toward slow peers. Thanks to these design parameters, it is possible to organize the peers such that they all finish the download at the same time: this optimizes the total download time that is our main performance index.

As in the Linear architecture, it is possible to identify different cases. In the following paragraphs we analyze them.

2.4.1 Independent

Total download time. Each class builds its own tree and evolves independently; no chunks are exchanged between the two trees. The total download time is the time necessary to the first chunk to reach the leaves plus the time to upload the file to the leaves using a bandwidth b_i/k for each upload[5]. The number of levels l in a tree with N_i nodes[6] can be found considering that the first level contains k nodes (the level 0 is the server itself), the second k^2 nodes and so forth; so $N_i = \sum_{j=1}^{l} k^j = k\frac{k^l-1}{k-1}$ and $l = log_k\left(N_i\frac{k-1}{k} + 1\right)$.

[5]For simplicity, we use the notation with k for both fast and slow trees.

[6]Without loss of generality, for notation simplicity we consider a full tree.

A single chunk reaches the leaves at time $\frac{\mathcal{F}/C}{b_i/k}l = \frac{\mathcal{F}}{b_i}\frac{k}{C}log_k\left(N_i\frac{k-1}{k}+1\right)$, so the total download time is

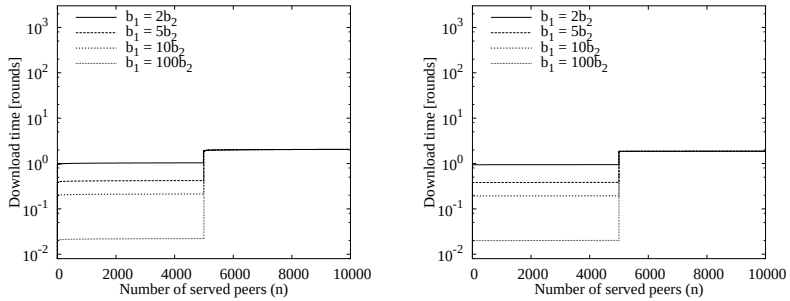
$$T^{\text{Class i}}_{\text{Tree, Lin}}(b_i, C, k, N_i) = \frac{\mathcal{F}}{b_i}k + \frac{\mathcal{F}}{b_i}\frac{k}{C}\left(log_k\left(N_i\frac{k-1}{k}+1\right)-1\right) \qquad (2.17)$$

where 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. If class 2 is considered, the parameter k has to be substituted by the parameter s. Since there are no chunk exchanges between trees, the parameter f is zero.

For the optimization problem of the outdegrees k and s, with this scheme it is not possible to adjust the parameters in order to obtain the same download time for the two classes: in fact, since they are independent, the modification of download time of one class does not influence the performance of the other class. This means that in this case we have to optimize the parameters independently, with the methodology proposed in [18]; then the optimal values are $k = s = 2$.

Figure 2.17 shows the total time against the number of peer n with two classes with the same number of peers ($N_1 = N_2 = N/2$, where $N = 10^4$), and different bandwidth ratios. Differently for Linear architecture, the influence of the number of chunks C on the total download time is negligible, since C appears only in the logarithmic term of (2.17) that is much smaller than the linear term.

Average download time. We consider the sum of the times necessary to complete the download of each peer and then we divide it by the number of peers. At time $\frac{\mathcal{F}}{b_i/k}$ the first k peers complete; at time $\frac{\mathcal{F}}{b_i/k}+\frac{\mathcal{F}}{Cb_i/k}$ other k^2 peers complete; at time $\frac{\mathcal{F}}{b_i/k}+2\frac{\mathcal{F}}{Cb_i/k}$ other k^3 peers complete. Generalizing



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

Figure 2.17: Tree architecture with independent classes: time necessary to complete the download ($N_1 = N_2 = N/2$, $N = 10^4$)

we have

$$D_i = \sum_{j=0}^{log_k(N_i(1-1/k)+1)} k^{j+1}\left(\frac{\mathcal{F}k}{b_i} + j\frac{\mathcal{F}k}{Cb_i}\right) . \qquad (2.18)$$

The above formula is valid for class 1; for class 2 instead of k the outdegree s has to be considered. The average download time is D_i/N_i for each class and $(D_1 + D_2)/(N_1 + N_2)$ globally.

Amount of work. The number of copies distributed by the server is equal to the number of started roots for each class, i.e., $k + s$. As regards peers, each interior node uploads k (fast peers) or s (slow peers) times, whereas each leaf node does not upload.

2.4.2 Generous

Slow peers do not upload any chunk, they only download from fast peers. Fast peers evolve according to a Tree architecture with outdegree k and capacity $b_1^* = b_1 - fb_2$; at the same time, they upload the chunks in parallel to f slow peers with bandwidth b_2 each. This means that, once the leaves

of the tree formed by fast peers are reached, the time necessary to fast peers to complete is $k\frac{\mathcal{F}}{b_1^*}$ rounds, while the time needed for slow peers to finish is equal to $\frac{\mathcal{F}}{b_2}$, without the factor k that slows down the performance.

For the optimization problem, we have to find the best value for k and f (in this case $s = 0$). The optimum value k for the tree built by fast peer is independent from the fact that part of the bandwidth is dedicated to slow peers, so $k = 2$. The optimum value f must be chosen considering that we minimize the total download time if all the peers finish as much as possible together: to do so, all the N_1 generous fast peer must upload to all the N_2 slow peers, so $f = \left\lceil \frac{N_2}{N_1} \right\rceil$. Nevertheless, if f is sufficiently big, the bandwidth for class 1 ($b_1^* = b_1 - fb_2$) could be smaller than b_2 (or, in the worst case, negative) and the total download time would be greater than in the Independent case: this result is due to the non-collaborative behavior of class 2. We can conclude that $b_1 - fb_2$ should be greater than b_2, i.e. $f < \frac{b_1}{b_2} - 1$ or equivalently $b_1 > (\frac{N_2}{N_1} + 1)b_2$.

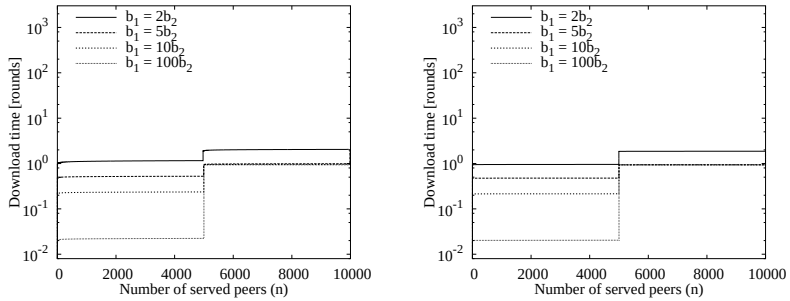
Total download time. Class 1 evolves with bandwidth $b_1^* = b_1 - fb_2$. The total time for this class can be found simply by substituting the value of the bandwidth in (2.17), obtaining

$$T^{\text{Class1}}_{\text{Tree, Gen}}(b_1^*, C, k, N_1) = \frac{\mathcal{F}}{b_1^*}k + \frac{\mathcal{F}}{b_1^*}\frac{k}{C}\left(log_k\left(N_1\frac{k-1}{k} + 1\right) - 1\right) \quad \text{given that } b_1 > \left(\frac{N_2}{N_1} + 1\right)b_2 \ . \tag{2.19}$$

For class 2, the download completes after $\mathcal{F}/b_2$ rounds the first chunk of class 1 has been completely uploaded to the leaves of the fast tree, so

$$T^{\text{Class2}}_{\text{Tree, Gen}}(b_2, C, k, N_2) = \frac{\mathcal{F}}{b_1^*}\frac{k}{C}log_k\left(N_1\frac{k-1}{k} + 1\right) + \frac{\mathcal{F}}{b_2} \quad \text{given that } b_1 > \left(\frac{N_2}{N_1} + 1\right)b_2 \ . \tag{2.20}$$

Figure 2.18 shows the total time against the number of peer n when $N_1 = N_2$.



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

Figure 2.18: Tree architecture: system evolution with Generous scheme ($N_1 = N_2$, $N = 10^4$)

Average download time. The metric can be found with (2.18). For class 1 we substitute $b_i = b_1^*$, finding D_1. For class 2, the average is equal to the average of class 1 added with a term $\frac{\mathcal{F}}{b_2} + \frac{\mathcal{F}}{b_1^*}\frac{k}{C} - \frac{\mathcal{F}}{b_1^*}k$ that is the delay that each slow peer experiments with respect to a peer of class 1.

Amount of work. The number of copies distributed by the server is equal to the number of fast roots, i.e., k. As regards peers, each fast peer uploads at most $k + \frac{N_2}{N_1}$ times, whereas slow peers do not upload. In particular, interior nodes of fast peer upload in parallel to k fast peers and to $\frac{N_2}{N_1}$ slow peers and leaf nodes upload to $\frac{N_2}{N_1}$ slow peers.

2.4.3 Leaves Only Generous

With Tree architecture leaf nodes of fast tree do not upload to other fast peers, but only to slow peers. The unused capacity of leaf nodes (b_1) can be exploited using an *alternative* Generous scheme. We can force interior nodes of a fast tree to upload only to other k fast nodes, whereas leaf nodes, as soon as they receive the first chunk, upload to all the slow peers.

The number of leaves in a full tree is k^l, i.e., $\frac{k-1}{k}N_1$. Each leaf node of the fast tree has to upload to $\frac{k}{k-1}\frac{N_2}{N_1}$ slow peers with bandwidth b_2. This means that this alternative scheme is not applicable with good performance if $b_1 < \frac{k}{k-1}\frac{N_2}{N_1}b_2$. With respect the previous Generous scheme, this constraint imposes higher capacity b_1 when the ratio between the number of peers of each class grows. If the constraint is satisfied, the total download time for class 1 is equal to the total download time in the Independent case, since interior nodes can use all the capacity. So

$$T^{\text{Class1}}_{\text{Tree, LeavGen}}(b_1, C, k, N_1) = \frac{\mathcal{F}}{b_1}k + \frac{\mathcal{F}}{b_1}\frac{k}{C}\left(log_k\left(N_1\frac{k-1}{k}+1\right)-1\right) \quad \text{given that } b_1 > \frac{k}{k-1}\frac{N_2}{N_1}b_2\ . \tag{2.21}$$

For class 2, it is sufficient to substitute b_1 to b_1^* in (2.19), obtaining

$$T^{\text{Class2}}_{\text{Tree, LeavGen}}(b_2, C, k, N_2) = \frac{\mathcal{F}}{b_1}\frac{k}{C}log_k\left(N_1\frac{k-1}{k}+1\right)+\frac{\mathcal{F}}{b_2} \quad \text{given that } b_1 > \frac{k}{k-1}\frac{N_2}{N_1}b_2\ . \tag{2.22}$$

This scheme could be interesting if the constraint $b_1 > \frac{k^2}{k-1}\frac{N_2}{N_1}b_2$ is not satisfied and class 2 collaborates in distributing the content: in this case, leaf nodes upload to as many as possible slow peers and each served slow peer starts a chain as soon as it receives the first chunk. Each leaf node serves b_1/b_2 slow peers, so the total number of chains is (at least) $\frac{k-1}{k^2}N_1\frac{b_1}{b_2}$ and the content is then rapidly distributed to all the slow peers.

Average download time. The metric can be found with (2.18). For class 1 we substitute $b_i = b_1$, finding D_1. For class 2, the average is simple the total download time, since all the slow peers start and finish together [7].

Amount of work. Interior nodes upload k times, whereas leaf nodes upload up to $\frac{k}{k-1}\frac{N_2}{N_1}$ times.

[7] Even if leaf nodes in the fast tree can stay in different levels, the difference in terms of delay is not significant, provided that $C \gg 1$.

2.4.4 Generous, with Collaboration

Total download time. With this scheme, we try to exploit the unused capacity of the slow peers already served: once a slow peer has received a chunk, it starts to redistribute it. The optimal outdegree s of slow peer trees is 1: in fact, from (2.17), the time necessary to complete a tree is at least equal to $\frac{\mathcal{F}}{b_2}s$ and a value of s greater than 1 implies that collaboration would not have effect, as it becomes evident through the comparison between (2.17), with $s > 1$, and (2.20)[8]. Figure 2.19 shows an example of the scheme.

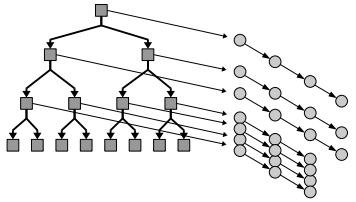


Figure 2.19: Tree architecture: tree building in case of k=2, s=1 and f=1

It is important to note that introducing collaboration influences class 1 performance, but has only a little impact on class 2 results. In fact collaborative behavior diminishes the parameter f and the capacity b_1^* becomes greater; nevertheless the capacity b_1^* is present only in the logarithmic term of (2.20) that gives the smallest contribution to the total time.

In order to reduce the number of parameters, we can choose k considering that fast trees evolve independently (so $k = 2$), having only the rate of creation of slow chains (the parameter f) to find. We start from the maximum value of f, i.e., $f = N_2/N_1$ and we calculate the total download time in this case (that is the Generous case, without collaboration, since fast peers help all the slow peers); we then iteratively reduce the value of

[8] If $b_1 \simeq b_2$ and $log_k(N_1) \gg log_s(N_2)$ this could not be true, but these degenerate cases are not considered.

f calculating if, with the collaboration of slow peers, it is possible to reach all the slow peers in lower time (with respect to the previous iteration).

Once the value f is found, class 1 total download time is given by (2.17) with $b_i = b_1^* = b_1 - fb_2$, so

$$T^{\text{Class1}}_{\text{Tree, GenColl}}(b_1^*, C, k, N_1) = \frac{\mathcal{F}}{b_1^*}k + \frac{\mathcal{F}}{b_1^*}\frac{k}{C}\left(log_k\left(N_1\frac{k-1}{k}+1\right)-1\right) \quad \text{given that } b_1 > \left(\frac{N_2}{N_1}+1\right)b_2 \ . \tag{2.23}$$

Even if slow peers collaborate in distributing the content, all the fast peers upload the content to a subset of slow peers (each fast peer uploads to f slow peers), so the bandwidth b_1^* does not change during the download. As concerns class 2, we have to analyze the behavior in detail. At the beginning, the first k fast peers finish to upload the first chunk to the $k \cdot f$ slow peers at time $\frac{k\mathcal{F}}{Cb_1^*} + \frac{\mathcal{F}}{Cb_2}$ (the first term is the time necessary to download the first chunk and the second term is the time necessary to upload it with bandwidth b_2). Therefore the $k \cdot f$ slow peers become heads of Linear chains that distribute the content with bandwidth b_2. The number of peers in these chains at time t is $kf(1 + \frac{t-(k\mathcal{F}/Cb_1^*+\mathcal{F}/b_2)}{\mathcal{F}/Cb_2})$. In the meanwhile the second level of the fast tree uploads the content to new slow peers: we obtain $k^2 f$ new slow peers. The number of slow peers in these chains is $k^2 f(1 + \frac{t-(2k\mathcal{F}/Cb_1^*+\mathcal{F}/b_2)}{\mathcal{F}/Cb_2})$. Generalizing we obtain the number of slow peers ($n_2(t)$) in time

$$n_2(t) = \sum_{j=1}^{log_k(N_1(1-1/k)+1)} k^j f\left(1 + \frac{t-(jk\mathcal{F}/Cb_1^*+\mathcal{F}/b_2)}{\mathcal{F}/Cb_2}\right) \ . \tag{2.24}$$

From this relation, it is possible to find

$$T^{\text{Class2}}_{\text{Tree, GenColl}}(b_2, C, k, N_2) \qquad \text{such that} \qquad n_2(T^{\text{Class2}}_{\text{Tree, GenColl}}) = N_2 \tag{2.25}$$

The evolution of the system with this scheme is similar to the Generous case, since collaboration does not greatly improve the performance, and the results are similar to what shown in Figure 2.18.

Average download time. As regards the average download time, for class 1 we can use (2.18) with $b_1^* = b_1 - fb_2$. For class 2, we can apply the procedure used to find (2.3), deriving

$$D_2 = \sum_{l=1}^{log_k(N_i(1-1/k)+1)} k^l f \sum_{j=0}^{\frac{t_{\text{class2}}-(lk\mathcal{F}/Cb_1^*+\mathcal{F}/b_2)}{\mathcal{F}/Cb_2}} \left(l\frac{k\mathcal{F}}{Cb_i^*} + \frac{\mathcal{F}}{b_2} + j\frac{\mathcal{F}}{Cb_2}\right) \tag{2.26}$$

where t_{class2} is the time necessary for class 2 to complete.

Amount of work. The number of copies distributed by the server is equal to the number of fast roots, i.e., k. Each fast peer uploads at most $k + \frac{N_2}{N_1}$ times, whereas each slow peer uploads at most once.

2.4.5 Altruistic

As in the Linear case, we remove the hypothesis that peers leave the system as soon as they complete. Each fast peer, after uploading the content to k fast peers, stays on-line and starts to serve slow peers; we consider that the number of slow peers is smaller than $\frac{b_1}{b_2}N_1$ and slow peers do not collaborate (these hypotheses simplify the calculation, without loss of generality).

Total download time. Class 1 evolve independently, so

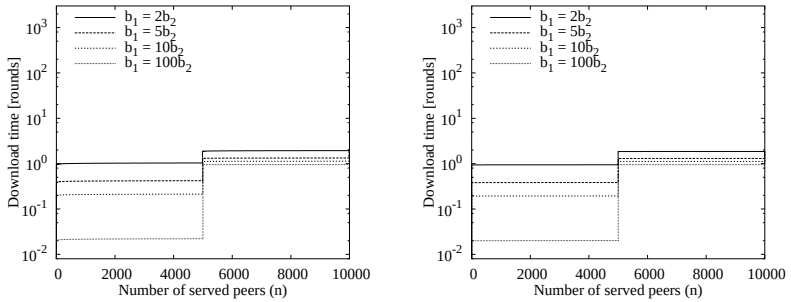
$$T^{\text{Class1}}_{\text{Tree, Altr}}(b_1, C, k, N_1) = \frac{\mathcal{F}}{b_1}k + \frac{\mathcal{F}}{b_1}\frac{k}{C}\left(log_k\left(N_1\frac{k-1}{k}+1\right)-1\right) \quad \text{given that } N_2 < \frac{b_1}{b_2}N_1 \ . \tag{2.27}$$

It is straightforward to see that the time necessary for class 2 to complete is

$$T^{\text{Class2}}_{\text{Tree, Altr}}(b_2, C, k, N_2) = \frac{\mathcal{F}}{b_2} + T^{\text{Class1}}_{\text{Tree, Altr}}(b_1, C, k, N_2\frac{b_2}{b_1}) \quad \text{given that } N_2 < \frac{b_1}{b_2}N_1 \tag{2.28}$$

where the last term is the time necessary to reach a number of fast peers that is able to serve all the slow peers (since every fast peer serves $\frac{b_1}{b_2}$ slow peers, $N_2\frac{b_2}{b_1}$ fast peer are needed).

Figure 2.20 shows the total time against the number of peer n, with $N_1 = N_2 = N/2$, where $N = 10^4$.



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

Figure 2.20: Tree architecture: system evolution with Altruistic scheme ($N_1 = N_2$, $N = 10^4$)

Average download time. The metric can be found with (2.18). For class 1 we substitute $b_i = b_1$, finding D_1. For class 2 the average is equal to the average of a fast tree with $N_2\frac{b_2}{b_1}$ nodes plus $\frac{\mathcal{F}}{b_2}$.

Amount of work. The number of copies distributed by the server is equal to the number of fast roots, i.e., k. Each fast peer uploads at most $k + \frac{N_2}{N_1}$ times, whereas slow peers do not upload. In this case leaf nodes may not contribute, depending on the number of slow peers and on the bandwidth ratio. In fact, if the number of interior nodes is sufficient and have sufficient bandwidth to serve all the slow peers, leaf nodes may not upload.

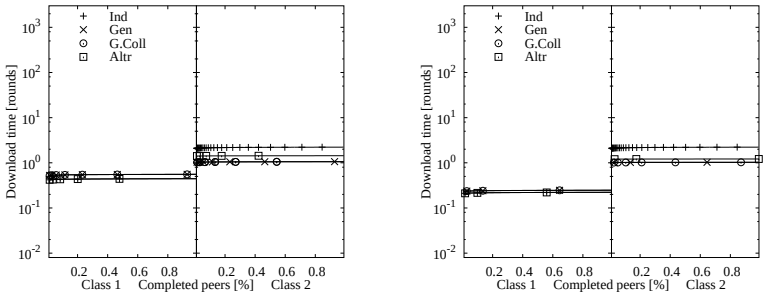
2.4.6 Comparative Analysis for Tree Architecture

As numerical example we consider the same set of number of peers and bandwidth ratios described in Sect. 2.3.5. We consider the same helping

scheme of the Linear architecture, i.e., Independent, Generous, Generous with Collaboration and Altruistic. The Leaves only Generous scheme has significant results when the difference between b_1 and b_2 is small ($b_1 = 2b_2$) and this case is treated separately. In all the other cases the results obtained with such a scheme are equal to those obtained with the Generous scheme.

Total Download Time

Figure 2.21 shows the system evolution using different schemes, with $b_1 = 5b_2$ (Fig 2.21(a)) and $b_1 = 10b_2$ (Fig 2.21(b)). Also with Tree architecture, helping schemes can improve class 2 performance without greatly affecting class 1 performance.



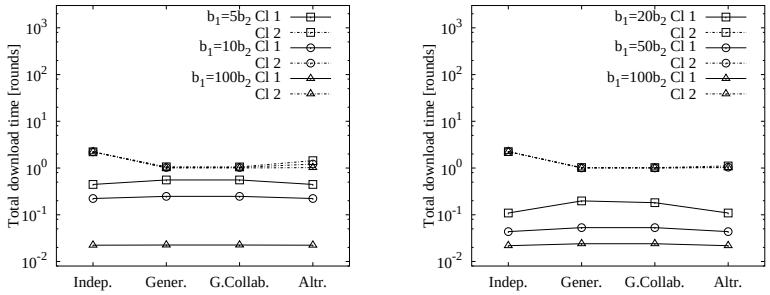
(a) $b_1 = 5b_2$ (b) $b_1 = 10b_2$

Figure 2.21: System evolution with Tree architecture ($N_1 = N_2$, $N = 10^4$, $C = 10^2$)

Figure 2.22 reports the whole set of results, for $N_1 = N_2$ and $N_2 = 10N_1$. In case of $N_2 = 10N_1$ we use a different set of bandwidth ratios, since, with Tree architecture, helping schemes can be applied only if $b_1 > \left(\frac{N_2}{N_1} + 1\right) b_2$; we use class 1 capacity 20, 50 and 100 times greater than class 2 capacity.

The Generous scheme performs optimally for both classes regardless of bandwidth ratios. The collaboration, in this case, has no influence, since

fast peers are able to help all the slow peers before the collaboration of slow peers can have effect. The Altruistic scheme obtains also good results, but slightly worse when $N_1 = N_2$, because each fast peer has to complete the download before starting to help slow peers (when $N_2 = 10N_1$ there are few fast peers and this effect is not visible).



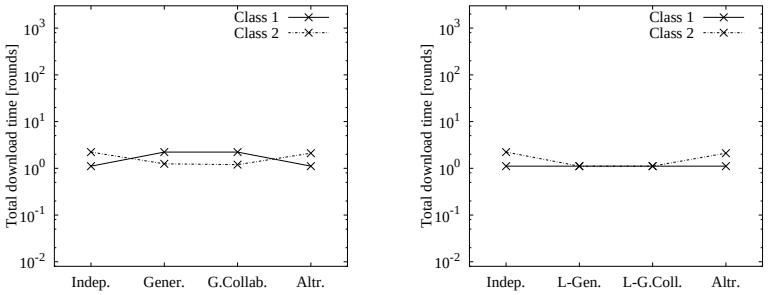
(a) $N_1 = N_2$ (b) $N_2 = 10N_1$

Figure 2.22: Total download time with Tree architecture achieved by each class with different schemes ($N = 10^4$, $C = 10^2$)

The case with $b_1 = 2b_2$, when $N_1 = N_2$, needs a separated analysis. The bandwidth for class 1 becomes $b_1^* = b_1 - b_2 = b_2$ and class 2 may complete the download before class 1 (Fig. 2.23(a)). In this situation, the Leaves only Generous scheme can be used to improve the performance of class 1. Figure 2.23(b) show the results that can be obtained when class 1 adopts a Leaf only Generous scheme.

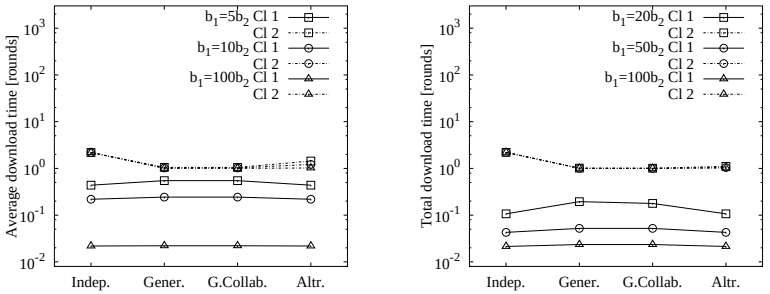
Average Download Time

Figure 2.24 shows the average download time with Tree architecture, when $N_1 = N_2$. The behavior with the different schemes is similar to that we found with total download time, so the conclusions we have drawn before still hold.



(a) Generous scheme (b) Leaves only Generous scheme

Figure 2.23: Total Download Time in case of Tree architecture, with $b_1 = 2b_2$, using different Generous schemes ($N_1 = N_2$, $N = 10^4$, $C = 10^2$)



(a) $N_1 = N_2$ (b) $N_2 = 10N_1$

Figure 2.24: Average Download Time with Tree architectures using different schemes ($N = 10^4$, $C = 10^2$)

Amount of Work

Table 2.2 summarizes the amount of work. Tree architecture presents differences among peers of the same class: in fact interior nodes upload much more than leaf nodes. It is important to note that unfairness among peers does not mean that scheme is not applicable. In fact, there are situations where peers are not able to upload (e.g., nodes behind a firewall or a NAT device). An asymmetric distribution architecture or scheme assumes a different meaning in such scenarios.

Table 2.2: Amount of work with Tree architecture

Schemes	Server	Class 1	Class 2
Independent	$k+s$	k	s
Generous	k	$k+N_2/N_1$	0
Gen. Collab.	k	$\leq (k+N_2/N_1)$	1
Altruistic	k	$k+N_2/N_1$	0

2.5 PTree

As in the Tree architecture it is possible to specify different outdegrees: k, the outdegree of a fast peer toward fast peers; f, the outdegree of a fast peer toward slow peers; s the outdegree of a slow peer toward slow peers. Note that the PTree distribution architecture is designed to exploit all the peer capacities, so the gain that is possible to achieve with a collaboration policy in negligible. This consideration is confirmed also by the analysis: the adoption of a scheme has impact only on the logarithmic term of the formula obtained in the Independent case, that is much smaller than the linear term of the same formula. In the following paragraphs we analyze the different schemes.

2.5.1 Independent

Total download time. Each class builds its own tree and evolves independently; no chunks are exchanged between the two trees. The server starts k different distribution trees and each tree contains N_i nodes. The number of levels l in a tree with N_i nodes[9] can be found considering that the first level contains 1 node, the second k nodes, the third k^2 nodes, and so forth; so $N_i = \sum_{j=0}^{l} k^j = \frac{k^{l+1}-1}{k-1}$ and $l = log_k\left(\frac{N_i(k-1)+1}{k}\right)$. The total download time is then

$$T^{\text{Class i}}_{\text{PTree, Ind}}(b_i, C, k, N_i) = \frac{\mathcal{F}}{b_i} + \frac{\mathcal{F}}{b_i}\frac{k}{C}\left(log_k\left(N_i\frac{k-1}{k}+\frac{1}{k}\right)-1\right) \qquad (2.29)$$

where 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. If class 2 is considered, the parameter k has to be substituted by the parameter s. Since there are no chunk exchanges between trees, the parameter f is zero.

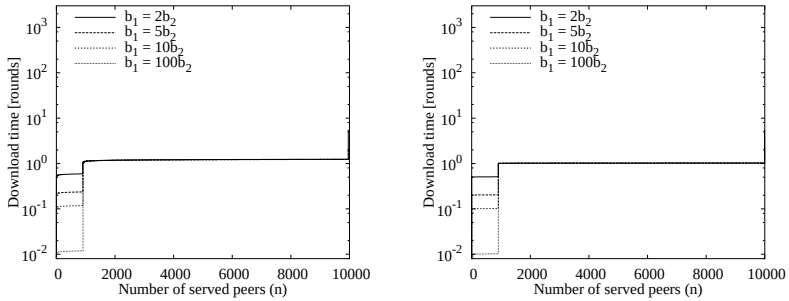
Also for this case the optimization problem is solved independently for the two classes (see [18]), leading to optimal values $k = s = 3$.

Figure 2.25 shows the total time against the number of peer n with two classes ($N_2 = 10N_1$, where $N_1 + N_2 = N = 10^4$) with different bandwidth ratios.

Average download time. The average download time, with this architecture, is trivial: since all the peers belonging to the same class finish at the same time, the average for each class is equal to the total download time and the global average is $T^{\text{mean-indep}}_{\text{PTree}} = (N_1 \cdot T^{\text{Class1}}_{\text{PTree}} + N_2 \cdot T^{\text{Class2}}_{\text{PTree}})/(N_1+N_2)$.

Amount of work. The number of copies distributed by the server is equal to the number of started roots for each class; since each root download

[9] Without loss of generality we consider a full tree.



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

Figure 2.25: PTree architecture with independent classes: time necessary to complete the download ($N_2 = 10N_1$, $N = 10^4$)

only a fraction of the file, the server uploads the whole file only twice. As regards fast and slow peers, each node uploads once.

2.5.2 Generous

Slow peers do not upload any chunk; they only download from fast peers; fast peers upload in parallel the chunk to k fast peers and to f slow peers. Here the bandwidth imposes more constraints: each fast peer, in fact, receives and distributes a fraction $1/k$ of the file $\mathcal{F}$ to k peers, while receiving in parallel the other $k-1$ fractions. The maximum bandwidth allowed toward a peer is then a fraction $1/k$ of its capacity (since the peer receives up to k different chunks in parallel). Fast peers upload to f slow peers with bandwidth b_2/k for each slow peer. The remaining bandwidth ($b_1^* = b_1 - fb_2/k$) is then used for fast peers.

Total download time. For the optimization problem, we have to find the best value for k and f (since there is no collaboration, $s = 0$). The optimum value k for the tree built by fast peers is independent from the fact that part

of the bandwidth is dedicated to slow peers, so $k = 3$. The optimum value f must be chosen considering that we minimize the total download time if all the peers finish as much as possible together: to do so, all the N_1 generous fast peer must upload to all the N_2 slow peers, so $f = \left\lceil \frac{N_2}{N_1} \right\rceil$. Again, if f is sufficiently big, the bandwidth for class 1 (b_1^*) could be smaller than b_2 or, in the worst case, less than zero, and the total download time would be greater than in the Independent case (since class 2 is non-collaborative). We can conclude that $b_1 - fb_2/k$ should be greater than b_2, i.e. $f < \left(\frac{b_1}{b_2} - 1\right) k$; this means that, if $b_1 < (\frac{1}{k}\frac{N_2}{N_1} + 1)b_2$ the scheme is not applicable with good performance.

The total download time for class 1 is given by (2.29), where $b_i = b_1^*$, obtaining

$$T^{\text{Class1}}_{\text{PTree, Gen}}(b_1^*, C, k, N_1) = \frac{\mathcal{F}}{b_1^*} + \frac{\mathcal{F}}{b_1^*}\frac{k}{C}\left(log_k\left(N_1\frac{k-1}{k} + \frac{1}{k}\right) - 1\right) \quad \text{given that } \; b_1 < \left(\frac{1}{k}\frac{N_2}{N_1} + 1\right) b_2 \; . \tag{2.30}$$

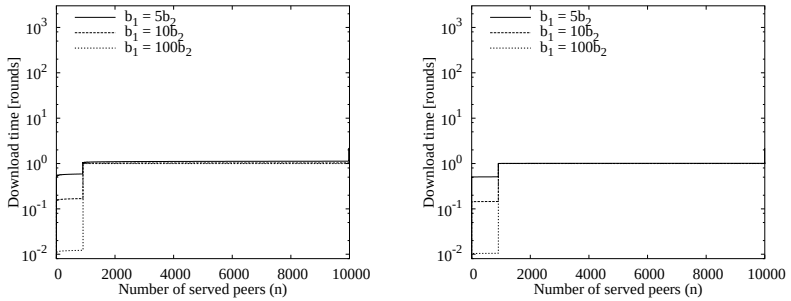
Looking at class 2, the download completes after $\mathcal{F}/b_2$ rounds class 1 has reached the leaves of the fast tree, so

$$T^{\text{Class2}}_{\text{PTree, Gen}}(b_2, C, k, N_2) = \frac{\mathcal{F}}{b_1^*}\frac{k}{C} log_k\left(N_1\frac{k-1}{k} + \frac{1}{k}\right) + \frac{\mathcal{F}}{b_2} \quad \text{given that } \; b_1 < \left(\frac{1}{k}\frac{N_2}{N_1} + 1\right) b_2 \; . \tag{2.31}$$

Figure 2.26 shows the total time against the number of peer n when $N_2 = 10N_1$, $N_1 + N_2 = N = 10^4$, with different bandwidth ratios.

Average download time. The average download time depends on the total download time for each class and is equal to $(N_1 \cdot t_{\text{class 1}} + N_2 \cdot t_{\text{class 2}})/(N_1 + N_2)$.

Amount of work. The server uploads the whole file only once to class 1. As regards peers, each fast peer uploads $1 + \frac{1}{k}\frac{N_2}{N_1}$ times, whereas slow peers do not upload at all.



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

Figure 2.26: PTree architecture: system evolution with Generous scheme ($N_2 = 10N_1$, $N = 10^4$)

2.5.3 Generous, with Collaboration

In this case, slow peers are helped by fast peer, but here each served slow peer starts a new tree with degree s. In order to reduce the number of parameters, we can choose k and s considering that trees evolve independently (so $k = s = 3$), having only the rate of creation of slow trees (the parameter f) to find. We start from the maximum value of f, i.e., $f = N_2/N_1$ and we calculate the total download time in this case (that is the Generous case, without collaboration, since fast peers help all the slow peers); we then iteratively reduce the value of f calculating if, with the collaboration of slow peers, it is possible to reach all the slow peers in lower time (with respect to the previous iteration). In particular we start from the total download time for class 1, i.e.,

$$T^{\text{Class1}}_{\text{PTree, GenColl}}(b_1^*, C, k, N_1) = \frac{\mathcal{F}}{b_1^*} + \frac{\mathcal{F}}{b_1^*}\frac{k}{C}\left(log_k\left(N_1\frac{k-1}{k} + \frac{1}{k}\right) - 1\right) \quad (2.32)$$

Starting from the maximum value of f, we can find $t^{\max}_{\text{class1}}$, maximum download time for class 1, and, with (2.31), $t^{\max}_{\text{class2}}$, maximum download time for class 2 without collaboration. If we introduce collaboration, new slow trees

have a time lapse equal to $t^{\max}_{\text{class2}}$ subtracted by the delay necessary to start the slow tree.

At the beginning, the first f slow peers have $t^{\max}_{\text{class2}} - \frac{k\mathcal{F}}{Cb_1^*}$ rounds to distribute the content. From (2.29) it is possible to find the number n' such that $T_{\text{PTree, Ind}}(b_2, C, s, n') = t^{\max}_{\text{class2}} - \frac{k\mathcal{F}}{Cb_1^*}$. In the meanwhile the second level of the fast tree uploads the content to new slow peers: we obtain kf new slow peers that have $t^{\max}_{\text{class2}} - 2\frac{k\mathcal{F}}{Cb_1^*}$ rounds to distribute the content. Again, we can find n'' such that $T_{\text{PTree, Ind}}(b_2, C, s, n'') = t^{\max}_{\text{class2}} - 2\frac{k\mathcal{F}}{Cb_1^*}$. At the end the sum of n', n'', n''', ... is the total number of slow peers that have completed the download with the Generous with Collaboration scheme. If this number is greater than N_2, we repeat the procedure decrementing f. The minimum f found with this procedure is the value we finally use in (2.32) to find the total download time of class 1. We can approximate the total download time for class 2 with

$$T^{\text{Class2}}_{\text{PTree, GenColl}}(b_2, C, k, N_2) = \frac{\mathcal{F}}{b_1^*}\frac{k}{C}log_k\left(N_1\frac{k-1}{k} + \frac{1}{k}\right) + \frac{\mathcal{F}}{b_2} \quad (2.33)$$

since, by construction, this is the time limit used in the procedure described above.

The evolution of the system with this scheme is similar to the Generous case and the results are similar to what shown in Figure 2.26.

2.5.4 Altruistic

Each fast peer, after uploading the content to k fast peers, stays on-line and starts to serve slow peers. In this case, we have to wait that all the fast peers complete the download; after that, $N_1\frac{b_1}{b_2}$ parallel copies are distributed to slow peers. We consider that this amount of copies is sufficient to complete the number of slow peers, i.e., $N_2 < \frac{b_1}{b_2}N_1$.

Total download time. Class 1 evolve independently, so

$$T^{\text{Class1}}_{\text{PTree, Altr}}(b_1, C, k, N_1) = \frac{\mathcal{F}}{b_1} + \frac{\mathcal{F}}{b_1}\frac{k}{C}\left(log_k\left(N_1\frac{k-1}{k} + \frac{1}{k}\right) - 1\right) \quad \text{given that} \quad N_2 < \frac{b_1}{b_2}N_1 \ . \tag{2.34}$$

The time necessary for class 2 to complete is

$$T^{\text{Class2}}_{\text{PTree, Altr}}(b_2, C, k, N_2) = \frac{\mathcal{F}}{b_2} + T^{\text{Class1}}_{\text{PTree, Altr}}(b_1, C, k, N_1) \quad \text{given that} \quad N_2 < \frac{b_1}{b_2}N_1 \tag{2.35}$$

In this case, before starting to serve slow peers, all the fast peers must finish: in fact, once the PTree structure is defined for the N_1 nodes, they complete to download all together. Even if a smaller number of fast peers is sufficient to serve slow peers (for instance, if $N_1 > N_2$), we have to wait for all the fast peers to complete.

The evolution of the system does not greatly change with respect to what shown in Fig 2.26, as becomes evident from the comparison of formulas (2.31) and (2.35).

Average download time. Again, the metric depends on the total download time for each class and is equal to $(N_1 \cdot T^{\text{Class1}}_{\text{PTree}} + N_2 \cdot T^{\text{Class2}}_{\text{PTree}})/(N_1 + N_2)$.

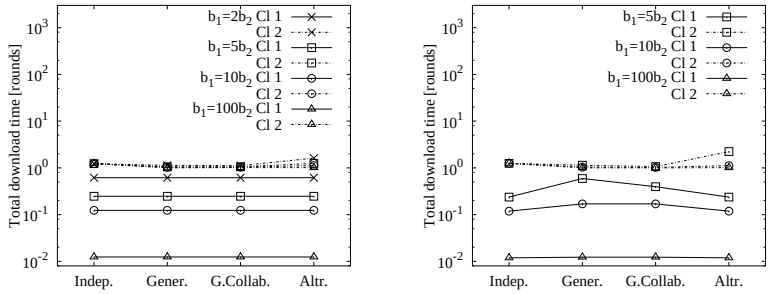
Amount of work. The server uploads the whole file only once to class 1. As regards peers, each fast peer uploads at most $1 + \frac{N_2}{N_1}$ times, whereas slow peers do not upload.

2.5.5 Comparative Analysis for PTree Architecture

As numerical example we consider the same set of number of peers and bandwidth ratios described in Sect. 2.3.5.

Total Download Time

Figure 2.27 shows the results with PTree architecture. PTree architecture imposes that helping schemes can be applied only if $b_1 > \left(\frac{1}{k}\frac{N_2}{N_1} + 1\right) b_2$. As we noted in Sect. 2.5, the different schemes do not improve the performance: in fact PTree obtains near-optimal performance for each class and a help from the other class has a little impact (only the logarithmic term is diminished, but this term is much smaller than the linear term).



(a) $N_1 = N_2$ (b) $N_2 = 10N_1$

Figure 2.27: Total Download Time with PTree architecture using different schemes ($N_1 + N_2 = N = 10^4$, $C = 10^2$)

Concluding on PTree, since this distributing architecture exploits almost completely the bandwidth resources, helping schemes simply reduce the performance of class 1 without improving the performance of class 2. With PTree architecture, then, the best scheme that can be chosen is to proceed independently.

Average Download Time

With PTree architecture the average download time is equal to the total download time, so the analysis does not change.

Amount of Work

Table 2.3 summarizes the amount of work. If we consider the Independent scheme, that obtains the best performance, the amount of work of both classes remains at the lowest possible value.

Table 2.3: Amount of work with PTree architecture

Schemes	Server	Class 1	Class 2
Independent	2	1	1
Generous	1	$1 + N_2/kN_1$	0
Gen. Collab.	1	$\leq (1 + N_2/kN_1)$	1
Altruistic	1	$1 + N_2/N_1$	0

2.6 Overall Comparative Analysis

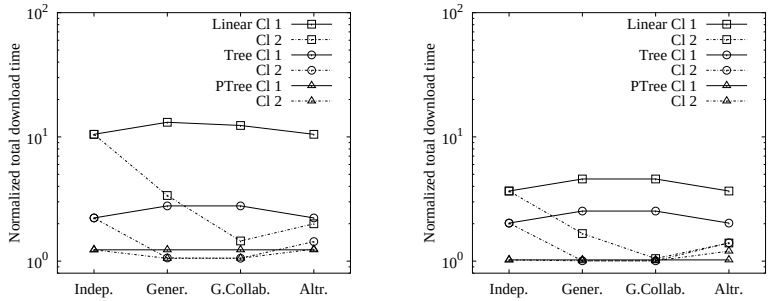
As overall comparative analysis we focus our attention on the total download time. After analyzing the single architectures, we can compare them finding some insights on the system. We normalize all the total download times using the optimal download time, i.e., the time it would take to download via unicast at rate b_1, b_2 respectively. The new metric is then

$$\text{Normalized Total Download Time} = \frac{\text{Total Download Time}}{\mathcal{F}/b_i} .$$

This means that the optimum achievable is equal to one, regardless of architecture, scheme or bandwidth ratios.

Figure 2.28 shows the results with different architectures when $b_1 = 5b_2$ and $N_1 = N_2$, for different values of C. Looking at class 1, as obtained in the analysis made in [18], the more complex the architecture is (Linear, Tree and PTree), the more the download time approaches to the optimum. The results are slightly influenced by the adopted helping scheme, since part of the bandwidth is dedicated to help slow peers.

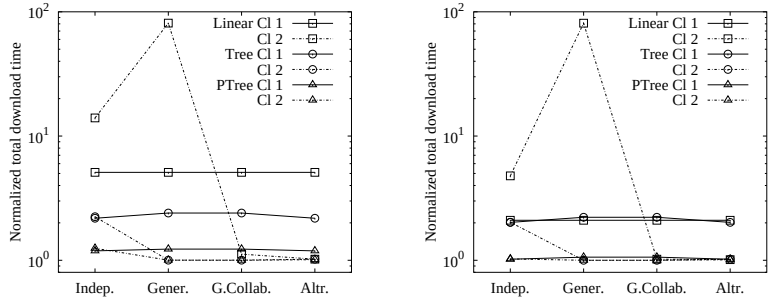
Results obtained for class 2 are completely different: a Generous scheme can greatly improve slow peer performance, obtaining, regardless of architecture, a near-optimal download time. Particularly for Linear architecture, the difference between Independent and Generous with Collaboration schemes is almost an order of magnitude.



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

Figure 2.28: Total Download Time with different architectures and different schemes ($N_1 = N_2$, $N = 10^4$, $b_1 = 5b_2$)

The same results are confirmed when using $b_1 = 100b_2$ and $N_2 = 10N_1$. Few very fast peers can help a lot of slow peers, with total download time of slow peers that approaches to the optimum. This means that a very simple distribution architecture with a simple helping scheme (download from a fast or slow peer and upload to a slow peer) can be as effective as a complex and structured distribution architecture such as PTree. It is important to note that a Linear architecture with Generous with Collaboration scheme is equivalent to a mesh topology with indegree equal to one.



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

Figure 2.29: Total Download Time with different architectures and different schemes ($N_2 = 10N_1$, $N = 10^4$, $b_1 = 100b_2$)

2.7 Lesson Learned: Conclusions on the 2-Class Analysis

Heterogeneity introduces a set of degrees of freedom in designing distribution policies even in very simple cases. The interaction between the two classes, with different bandwidths we considered, can result in global performance improvement for simpler distribution architectures and schemes. The different cases we analyzed show that, when we consider heterogeneity, slow peers may not negatively affect the system, provided that the difference in terms of capacities (and the associated constraints on number of peers) is sufficiently big (the ratio between fast peer capacity and slow peer one should be at least three-four). Collaborative distribution of a file can obtain as good performance as in the ideal distribution[10].

With the Linear architecture, provided that the bandwidth of fast peer is sufficiently big, a Generous scheme with a collaborative behavior of slow peers obtains near-optimal performance.

[10]The ideal distribution is when there is a number of available copies in the network equal to the number of peers that wants to download the content, so that the time for downloading is $\mathcal{F}/b_i$.

With the Tree architecture, with the same hypothesis and scheme, we can also have near-optimal performance. The only constraint is that the capacity must be sufficiently large to serve all the N_2 slow peers, i.e., $b_1 > \left(\frac{N_2}{N_1} + 1\right) b_2$.

PTree architecture is a scheme that reaches near optimal performances even when two classes operate independently: this means that it is not possible to find a policy that significantly improves further the results. The best trade-off between complexity and performance is obtained by simply letting classes evolve separately. In general, we can conclude that the simpler the distribution architecture is, the more improvement can be obtained with a helping scheme; the more complex the scheme is, the less the gain we have.

An important lesson learned is relative to the Altruistic scheme: even if all the fast peers upload indefinitely to slow peers, the performance that can be obtained is worse than in a collaborative scheme. This means that collaboration of slow peers is necessary in order to achieve good results, no matter how small is bandwidth.

The simplicity of Linear and Tree architectures makes them good candidates for implementation in a mesh topology. For the Linear architecture is sufficient to impose that each fast peer must upload to a fast peer and to a slow peer and each slow peer must upload to another slow peer[11]. For the Tree architecture, it is sufficient to impose that each fast peer has to upload to k fast peers in parallel and concurrently to f slow peers (f can be set by the user or estimated during the previous downloads); slow peers may or not upload to other slow peers. A problem that has to be solved is the global view of the network, i.e., peers have to upload choosing among their neighbors and not among all the peers in the network: the analy-

[11]The obvious constraints is that a peer can start to upload only to a peer that has not yet received any chunk.

sis of the performance with this limited network knowledge is provided in Chapter 4.

Chapter 3

Stochastic Analysis of Simple Distribution Architectures

The deterministic analysis, when the knowledge about neighbor characteristics is complete, shows that it is possible to define simple collaboration policies that allow to obtain near optimal results for slow peers without affecting significantly fast peers. The analysis of more than two classes becomes only a matter of cumbersome calculations, but does not increase the insight in the problem. Additionally, the hypothesis of complete knowledge is unrealistic, not to mention that there are a lot of situations where the knowledge about neighbor characteristics is not available or is changing too fast to be used. Most probably in real scenarios nodes will be randomly chosen with respect to the considered characteristic.

In this chapter we remove the hypothesis that peers are able to know the neighbor bandwidths. We propose an analytical solution of the distribution process that not only yields the mean download time but also the distribution of the download times. We validate the analytical model against simulations, which can also be used to analyze scenarios where correlation and dynamic behavior make the theoretical analysis too approximate.

3.1 Introduction

Consider the problem described in the previous chapter, i.e., distributing a file in a heterogeneous environment. We focus on a single file with size $\mathcal{F}$ that must be delivered to all peers in the network, assuming a BitTorrent-like distribution protocol [41], where the file is broken up into C pieces called "chunks," and peers that have received a chunk are able to upload it to other peers.

The time it takes to download the file to all peers depends on how the chunks propagate between peers, which is referred to as peer organization strategy, or *distribution architecture*. We consider the distribution architectures identified in [18], briefly summarized in Sect. 2.2.1, and derive a stochastic model for the download performance in presence of peers with heterogeneous bandwidth. One of the aims of the analysis is to obtain insights that can help in designing more efficient distribution protocols.

We derive an approximated analytical model that yields the stochastic distribution of the file delivery performance as a function of the distribution of the peer bandwidth. The model allows to assess the impact of slow peers on the delivery and gives enough insight in the problem to devise dynamic distribution strategies to overcome the impact of slow peers. In tree based architectures, the model yields a lower bound on performance, thus it is suited for design and dimensioning.

We develop an overlay network simulator to validate the model results and to analyze distribution architectures where the tree degree can vary dynamically based on locally available resources. In particular, we analyze the effect of bounds of the node outdegree (the tree degree local to a given node) on global performance.

The simulator is used to assess the performance of the distribution in presence of selfish or malfunctioning peers as a function of the distribution

architecture. Results shows that selfish peers have no great impact and the results on performance we obtain considering no selfish peer still hold.

This chapter is organized as follows. Sect. 3.2 describes the analytical model and the approximations introduced. In Sect. 3.3 we introduce the simulator and validate the analytical model. Sects. 3.4 and 3.5 discuss results in different networking scenarios and in presence of selfish peers. Sect. 3.6 ends the chapter with some additional discussions.

3.2 The Analytical Model

Consider a scenario where peers have different symmetric access link bandwidths (we discuss implications of asymmetric bandwidths in Sect. 3.5). There is only one server in the system with bandwidth at least equal to the highest peer bandwidth. All peers are independent, so we can describe the system through a random variable $b_{\text{p\_i}}$, the bandwidth of peer i, having a known density, which is identically distributed for all peers. The density function of $b_{\text{p\_i}}$ summarizes the fact that peers in the network might dedicate only part of the bandwidth for file distribution and also the fact that there could be peers with different access technologies. The bandwidth of a peer stays constant during the whole file transfer.

We assume that the file is partitioned into C independent chunks. Each peer can start serving the file to another peer once it has completely received the first chunk. The file size is $\mathcal{F}$; the time needed to download the complete file at the lowest bandwidth in the network is referred to as $T_R = \frac{\mathcal{F}}{\min(b_{\text{p\_i}})}$ and is also called *one round*.

Initially, we suppose that peers remain on line till the end of the distribution process. The relaxation of this hypothesis is discussed in Sect. 3.5. The signaling messages necessary to manage the dynamics of the overlay structure (join, leave, synchronization with neighbors, message used to

build the distribution architecture) are negligible with respect to the file size, and no errors, failures or other bottlenecks other than the peer access link are present.

The delivery process is distributed and the topology (chain, tree, ...) is built step-by-step. Each peer has a (possibly partial) list of the peers involved in the distribution process and contacts a node in the list randomly, checking that it has not yet been contacted. The optimization of this process is beyond the scope of this contribution.

3.2.1 Single Chain Analysis

Similarly to a normal multi-link transmission, the total download time, for a given n, number of peers, can be divided into two terms: the time necessary to reach the n-th peer and the time necessary to upload the whole file to that peer. We obtain

$$t^{(n)}_{\text{total}} = t^{(n)}_{\text{reach}} + t^{(n)}_{\text{dwnl-file}} \tag{3.1}$$

From the probability distribution of the random variable b we can derive the probability distribution of the download time for a single peer, i.e., $f_{t_{\text{dwnl-file}}}(\tau)$, with $t_{\text{dwnl-file}} = \frac{\mathcal{F}}{b}$, where the file size $\mathcal{F}$is a constant. The file is divided in C chunks and we can derive $f_{t_{\text{dwnl-chunk}}}(\tau)$, with $t_{\text{dwnl-chunk}} = \frac{\mathcal{F}}{Cb}$.

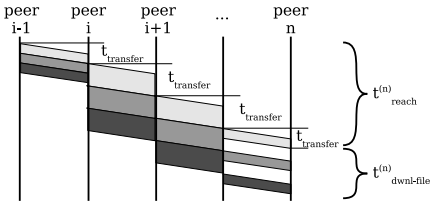


Figure 3.1: File transfer over a single chain

The time to reach the n-th peer is the time to transmit the single chunk through the chain. The transmission rate at each step is determined by the

minimum capacity between the uploading and the downloading peer and corresponds to the maximum transfer time. The probability distribution of a single transfer can be found through the cumulative distribution of a single chunk download time:

$$\begin{aligned} t_{\text{transfer}} &= \max_{node-u,node-d}(t_{\text{dwnl-chunk}}) \\ F_{t_{\text{transfer}}}(\tau) &= F^2_{t_{\text{dwnl-chunk}}}(\tau) \end{aligned} \tag{3.2}$$

where $F_{t_{\text{dwnl-chunk}}}$ is the cumulative distribution function (CDF) of $t_{\text{dwnl-chunk}}$. Equation (3.2) is correct as far as peers are i.i.d. Since we suppose that the transfer time on each link is independent with respect to previous links, the probability distribution of n transfers is the convolution of the distributions:

$$\begin{aligned} t^{(n)}_{\text{reach}} &= \overbrace{t_{\text{transfer}} + t_{\text{transfer}} + \cdots + t_{\text{transfer}}}^{n \text{ times}} \\ f_{t_{\text{reach}}|n}(\tau) &= \overbrace{f_{t_{\text{transfer}}}(\tau) * \cdots * f_{t_{\text{transfer}}}(\tau)}^{n \text{ times}} \end{aligned} \tag{3.3}$$

where the symbol '$*$' defines convolution.

The time necessary to transmit the whole file, once a peer is reached, depends on the capacity of all the previous nodes. Let b^i be the capacity of node i and $b^{(i)}$ be the capacity used to transmit. By construction $b^i \geq b^{(i-1)}$, since the transmit rate $b^{(i-1)}$ includes the capacity of node i and node i receives a chunk every $\frac{\mathcal{F}}{Cb^{(i-1)}}$. The transfer rate to the node $i+1$ depends on the capacity b^{i+1}. Node i will upload chunks with a rate that is the minimum between the rate it receives the chunks and the rate node $i+1$ can accept chunks. In formulas

$$b^{(i)} = \min(b^{(i-1)}, b^{i+1})$$

or, equivalently, the time necessary to transmit the file at step i, $t^{(i)}$, is

$$\begin{aligned} t^{(i)} &= \max(t^{(i-1)}, t^{i+1}) \\ F_{t|i}(\tau) &= F_{t|i-1}(\tau) F_{t_{\text{dwnl-file}}}(\tau) = F^{i+1}_{t_{\text{dwnl-file}}}(\tau) \end{aligned} \tag{3.4}$$

where $F_{t|i}(\tau)$ defines the conditional CDF of the file distribution after i peers have been reached, and $F_{t_{\text{dwnl-file}}}(\tau)$ is the CDF of the file download of a single peer and the last equality is obtained by iteration.

With (3.3) and (3.4) we have the distributions of $t^{(n)}_{\text{reach}}$ and $t^{(n)}_{\text{dwnl-file}}$ respectively. Unfortunately these variables are not independent; however, since we are interested in large n and large C to exploit parallelism, the distribution of $t_{\text{dwnl-file}}$ tends rapidly to the maximum download time T_d, so that

$$f_{t_{\text{total}}|n}(\tau) \xrightarrow{n \gg 1} f_{t_{\text{reach}}|n}(\tau - T_d) \ . \tag{3.5}$$

Eq. (3.5) has been validated by simulation. With the probability distribution we can calculate the mean time necessary to reach n peers and then we can build a graph of time versus the number of peers and study how the chain evolves with different bandwidth distributions as input. For notation simplicity we set $t = t_{\text{total}}$.

Starting from Eq. 3.5, with simple stochastic manipulation we can also obtain $f_{n|t}(\eta)$, which describes, at a given time instant, the distribution of the number of reached peers.

Figure 3.2 shows the conditional distributions, $f_{t|n}(\tau)$ and $f_{n|t}(\eta)$, for different values of n and t respectively, obtained with the bandwidth distribution in Table 3.1.

3.2.2 Multiple chains

With a chain based architecture, the server, as soon as it finishes uploading the file to a peer, starts to upload to another peer, creating a new chain. A new chain is created every $\Delta = \frac{\mathcal{F}}{b_{\text{fast}}}$ seconds. Considering that the system has started the distribution at time $t = 0$, given a instant of observation t, we know exactly how many chains are present in the system and the probability distribution $f_{n|t}(\eta)$ of the number of peers in each chain. The

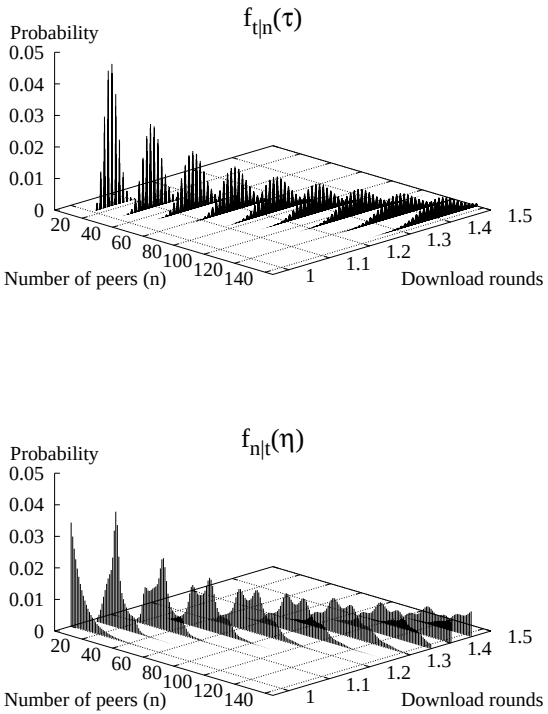


Figure 3.2: Example of different conditional distributions obtained with the bandwidth distribution in Table 3.1

total number of peers in the system at time t is the sum of the peers in each chain and, from the probability distribution of the number of peers in each chain, we can find the distribution of the total number of peers. Since all the chains are independent, we have

$$f_{n_{total}|t}(\eta) = f_{n|t}(\eta) * f_{n|t-\Delta}(\eta) * f_{n|t-2\Delta}(\eta) * \cdots \tag{3.6}$$

where the convolution is repeated until $t - i\Delta > 0$. If we are interested only in the mean number of peers, since the sum is a linear operation, we can find it from the mean number of peers of a single chain at time t, $t-\Delta$, $t-2\Delta$, ...

3.2.3 Tree Based Architectures

The analytical model defined in the previous section can be extended to tree architectures. Considering a tree (for instance, a binary tree) and following a sample-path from the root to a leaf we obtain a chain, as pictorially represented by the black nodes in Fig. 3.3. We can apply the same analysis technique used for chains to the tree case, since, in a stochastic sense, the node in the path at level j of the tree is representative of the whole level.

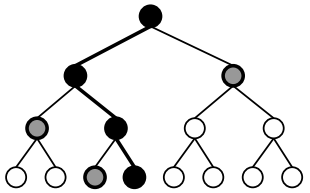


Figure 3.3: Example of a sample path in a binary tree; black nodes are in the sample path, gray nodes are those that influence the computation of b_j.

In each step we analyze all the children of a node and assign the bandwidth of the parent node among children: we define the bandwidth b_j used to calculate the chunk transfer time at the j-th step, as the bandwidth

assigned according to the max-min fairness criterion. Given the outdegree and the pdf of $b_{\text{p\_i}}$, the $f_{b_j}(b)$ can be easily computed with combinatorial techniques, analyzing the possible permutations of children bandwidths. Given the pdf $f_{b_j}(b)$ the term $t^{(n)}_{\text{reach}}$ is computed with (3.3).

Now, the term $t^{(n)}_{\text{dwnl-file}}$ must consider the brother nodes (gray circles in Fig. 3.3). In fact, considering a path, the interval between chunks not only depends on the bandwidths of the peers belonging to the path, but also on the bandwidths of the peers branching from the path. Due to this dependence, (3.5) is not valid anymore. Nevertheless, with the approximation of independent transfer times we have

$$\begin{aligned} t^{(n)}_{\text{dwnl-file}} &= \max\left(t^{(n-1)}_{\text{dwnl-file}}, (C-1)\frac{\mathcal{F}}{Cb_n}\right) \\ &= (C-1)\max\left(\frac{\mathcal{F}}{Cb_1}, \frac{\mathcal{F}}{Cb_2}, \ldots, \frac{\mathcal{F}}{Cb_n}\right) \end{aligned} \tag{3.7}$$

Equation 3.7 still allows for an iterative solution, with the only drawback that we consider twice the bandwidth of each node, which leads to overestimate the transfer time as we can see in the simple example of Fig. 3.4. We assume a binary tree and let p_F be the probability to have a fast node. The maximum rate in a step is $b_{j\max} = (b_{\text{fast}} - b_{\text{slow}})$, where b_{fast} and b_{slow} are the bandwidths of fast and slow peers respectively. To have two consecutive maximum rate steps we need three fast and two slow nodes involved (left hand side of Fig. 3.4), so that the probability of this event is $p_{2\text{f}} = p_F^3(1-p_F)^2$. Assuming instead that steps are independent, is equivalent to "duplicating" nodes, as in the right hand side of the figure, which implies that the approximated probability of having two consecutive maximum rate steps is $p'_{2\text{f}} = p_F^4(1-p_F)^2 < p_{2\text{f}}$. Repeating the operation for any number of fast steps yields that the probability of fast (not only the fastest) paths is *underestimated* when we assume independent steps.

On the other hand, to have a slow path we just need a single slow

step, which occurs with probability $p_{1\text{f}} = (1-p_F)^3$ and is unchanged by the independence approximation. This simple induction implies that the download time computed assuming independence is an upper bound to the real download time.

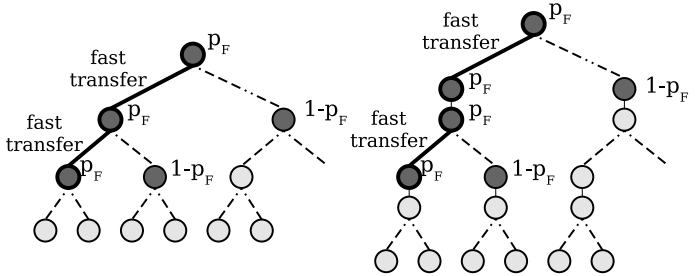


Figure 3.4: Real distribution tree (left) and equivalent tree with "duplicated" nodes considering independent steps (right).

The analysis of PTrees is surprisingly trivial, with the assumption that each node gives priority to the download of the stripe for which it is an interior node (say stripe m), and starts downloading the other stripes (for which it is a leaf) only after it finishes downloading stripe m. This assumption is reasonable because the order of stripe downloads does not affect the total download time for a node. Therefore, a node has thus no reason to delay the delivery of the stripe for which it is an interior node. The upload of stripe m to other peers does not affect the download of the other stripes.

In PTree, every peer is interior node in one tree and leaf node in other $r-1$ trees; this implies that (i) the total download time is dominated by the stripes received as leaf node, and (ii) all peers are equal, i.e., they have the same pdf of the total download time. The rate at which the file arrives at leaf nodes is determined by the slowest bandwidth encountered in the path; the probability that this rate is equal to the lowest bandwidth in the system ($\frac{b_{\text{slow}}}{k}$, since the bandwidth is divided among the k children)

increases rapidly with the depth of the tree. We approximate the pdf to a single Dirac's delta, obtaining (we recall that each stripe is $\mathcal{F}/r$ bits)

$$t_{\text{total}}^{(l)} = \frac{k}{r}\frac{\mathcal{F}}{b_{\text{slow}}} + t_{\text{reach}}^{(l)} \tag{3.8}$$

where $t_{\text{reach}}^{(l)}$ is the time necessary to reach leaf nodes when the tree depth is l. The term $t_{\text{reach}}^{(l)}$ grows logarithmically with n, number of peers, and we can suppose that is equal to l times the mean chunk transfer time between two peers. If we set $k = r$ we obtain that all peers terminate approximately at $\frac{\mathcal{F}}{b_{\text{slow}}} + l \cdot t_{\text{Cslow}}$, where t_{Cslow} is the chunk transfer time for the slowest peer.

3.2.4 Results of the Analytical Model

As numerical example we consider two different density functions for the peer bandwidth, summarized in Table 3.1. The distribution in the upper table is taken from [34]. The other one is an example that aims at studying the interaction among high speed peers that reserve a percentage of their bandwidth for other tasks. We assume that the bandwidth available for the file distribution process is uniformly distributed between 80% and 100% of the peer bandwidth.

When reporting results, we normalize the data such that $T_R = \frac{\mathcal{F}}{\min(b_{\text{p\_i}})} =$ 1 round. We use a number of chunks C equal to 100, but a sensitivity analysis with different values of C indicates a qualitative behavior independent of C, as far as $1 \ll C \ll N$, where N is the total number of peers.

Fig. 3.5 shows how the pdf of $t_{\text{dwnl-file}}$ evolves along different levels of the same tree. In this case we consider a binary tree with input pdf A from Table 3.1. Initially (we show here the 8-th level of the tree) the contribution of the three classes of peers is clear (continuous lines); the maximum download time is 2 rounds since the outdegree is 2. As the number of levels increases (when we reach the 25-th level of the tree), the

Table 3.1: Rate distributions used in the examples

Rate distribution A	
Rate	Weight
56 kbit/s	13%
640 kbit/s	23%
1.2 Mbit/s	64%

Rate distribution B		
Rate	Density	Weight
640 kbit/s	Uniform(80%-100%)	30%
2 Mbit/s	Uniform(80%-100%)	70%

distribution starts to concentrate on the maximum download time (dashed line), because the minimum bandwidth dominates.

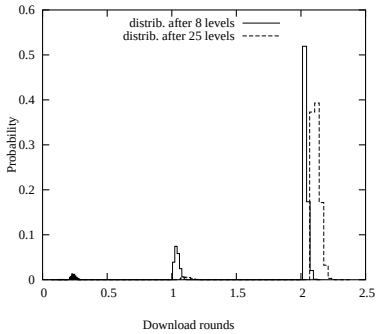


Figure 3.5: Evolution of the pdfs of the download time for two different tree heights.

Fig. 3.6 shows results of chain architecture and the tree with outdegree 2 and 3, using the density A of Table 3.1. We compare the heterogeneous case with the homogeneous one: the comparison considers for the homogeneous case an equivalent bandwidth equal to the mean transfer bandwidth of a single step. We do not present here results for PTree for clarity, since its

behavior is clear from Fig. 3.7.

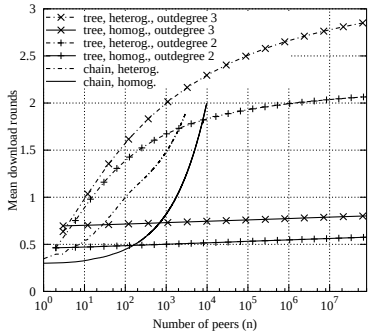


Figure 3.6: Mean completion time for a given number of peers: comparison between chain based and tree based architecture, heterogeneous peers (rate distribution A) and homogeneous with average bandwidth.

A detailed analysis of the results for homogeneous case can be found in [18]. Here we report a few essential observations. In a tree architecture, each peer uploads with bandwidth b_{p}/k, so the minimum necessary download time is $k\mathcal{F}/b_{\mathrm{p}}$. The time to receive the first chunk grows logarithmically in the number of peers since in a homogeneous environment all the transfers occur with the same rate and the only difference in the total download time is the time necessary to reach the l-th level.

The performance degradation in case of heterogeneous bandwidth is clearly due to the fact that a slow peer influences all the downstream nodes, regardless of their bandwidths. For instance, in case of tree architecture, the subtree under the slow peer proceeds slowly, so the number of peers that are forced to continue with low rate grows exponentially. The results of these dynamics are the convergence of the curve to k rounds. After few levels, the total download time of each peer, $k\mathcal{F}/b_{\mathrm{p\_i}}$, tends to become $k\mathcal{F}/b_{\mathrm{slow}} = k$ rounds. The chain based architecture has the same behavior. In general, the heterogeneous case converges, as the number of peers increases, to the homogeneous case, with the homogeneous bandwidth equal to the minimum bandwidth. As is clear from the analysis of the chains, the download time increases polynomially with the number of peers.

We observe the same behavior with different input pdfs. Fig. 3.7 presents the results using the density B of Table 3.1. We remark only a slight difference due to higher percentage of slow peers than in the previous case, so the curves converge faster to k rounds. For PTree architecture, results converge to the maximum download time for all peers. Note that results are normalized using the smallest bandwidth in the system: in the two cases (with input pdf A and B) the normalization factors are different and so the absolute times.

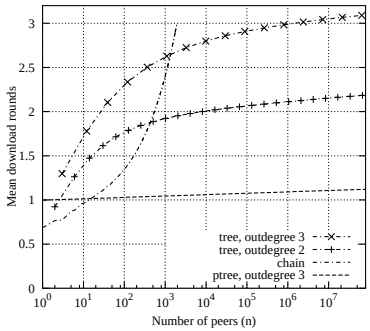


Figure 3.7: Mean completion time for a given number of peers: comparison between chain based and tree based architecture (rate distribution B).

From the study of the analytical results it is possible to draw some observations. As the number of peers grows, tree-based architectures are obviously faster than chain based ones, but tree-based architecture pay an initial cost due to the multiplication of the download time by the outdegree k. Therefore, initially, short chains are faster than small trees. Biersack et al. [18] demonstrated that the cross point between chain and binary tree in the homogeneous case is a function of the number of chunks C. In

the heterogeneous case, we can observe that this cross-over point occurs for a number of peers that is one order of magnitude greater than in the homogeneous case. We conclude that short chains are preferable up to roughly C peers.

3.3 Overlay Network Simulator

In order to evaluate the impact of the independence assumptions made in solving the analytical model, we developed a simulator that takes into account all correlations in the different distribution architectures . The simulator can also handle dynamic distribution Trees and PTrees and other cases discussed later and can be extended to generic meshes. For the simulator we use the same general assumptions made for the analytical model (Sect. 3.2).

3.3.1 Simulator Description

The simulator takes as input any discrete probability density function (pdf) of peer bandwidths and builds step by step a sample path. By performing the building process multiple times, it derives a histogram of the pdf of the relevant performance figures. We terminate the simulations when the mean and the 90-percentile of the total download time reaches a 99% confidence level for an interval $\pm 10\%$ of the point estimate. We use the batch means technique with repeated independent simulations.

The simulations are fast: indeed the transfer rate is dominated by the lowest bandwidth encountered along the path, and the histogram converges rapidly. Moreover, if the bandwidth of the peer are sufficiently large, the tail of the distribution is limited and its upper bound can be easily estimated.

In tree architectures we use a single path in the tree as we did for the

analysis. At each step down the tree, a number of children nodes equal to the outdegree of the parent tree are randomly chosen, and the bandwidth is divided among the different children according to the max-min fairness criterion. The following step in the path is selected choosing randomly one of the children nodes and repeating the process.

3.3.2 Comparison with the Analysis

Fig. 3.8 compares the results of the analytical model with the ones of the simulator, using the rate distributions A and B of Table 3.1. The upper plot refers to distribution A, and the lower one to distribution B. In the chain and PTree architectures, the analytical model fits the simulation experiments almost exactly.

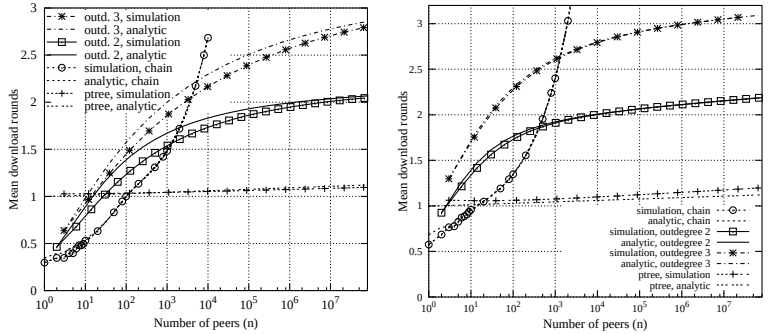


Figure 3.8: Comparison between analysis and simulations with different architectures; the upper plot refers to rate distribution A and the lower one to B.

Considering the tree architecture, as discussed in Sect. 3.2.4, the analysis yields indeed an upper bound of the actual average download time, as is confirmed by the simulation results. It is interesting to notice that the bound becomes tighter as the number of peer increases and as the fraction of slow peer increases (lower plot). This is easily understood considering

that the overestimation comes from underestimating the weight of fast paths down the tree and thus disappears as the probability of fast paths tends to zero.

In the following sections we discuss some interesting results in cases that do not lend themselves to an easy theoretical analysis, and all results presented are obtained using the simulator.

3.4 Improving the Distribution Architecture

3.4.1 Analysis of Hybrid Architectures

With tree architectures we have a degree of freedom in the design process: the outdegree. In the homogeneous case, the decision of the outdegree is a classical optimization problem. In a heterogeneous environment, when it is not possible to know the bandwidth of the children nodes, the optimal outdegree does not exist. The best solution is to let the outdegree to *dynamically adapt* to the changing conditions.

With a dynamic outdegree, peers need a criterion to set the outdegree. There are essentially two cases where increasing the outdegree is beneficial:

- The selected children are not able to use all the parent peer's upload bandwidth;
- A peer is downloading from a parent with a rate lower than its upload rate.

The first case is self explanatory. The second case can be explained as follows. If a peer has a bandwidth b and a number of children equal to k, it takes $t_u = \frac{\text{chunk\_size}}{b/k} = k\frac{\text{chunk\_size}}{b}$ to distribute one chunk to all its children. Let t_δ be the download time of a chunk; if $t_\delta > t_u$ the peer can increase the number of children k until reaching $t_u \simeq t_\delta$. The value of t_δ can be

estimated during the download process, so that a peer can increase its own outdegree without the risk of becoming a bottleneck.

Fig. 3.9 shows the mean download time, given the mean number of peers. We consider the mean number of peers since we measure the mean outdegree at each level and then we accordingly calculate the mean number of peers in each level. The bandwidth distribution is A, and we consider different upper limits to the outdegree.

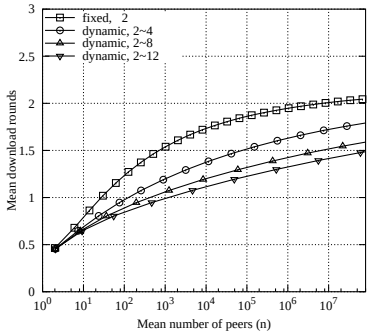


Figure 3.9: Mean completion time for a given number of peers in a tree based architecture with dynamic outdegree; rate distribution A.

As the maximum number of allowed children increases, the mean download time decreases. The explanation of such results can be found analyzing the cumulative distribution of the completion time. We set the same number of reached nodes and we compare the total download time of trees with different variable outdegree. Fig. 3.10 shows the percentage of completed peers as a function of time for $n = 10^7$.

Allowing the outdegree to increase, enables more and more peers to be reached in a given level. Moreover it enhances the probability that one of the children is fast. This is reflected by a larger fraction of peers ending the transfer early (thus reducing the average). However, most of the peers still need 2 rounds to finish, and the gain we have as the maximum outdegree

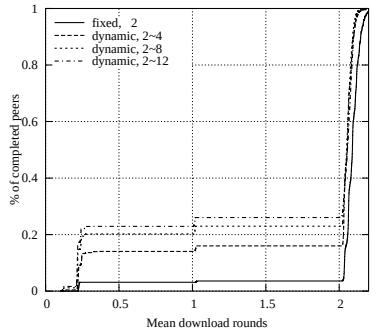


Figure 3.10: Percentage of completed peers at a given time in a tree based architecture with dynamic outdegree; rate distribution A.

increases becomes progressively smaller.

3.4.2 Changing the Minimum Outdegree

Fig. 3.10 highlights the fact that the performance is dominated by peers that terminate at time 2 rounds. In a tree with a minimum outdegree $k_{\min}$, the minimum time necessary for slow peers to download the file is $k_{\min}$ rounds. Recall the results obtained in Sect. 3.2.4: short chains are faster than small trees. If we combine chains and trees, with short chains joining different parts of the tree, when the $t_{\text{dwnl-file}}$ via a tree would be larger than 1 round, we obtain a hybrid distribution architecture that should be faster than trees. This is readily obtained allowing a lower outdegree equal to 1. Fig. 3.11 shows an example of tree evolution with two classes of peers.

Fig. 3.12 shows the performance obtained with a minimum outdegree equal to 1, compared to a minimum outdegree 2, using the input rate distribution A.

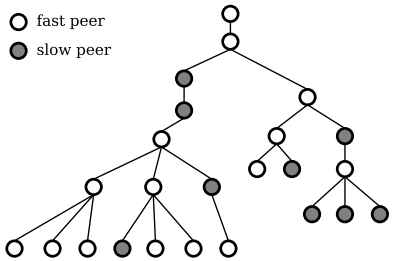


Figure 3.11: Example of evolution of fast hybrid tree.

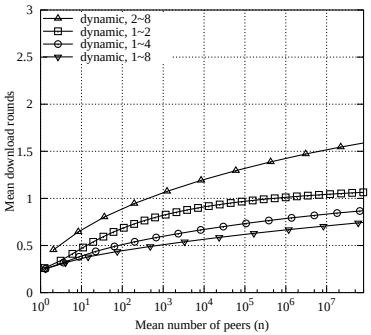


Figure 3.12: Mean completion time for a given number of peers in a tree based architecture, with minimum outdegree 1; rate distribution A.

3.4.3 PTree with dynamic outdegree

So far, we have discussed only briefly the performance of PTree, which is however the best performing architecture, since it fully exploits the resources of all peers. On the other hand, the PTree architecture is also the most sophisticated one and can be fragile in case of uncertainty and dynamic behaviors. For example, if the tree grows dynamically and becomes unbalanced, it may happen that in one of the sub-trees there are not enough internal nodes to serve the other peers.

The PTree architecture has two degrees of freedom: the outdegree k and the number of stripes r a file is divided up. Note that the number of stripes is not related to the number of chunks C. In fact, usually the number of chunks is of the order of hundreds to thousands, while the number of stripes can be in the range of 3-8. Each stripe then contains C/r chunks. For simplicity, we consider that C is a multiple of r.

With homogeneous nodes and a fixed outdegree k, PTree allows to receive in parallel r stripes with bandwidth b_{peer}/k each, obtaining a download time equal to $\frac{k}{r}\frac{\mathcal{F}}{b_{\text{peer}}}$. Choosing $k = r$, we obtain a minimum download time $t^{\min}_{\text{dwnl-file}} = \frac{\mathcal{F}}{b_{\text{peer}}}$, that is the time it takes a node to download the file via unicast communication, hence the optimal performance.

When we introduce heterogeneity, this equilibrium is broken. If a single stripe of the file is downloaded at a smaller rate, say b_{slow}, the whole performance is dominated by the download of this stripe. Indeed, the peer terminates the download only when this stripe is completely received, regardless of the number of stripes that download faster.

Additionally, when introducing a dynamic outdegree, in particular with a minimum outdegree equal to 1, we have to guarantee a minimum level of fairness and, at the same time, that there are enough leaf nodes in any tree to take the role of interior nodes in the remaining $r-1$ trees. In Sect. 2.2.1

we stated that $k \geq r$. We have verified empirically by simulation that a value of $k = 1.5r$ is sufficient to guarantee that the distribution process can take place without problems for any reasonable distribution of $b_{\text{p\_i}}$. The value of 1.5 is the ratio of data uploaded to data downloaded, which is in any case better than in a Tree architecture, where half of the peers do not upload at all.

Fig. 3.13 shows the performance of PTree with $r = 3$ and rate distribution A. Decreasing the minimum outdegree, decreases the mean download time as expected. Surprisingly, the results are only marginally influenced by the maximum outdegree, provided that the maximum outdegree is sufficiently large to ensure a mean outdegree greater than the number of stripes. Still, having a higher outdegree has the advantage of reaching a larger number of nodes for a given number of levels in the tree structure.

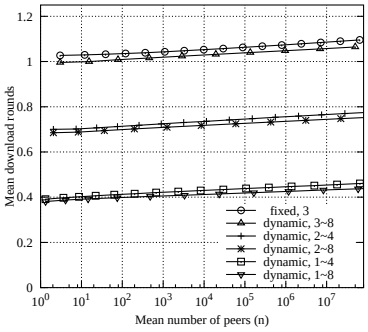


Figure 3.13: PTree performance with different dynamic outdegrees; $r = 3$.

Fig. 3.14 shows the cumulative distribution of the completion time of the peers. Most of the peers terminate roughly at the same time, as they would in a PTree with only fast peers while the 13% of slow peers terminate at time equal to 1 round, which is their best possible performance.

Since performances are not influenced by the maximum outdegree and

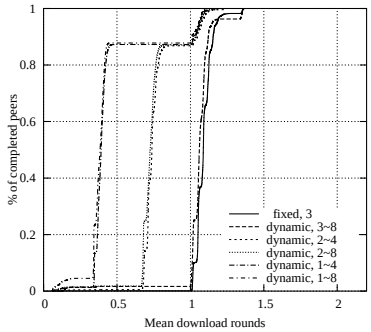


Figure 3.14: Percentage of completed peers at a given time in PTree architecture with dynamic outdegree; rate distribution A.

the optimal minimum outdegree is 1, the last parameter that can be tuned is the number of stripes r. Fig. 3.15 shows the total download time when the file is divided in different stripes. In all cases, we use a tree with dynamic outdegree in a range of 1 – 8. As expected, the performance improves with increasing number of stripes. The limit to this gain is given by the ratio between the bandwidth of the fast nodes and slow nodes: we can obtain an improvement as long as the bandwidth of the fastest nodes is r times greater than the one of the slowest node, i.e., it can accept in parallel all the r stripes (one as interior node, the other as leaf node).

The absolute values of the performances depends also on the specific rate distribution. However, Fig. 3.16, where rate distribution B is used, shows that the relative merit of the distribution architecture as a function of parameters remains unchanged.

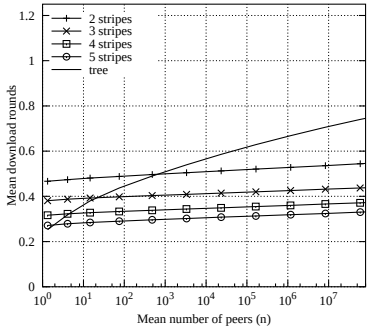


Figure 3.15: Tree vs. PTree performance with different number of stripes, the outdegree for all the configurations is 1–8.

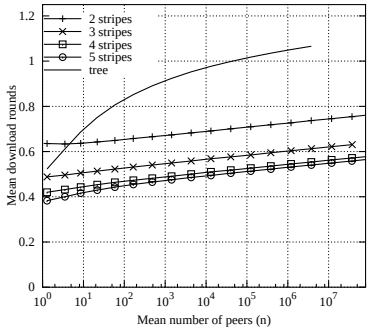


Figure 3.16: PTree performance with different number of stripes, rate distribution B (the outdegree for all the configurations is 1–8).

3.5 Additional Insights

3.5.1 Asymmetric Access Bandwidth

Throughout the analysis of the architectures, we have assumed *symmetric* bandwidth for upload and download. Recently, asymmetric broadband access links, like ADSL, have become very popular, and assessing how ADSL affects the download time is important. As a general consideration, we observe that a peer with asymmetric bandwidth can increase its own download performance, but cannot increase the performance of the whole distribution process that leverages on the upload bandwidths.

Table 3.2: Rate distribution with asymmetric bandwidths

Rate Downlink	Rate Uplink	Weight
640 kbit/s	128 kbit/s	20%
2 Mbit/s	2 Mbit/s	80%

Consider the pdf of the bandwidth as given in Table 3.2, where one class has ADSL access. We compare the results with the case when the ADSL class has symmetric bandwidth equal to the downlink and uplink bandwidth of the ADSL respectively. Fig. 3.17 shows results for the Tree architecture. The performance with asymmetric peers is equivalent to the one with symmetric peers with minimum bandwidth.

We observe a different behavior with PTree (Fig. 3.18). The performance of asymmetric peers lies between the two symmetric cases. PTree results depend on parallel downloads, each of the download has a rate equal to the minimum rate in the system. In this case ADSL users are the slowest peers and with symmetric bandwidth they can not exploit parallel downloading. With asymmetric bandwidth, instead, the downloading bandwidth can accept all the incoming downloading and the total download time decreases.

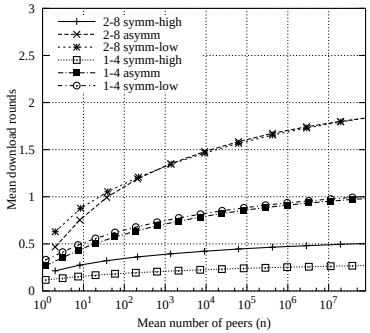


Figure 3.17: Mean download time with ADSL (tree architecture) compared with the symmetric case, where the bandwidth is either set to the minimum or to the maximum of the ADSL bandwidth.

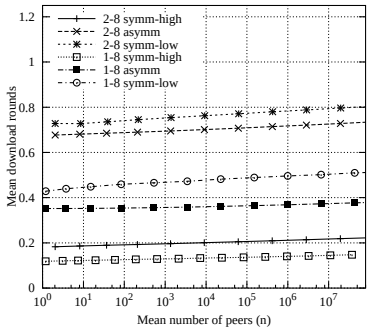


Figure 3.18: Mean download time with ADSL (PTree architecture) compared with the symmetric case where the bandwidth is either set to the minimum or to the maximum of the ADSL bandwidth.

3.5.2 Selfish Peers

The distribution process relies on peer collaboration: the more altruistic peers are, the faster the distribution. When the file distribution process is done in the global Internet, it has to face with free-riding. Free riders are peers that do not contribute by uploading the file, i.e., they are "selfish," and stall the distribution process.

The impact of selfish peers has been discussed in many papers (e.g., [24] [2]) in generic P2P networks, but not in cooperative content delivery. We assume that each peer has a probability p_{selfish} to be selfish. When we build the sample path in the tree, each step selects its children according to the bandwidths and the outdegree constraints. Since the presence of selfish peers decreases the number of reached peers at each step, in order to maintain the same total number of peers that complete the download, we increase the depth of the tree as p_{selfish} increases.

Fig. 3.19 shows the results for a tree architecture with different percentages of selfish peers and a maximum outdegree of 8. We can note only slight differences between different curves: in fact the tree structure is weak during the first steps, but, as soon as the depth, and consequently the number of peers, increases, the effect of selfish peers becomes smaller and smaller.

We obtain the same effect for the PTree architecture (Fig. 3.20). Since we increase the number of levels in each tree to reach the same number of peers, consequently, as the probability increases, the term $t^{(l)}_{\text{reach}}$ of (3.8) increases.

3.6 Discussion and Conclusions

In this chapter we discussed the performance of three simple distribution architectures in presence of heterogeneous peer bandwidth. We introduced

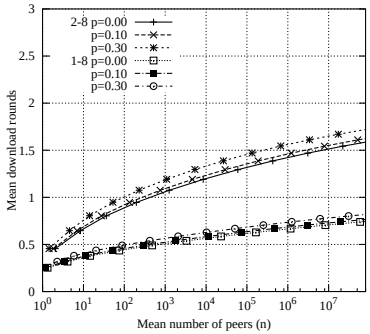


Figure 3.19: Mean download time with different percentage of selfish peers (tree architecture): the maximum outdegree is 8.

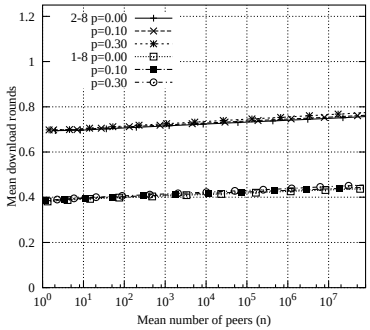


Figure 3.20: Mean download time with different percentage of selfish peers (PTree architecture): the maximum outdegree is 8.

an analytical model that provides a closed form solution assuming no correlation among successive distribution steps. The model proved to be very accurate due to a weak correlation structure of the distribution process.

The results obtained have been validated against simulations. Both the analytical model and the simulations yield the pdf of the distribution process, as a function of time and as a function of the number of nodes reached.

The insight given by the model has been used to devise a modified dynamic distribution architectures that enable the delivery to "get around" slow peers even when the knowledge about the peer bandwidth is limited or null. Even if the solution of dynamically modifying the degree of the distribution tree is not entirely novel (it has been used in many applications, mainly, protocols for streaming services), we have exactly quantified its impact as a function of the upper and lower bounds of the degree and a thorough analysis has been carried out.

Finally the presence of free-riders has been taken into account, showing that the selfish peers have not a great impact on performances. Although this might be intuitive, the quantitative analysis gives a robust method to evaluate whether it is necessary to introduce countermeasures in the application or the basic distribution architecture is robust enough to tolerate some misbehavior.

Chapter 4

Stochastic Graph Processes

The analysis on more sophisticated architectures, that comprises additional constraints such as a maximum number of neighbors, requires improved analytical tool: this chapter proposes a new methodology to model the distribution of finite size content to a group of users connected through an overlay network.

The methodology describes the distribution process as a constrained stochastic graph process (CSGP), where the constraints dictated by the content distribution protocol and the characteristics of the overlay network define the interaction among nodes. A CSGP is a semi-Markov process whose state is described by the graph itself. CSGPs offer a powerful description technique that can be exploited by Monte Carlo integration methods to compute in a very efficient way not only the mean but also the full distribution of metrics such as the file download times or number of hops from the source to the receiving nodes.

We model several distribution architectures based on trees and meshes as CSGPs and solve them numerically. We are able to study scenarios with a very large number of nodes and we can precisely quantify the performance differences between the tree- and mesh-based distribution architectures.

4.1 Introduction

In this chapter we develop a new model for file distribution processes. We do not attempt to model a specific file-distribution system, although this task may be possible using the proposed methodology. We are also not proposing yet-another file swarming application or a specific file distribution protocol. Rather, we propose a methodology to capture general properties of distribution protocols and algorithms in the attempt of providing guidelines for the protocol and system design.

We formalize the problem of building a content distribution overlay as a Constraint Stochastic Graph Process (CSGP), which is a discrete time Markov chain whose states are graphs with additional constraints describing the features of the distribution system. The graphs we are interested in are directed acyclic graphs, such as trees or meshes. Depending on how we define the transition rules from one state of CSGP to the next one, we get different content distribution overlays. We consider three different overlays, two of which are trees (hop-driven and cost-driven trees) and the third one is a mesh.

We then explain in detail the solution approach and the motivations underlying the use of the proposed methodology. We show how to use the formalism of CSGP to compute the metrics of interest for the content distribution such as the file download times, and the percentage of upload bandwidth left unused. Numerical results are obtained solving the stochastic process via Monte Carlo integration. Monte Carlo integration converges very quickly allowing to obtain results for very large overlays with up to a million of nodes, where standard event-driven simulations would fail for lack or time or memory. Monte Carlo integration has been used in other fields such as physics and chemistry, but, to the best of our knowledge, it has never been applied to modeling overlay or P2P networks. We also

briefly discuss, in Appendix A the similarity between modeling the content propagation across an overlay and the modeling of chemical reactions of a set of molecules.

The results we obtain precisely quantify the impact of the minimum outdegree and the improvement of mesh-based overlays as compared to tree-based overlays and provide important practical insights into how content distribution overlays should be constructed.

4.2 Problem Formulation

The novel methodology to model and analyze file distribution is based on a class of semi-Markov processes whose state is described through a graph. This property allows adding a reward structure to the process that is related to the topological properties of the graph, which enables the computation of the performance metrics and gives deep insight in the behavior of the file distribution systems.

4.2.1 Content Distribution

The aim of the service is the delivery of a given finite size content $\mathcal{F}$ to a set of users $\mathcal{N}$. The only requirement of the service is content integrity. The main performance metrics are the *download time* T of the content, either for a given user i (T_i), or for the whole community (T_t). We also consider the mean $\overline{T}$ of all the individual download times T_i.

We assume that each node knows a subset of the whole set of users, i.e., a node has a finite number of neighbors.

The content $\mathcal{F}$ is divided in C pieces called *chunks*. A chunk represents a basic unit of transmission that can be distributed independently. During the distribution of $\mathcal{F}$, a node that has started to download $\mathcal{F}$ can in turn start to upload $\mathcal{F}$ after entirely receiving the first chunk. The order in

which the chunks are received by a node is not important. However, the transfer is not complete until all chunks have been received by all interested nodes.

For each node i, we define b_i^u and b_i^d as the upload and download bandwidth respectively, which can be either symmetric, asymmetric or correlated, e.g., $b_i^u + b_i^d$ constant, as in a shared medium based access. The bandwidths are random variables described by a probability density function (pdf) that is known (e.g., derived from measurement studies).

When a node starts uploading chunks of $\mathcal{F}$, the *effective rate* used to transfer the chunks to each child is computed according to the *max-min fairness* criterion. The effective rate depends on multiple factors, such as the number of children of the uploading node, the rate the uploading node is receiving, etc. While the bandwidths (upload and download) are a given, the effective rates are computed during the distribution process.

We define the *eligibility time* t_i^{el} of node i as the time at which node i can start uploading chunks to other nodes, i.e., it has completely received the first chunk. If a node j is child of node i and receives at an effective rate r_{ij}, its eligibility time is $t_j^{\text{el}} = t_i^{\text{el}} + t_{ij}^{\text{step}}$, where $t_{ij}^{\text{step}} \stackrel{\triangle}{=} \frac{\mathcal{F}}{C}\frac{1}{r_{ij}}$. Knowing the eligibility time and the rate at which a node j receives chunks, the *total download time* can be computed as $T_j \stackrel{\triangle}{=} t_j^{\text{el}} + (C-1)t_{ij}^{\text{step}}$ (at the end of the eligibility time the first chunk is received, so there are $C-1$ chunks left).

As a last assumption we suppose that node i chooses its children j uniformly at random among all its neighbors, not taking into account upload and download bandwidths.

4.2.2 General definitions

The distribution of a content within a community of users can be formalized as the propagation of the content across a graph of nodes and edges with some stochastically defined characteristics. Nodes are the users and edges

summarize all the characteristics of the communication paths between the users.

Let $\mathcal{N}$ be the set of nodes, i.e., the vertices of the graph, and $\mathcal{A}$ the set of all the arcs that connect pairs of nodes, $\mathcal{A} \subseteq \mathcal{N} \times \mathcal{N}$. We only consider connected networks with bidirectional connections, so that $\mathcal{A}$ can be represented by an irreducible, symmetric adjacency matrix. $\mathcal{B}_i$ is the set of neighbors of user i, i.e., all those nodes in $\mathcal{N}$ that are known and directly reachable from node i, with $\bigcup_{i \in \mathcal{N}} \mathcal{B}_i = \mathcal{N}$, since the network is fully reachable. $\mathcal{B}_i$ is represented also by row i of the adjacency matrix $\mathcal{A}$.

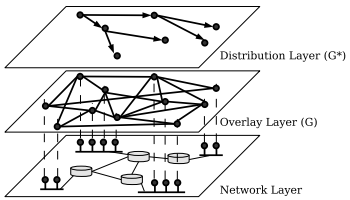


Figure 4.1: Overlay and distribution graphs

The graph $\mathcal{G}(\mathcal{N}, \mathcal{A})$ represents the overlay network created, for instance, by a P2P network (see Fig. 4.1). In general, $\mathcal{G}(\mathcal{N}, \mathcal{A})$ is time varying, i.e., nodes and edges can change in time, and even appear or disappear. The overlay layer is the basis on top of which the *distribution graph* is built. We define the distribution graph $\mathcal{G}^*(\mathcal{N}, \mathcal{E})$ as a directed subgraph of $\mathcal{G}(\mathcal{N}, \mathcal{A})$, with $\mathcal{E} \subseteq \mathcal{A}$. $\mathcal{G}^*$ is a directed graph, since, from the content distribution point of view, the content propagates from the source to the destinations.

4.2.3 Stochastic Graph Processes for Content Distribution

How to obtain the distribution graph $\mathcal{G}^*$ from $\mathcal{G}$ is determined by the rules implemented in the specific content distribution protocol. In general, we can assume that the distribution graph $\mathcal{G}^*$ is built step by step following

a given protocol. The building process can be modeled as a discrete time Markov chain, since the distribution graph $\mathcal{G}^*$ at step i contains all the information required to (stochastically) define the distribution graph $\mathcal{G}^*$ at step $i+1$. Let $\mathcal{N}_n^*$ be the set of nodes that belong to the distribution graph at step n, and $\overline{\mathcal{N}_n^*}$ its complement with respect to $\mathcal{N}$. The distribution graph $\mathcal{G}_{n+1}^*$ at step $n+1$ is obtained from $\mathcal{G}_n^*$ by adding new edges $\in \mathcal{A}$ from nodes in $\mathcal{N}_n^*$ to nodes in $\overline{\mathcal{N}_n^*}$. The complete distribution graph $\mathcal{G}^*(\mathcal{N}, \mathcal{E})$ is obtained when $\mathcal{N}_n^* = \mathcal{N}$, and $\overline{\mathcal{N}_n^*}$ is the empty set.

The dynamic behavior of the distribution graph can be modeled as a *stochastic graph process*. We recall here the general definition of stochastic graph processes [20], while in Sect. 4.3, we specialize them for the analysis of content distribution.

Definition 4.2.1 *A stochastic graph process (SGP) on a set of nodes $\mathcal{N}$ is a discrete time Markov chain (DTMC) whose states are graphs on $\mathcal{N}$.*

Even if not stated in the definition, two observations are in order:

- nodes can be connected only through edges that belong to $\mathcal{A}$ (in the next definitions, we state explicitly this dependence);
- the SGP is embedded in a continuous time semi-Markov chain that is sampled at the instants of adding nodes and arcs to obtain the SGP.

Adhering to the definition given in [20], the focus is the building process and the evolution of the SGP implies that the graph is built step by step by adding nodes and edges at each step.

In content distribution, the distribution graph is naturally built step by step, so using the graph $\mathcal{G}^*(\mathcal{N}, \mathcal{E})$ as a formal representation of the state of the system is appropriate and complete: i.e., that state contains all the information required to define (stochastically) the next state of the evolution. The time between two steps depends on the sojourn time of

the state. If sojourn times are exponentially distributed, then we obtain a Markov chain. However, in general, this assumption is not true and in continuous time we have a semi-Markov chain.

Definition 4.2.2 *A constrained stochastic graph process (CSGP) on a graph $\mathcal{G}(\mathcal{N}, \mathcal{A})$ is a semi-Markov chain whose states are subgraphs on $\mathcal{G}$. The semi-Markov chain embeds a Discrete Time Markov chain (DTMC) obtained by sampling the process exactly at transition instants. The "constraints" limit the degrees of freedom during state transitions and 'govern' the evolution of the process.*

A CSGP is a precise representation of the content distribution process.

Given $\mathcal{G}_n^*$, the next state $\mathcal{G}_{n+1}^*$ depends only on the eligibility times of the nodes in $\mathcal{N}_n^*$, and the transition probabilities can be easily defined step-by-step.

The eligibility times t_j^{el} influence the semi-Markov process in two different ways. In the general case of randomly varying t_{ij}^{step}, they define both the transition probabilities between states and the state sojourn times. In the particular case of deterministic t_{ij}^{step} (e.g., when the bandwidth is only determined by access links), sojourn times are deterministic, and the t_j^{el} define only the state transition probabilities. Notice, however, that the file distribution is entirely described by the embedded DTMC, so that only transition probabilities are important.

The DTMC that describes a CSGP is a transient chain with a set of adsorbing states $\mathcal{G}^*(\mathcal{N}, \mathcal{E})$ that are reached when $\mathcal{N}_n^* = \mathcal{N}$, i.e. all nodes interested in the content are reached.

The way we defined a CSGP implies that nodes are stable and collaborative, and that the networking infrastructure is reliable enough to allow edge stability. Clearly there is the possibility of extending the analysis to cases where nodes (or edges) can disappear during the distribution process,

so that $\mathcal{G}_n^*$ is derived from $\mathcal{G}_{n-1}^*$ not only by adding an edge and a node, but also by removing one node and all the edges relative to it.

A well known class of CSGP are the *random graph processes* studied back in the '50s by Erdös and Renyi in [21]. In random graph processes, edges are added uniformly at random, which does not capture the propagation of the data blocks in content distribution graph.

4.3 Content-Delivery CSGP

In the following, we refine the CSGP by adding constraints. These constraints concern the minimal and maximal outdegree and also the depth of graph. We define three different distribution architectures, referred to as 'Content-Delivery constrained stochastic graph processes', or CD-CSGP for short. Two of the CD-CSGP form trees and the third one a mesh. We have seen that the distribution graph as defined by a CD-CSGP grows step by step by connecting each time a new node to a node n in the existing graph. As we will see for the two tree-based distribution architectures, the final distribution tree will be very much affected by the way we select the node n in the graph to which the new node will be attached. The mesh-based distribution graph will be obtained by first constructing a tree-based distribution graph, which will then be augmented by additional links originating at the leaves of the tree. The advantage of a mesh is that the leaf nodes of a tree, which do not at all contribute to the distribution of the content, will now help distribute the file content.

4.3.1 Content-Delivery Related Definitions

Before we introduce the 'Content-Delivery constrained stochastic graph processes,' or CD-CSGP, we need some additional definitions that will simplify the characterization of each CD-CSGP.

Each node has a constraint on maximum and minimum number of active uploads that limit the possible outdegree of the node: $k_i^{\max}$ is the *maximum outdegree* and and $k_i^{\min}$ is the *minimum outdegree*.

Definition 4.3.1 (saturated node) *A node $i \in \mathcal{N}_n^*$ is called* saturated *if it*

- *has k_i^{max} outgoing edges that belong to $\mathcal{G}_n^*$ or*
- *has at least k_i^{min} outgoing edges and fully uses b_i^u to transmit chunks to neighboring nodes that belong to $\mathcal{G}_n^*$.*

Definition 4.3.2 (interior subset) *The subset $\mathcal{I}_n \subseteq \mathcal{N}_n^*$ of nodes that are saturated at step n is called the interior node subset at step n.*

Definition 4.3.3 (leaf subset) *The set of nodes $\mathcal{L}_n \in \mathcal{N}_n^*$ that are not interior nodes is called the leaf node subset at step n, with $\mathcal{L}_n = \mathcal{N}_n^* \setminus \mathcal{I}_n$.*

We consider a single node as a root of the stochastic graph. We define a distance measure based on the number of hops from the root to any node i.

Definition 4.3.4 (step distance) *The number of hops from the root to a node i following the shortest path is called* step distance *or* step depth*, $d^{(i)}$.*

In a tree, $\max_i(d^{(i)})$ is the *tree depth*.

4.3.2 Cost-driven and Hop-driven Trees

We now define the precise steps that must be executed in building the content distribution tree. The exact rules for building the tree have a big impact on the shape (number of hops, outdegree) of the final content distribution tree. In the following, we will introduce two different trees called *cost-driven* and *hop-driven* trees.

CD-CSGP 1 *A constrained stochastic graph process on graph $\mathcal{G}(\mathcal{N}, \mathcal{A})$ is called* ***tree-based*** *and* ***cost-driven*** *if*

1. *$\mathcal{G}_0^*$ is a node, called root, randomly chosen in $\mathcal{N}$.*
2. *$\mathcal{G}_n^*$ is obtained from $\mathcal{G}_{n-1}^*$ as follows*
   (a) *Choose the node i from $\mathcal{L}_{n-1}$ with the smallest eligibility time: $t_i^{el} = \min_j(t_j^{el})$; if several nodes have the same eligibility time, one of the nodes is chosen randomly;*
   (b) *Add edges from node i to nodes randomly chosen from $\mathcal{B}_i \bigcap \overline{\mathcal{N}_{n-1}^*}$, until node i becomes saturated.*

Figure 4.2 shows an example with few states of the DTMC generated by a CD-CSGP 1 process. In this case, we have only two possible bandwidths (slow nodes with black circles, fast nodes with white ones, with slow bandwidth less than half of the fast bandwidth) and $k_i^{\max} = 2$ and $k_i^{\min} = 1$. Starting, for instance, from a state where the server is uploading to a slow and to a fast node, the fast node has the smallest eligibility time and there are only three next possible states:

(i) the fast node selects a fast node among its neighbors and becomes saturated; alternatively, the fast node chooses a slow node so it has to select another node,

(ii) the selected node is fast and we have bandwidth saturation, or

(iii) the node chosen is slow and we have saturation because $k^{\max}$ is reached.

Note that in case (i) the node becomes saturated since the rate of the content it is receiving is high. If, for instance, the rate were slow (consider the fast node under the slow node in the shadowed state), the number of children would be always two, since the rate to each child is at most equal to the rate it is receiving.

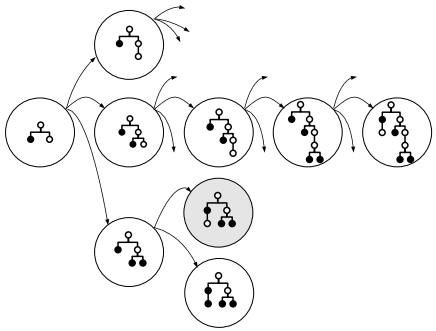


Figure 4.2: Sample of the embedded DTMC for a CD-CSGP 1 process; states are graphs built on $\mathcal{G}$, black and white circles represent slow and fast nodes respectively, $k_i^{\max} = 2$ and $k_i^{\min} = 1$.

The resulting tree is called "*cost-driven*" since, in general, the nodes in the leaf set $\mathcal{L}_n$ do not all have the same step distance from the root. As the speed of growth of the different branches is not the same, the deeper branches will contain faster nodes, i.e., nodes with smaller eligibility times t^{el} (eligibility times represent the costs).

There is a vast literature on tree-based distribution architectures and their performance. The majority of the proposals consider trees where leaves have the same step distance from the root (the model introduced in the previous chapter falls in this category). We call such trees "*hop-driven*"[1].

We define a special stochastic graph process for this type of tree. To do so, we consider a subset of the leaf set $\mathcal{L}_n$.

Definition 4.3.5 *Let $d_n^{MAX} = \max_j(d_n^{(j)})$ be the maximum step distance of the nodes $j \in \mathcal{L}_n$. The subset $\widetilde{\mathcal{L}_n} \subseteq \mathcal{L}_n$ is defined as follows: $\widetilde{\mathcal{L}_n} = \{\ i \in \mathcal{L}_n \mid d_n^{(i)} < d_n^{MAX}\}$.*

[1]Hop-driven are not necessarily balanced, since we have a variable outdegree.

Now we can define the process that leads to hop-driven trees.

CD-CSGP 2 *A constrained stochastic graph process on graph $\mathcal{G}(\mathcal{N}, \mathcal{A})$ is called* **tree-based** *and* **hop-driven** *if*

1. *$\mathcal{G}_0^*$ is a node, called root, randomly chosen in $\mathcal{N}$.*
2. *$\mathcal{G}_n^*$ is obtained from $\mathcal{G}_{n-1}^*$ as follows*
   (a) *Choose a node i from $\widetilde{\mathcal{L}_{n-1}}$ if not empty, otherwise from $\mathcal{L}_{n-1}$, with the smallest eligibility time; if several nodes have the eligibility time, one of the nodes is chosen randomly;*
   (b) *Add edges from node i to nodes randomly chosen from $\mathcal{B}_i \bigcap \overline{\mathcal{N}_{n-1}^*}$, until the node becomes saturated.*

Since we are interested in the file download time, it is worth to look at a weighted graph where the weight associated to a directed edge is given by the difference between the download times of the nodes connected by the edge. Considering hop-driven trees, this representation shows the disparity in terms of download time among leaf nodes that have the same step distance. In Fig. 4.3 the weight is represented by the edge length. Conversely, in cost-driven trees, leaf nodes are at different step distances and the weighted graph gives a pictorial illustration why the tree grows in this way: a new edge is added only after a node becomes eligible and this forces a uniform growth of the *weighted* graph.

The difference between hop- and cost-driven trees is significant. A node N in the tree will influence the reception speed of all nodes in the subtree with N as root. A slow node S will slow down the reception of the chucks for all the nodes in the subtree with S as root . Since slow nodes become eligible later than fast nodes, in the case of cost-driven tree, a subtree with a slow node as its root will grow much slower, i.e. have fewer nodes than

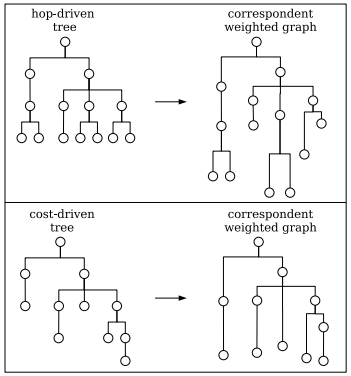


Figure 4.3: Difference between hop- and cost-driven trees, considering the corresponding weighted graphs where the length of edge between nodes i and j is given by t_{ij}^{step}

a subtree consisting of fast nodes, which will help to "limit" the impact of slow nodes (see the leftmost branch of Fig. 4.3)

We will show in Sect. 4.5 how the choice of the tree — hop- or cost-driven — affects the performance.

4.3.3 General Mesh Architecture

Tree based architectures allow the content to rapidly diffuse to nodes. However trees also have known shortcomings. Each node has only one ancestor and in case of a node failure, the entire subtree will stop receiving data. Each node must divide the upload bandwidth among its children, so children use only a fraction of their download bandwidth for receiving chunks; if we consider the case of asymmetric capacities, where the upload bandwidth is smaller than the download bandwidth (as in the case of ADSL), the percentage of unused download bandwidth increases even further. Finally, the leaf nodes of a tree receive the entire file without uploading a single chunk.

Mesh based architectures are meant to overcome these problems. In addition, if the direction of the content diffusion within the structure is random, the number of leaf nodes can be reduced to a single node given a perfect knowledge of the network [18].

Nevertheless, the mesh architecture introduces a new problem: the chunk selection strategy. A node i can help another node j if i has parts of $\mathcal{F}$ not yet received by j, i.e., if it has 'fresh' information. We are not concerned here on how freshness is checked and/or imposed, see for instance [35, 36] for work on the topic. We assume an ideal situation where if a node has received only part of $\mathcal{F}$ and it is contacted by another node that is not already its ancestor, then all the information it can provide is either completely fresh or completely stale. Any impairment can be easily taken into account with a probabilistic approach.

Allowing the generation of mesh topologies means that a node i already included in $\mathcal{G}_n^*$ may be contacted by other nodes j, also in $\mathcal{G}_n^*$ to receive parts of $\mathcal{F}$ it does not already have. If j only has information that is not fresh for i, then the 'delivery connection' is not established.

The building process of a mesh architecture can be divided into two phases: diffusion and interconnection. In the first phase, we assume that the root node uploads the chunks to its k children in different orders. For instance, child i will first receive the chunk j with $j \in \{1, ..., C\}$ and $j \bmod k = i$. This means that each child receives the whole file, but the first C/k chunks are disjoint with respect to the content received by the other children[2]. Each of the children generates its own *diffusion subtree* $\mathcal{FG}$, where new nodes not yet reached by the content (*untouched nodes*) are added to the subtrees. Once no more untouched nodes are available, subtrees start to interconnect. Leaf nodes of a subtree upload chunks to

[2]Other techniques such as network coding could also be used to assure that different children receives different chunks, but these details do not impact the performance of distribution architectures.

nodes belonging to different diffusion subtrees, providing up to C/k disjoint chunks. In the final configuration, we obtain a mesh, where each nodes receives from up to k nodes (belonging to distinct diffusion subtrees) and uploads to other nodes according to the saturation rules. This constrained mesh diffusion process is depicted in Fig. 4.4. Note that the diffusion direction is 'reverted' at leaves, ensuring a better spreading of the content.

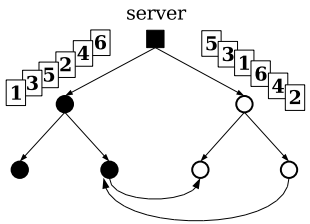


Figure 4.4: Mesh topologies obtained from interconnection of different diffusion subtrees

We can formally define the CSGP that leads to these architectures.

CD-CSGP 3 *A constrained stochastic graph process on graph $\mathcal{G}(\mathcal{N}, \mathcal{A})$ is called constrained* ***mesh-based*** *if*

1. *$\mathcal{G}_0^*$ is a node, called root, randomly chosen in $\mathcal{N}$.*
2. *$\mathcal{G}_1^*$ consists of the root node and the k nodes randomly chosen in $\mathcal{N} \setminus \{root\}$ called the first generation nodes, where and each of them will generate a subtree $\mathcal{FG}$.*
3. *$\mathcal{G}_n^*$ is obtained from $\mathcal{G}_{n-1}^*$ as follows*
   (a) *Choose a node i from $\mathcal{L}_{n-1}$ with the smallest eligibility time; if several nodes have the same eligibility time, one of the node is chosen randomly; $\mathcal{FG}_i$ denotes the subtree to which i belongs;*

   (b) *Add edges from node i to nodes randomly chosen from $\mathcal{B}_i \bigcap \overline{\mathcal{N}_{n-1}^*}$, until the node becomes saturated and $\overline{\mathcal{N}_{n-1}^*}$ is not empty;*
   (c) *If $\mathcal{B}_i \bigcap \overline{\mathcal{N}_{n-1}^*}$ is empty and node i is not saturated, add edges from node i to nodes randomly chosen from $\mathcal{B}_i \bigcap \overline{\mathcal{N}_{n-1}^\dagger}$ until the node becomes saturated, where $\mathcal{N}_{n-1}^\dagger$ is the set of nodes in the subtree $\mathcal{FG}_i$ at step $n-1$.*

CD-CSGP 3 can describe construction processes such as *SplitStream* [14] and *PTree* [18]: both of the schemes use a fixed outdegree k and partition the file in exactly k stripes, where each stripe is distributed along one of the k diffusion trees. The process we define is more general since we do not impose any fixed outdegree and allow that nodes upload the whole file, however in different order.

4.4 Solution of the CD-CSGP

The CD-CSGPs defined in Sect. 4.3 describe the evolution over time of the underlying Markov process: the Markov chain that models the process is a transient chain and its absorbing states are the states where the final graph contains all the nodes. So, we are interested in the analysis of the transient behavior of the system, i.e., we focus on time necessary to reach an absorbing state, or, given a time bound, the mean number of nodes contained in the distribution graph.

The transient analysis of a Markov process can be done considering the well known Kolmogorov forward (or backward) equations, coupled with the Chapman-Kolmogorov Equations, which lead to a set of differential equations that describe how the probabilities to be in a given state change over time. A closed form solution of these equations is not feasible unless for very small number of nodes, so we need to resort to numerical integration.

A numerical solution of the Markov process equations can be done using different methodologies. Direct methods that approximate numerically the equations have similar problems as the closed form solution: the equations can be written only for a small number of nodes.

However, the structure of the transition matrix that describes the stochastic process, is extremely suited for an efficient numerical solution based on Monte Carlo techniques [30][31][32][3].

Monte Carlo integration is basically a random walk in the state space of the process. The convenience of the methodology is given by the fact that it is very simple to build a random walk following the process definitions given in Sect. 4.3. Each random walk starts from the initial empty state and finishes in an absorbing state. At each step, we generate transition probabilities at run-time when hitting a state.

These samples are, by construction, independent and identically distributed, so we can compute the average characteristics of these random walks, along with the confidence interval given a desired confidence level.

Monte Carlo integration is efficient for a number of reasons.

- The small number of transitions from each state, all of them with non-vanishing probability, so that rare paths are non existing.
- The reward structure defined on the DTMC (see Sect. 4.4.2 for the definition of the metrics we use) ensures that there are no dominating rewards associated to low probability states (no 'rare event' syndrome is present in the problem), and that the coefficient of variation (standard deviation divided by the average value), of all output metrics decreases as the number of nodes increases. In fact, the variance remains bounded while the mean download time increases monotonically as we add more and more nodes. The consequence is that the numerical results yielding the histograms of the distribution converges (almost surely) more and more rapidly to the exact distribution as the number of nodes increases.
- The difference between the estimates and the exact value can be evaluated with standard stochastic means [33] yielding a powerful tool to stop the solution iteration.

[3] In physical and chemical sciences this technique is often called *Stochastic Simulation Algorithm* or *Gillespie Algorithm*, but we prefer to stick to the term 'Monte Carlo' normally used in computer science.

The Monte Carlo integration we propose is not a 'generic' event driven simulation, but the mere numerical solution technique for the analytical model. The definition of the problem as a CSGP ensures that the numerical solution generates all and only the states that are relevant to compute the performance metrics. A generic event driven simulator, written without formally defining the underlying stochastic process, may end up in exploring a state space where interesting states are sparse in the middle of states that are not useful to find the metrics of interest.

4.4.1 Detailed Description

The results of the Monte Carlo integration are an approximated solution of the differential equations that describe the process, but with a known error. All the probabilities that we compute are estimations of the real probabilities. For instance, when we find performance metrics as described in Sect. 4.4.2, we consider the probabilities of the absorbing states: actually these probabilities are the estimated probabilities, with an error bound given by confidence intervals.

The fast convergence of the integration using Monte Carlo is due to the fact that we look for node properties that are not necessarily related to the graph structure: for instance, given a time t and a specific number of nodes N_t that have completed the download, there are many different possible graphs that contain N_t nodes at time t. Another example can

be the download rates of nodes in the absorbing states: there are many graph structures where the number of nodes that complete the download with the slowest rate is the same. Thanks to these aggregate measures, the convergence of the distribution is fast.

4.4.2 Computation of the Performance Metrics

In order to compute the mean download time $\overline{T}$ we assign to each absorbing state $\boldsymbol{S}_k \in \boldsymbol{S}^a$ ($\boldsymbol{S}^a$ is the set of absorbing states, i.e., the states where all elements of $\mathcal{N}$ have downloaded $\mathcal{F}$ and no further transitions are possible) a reward $\overline{T}_k$ equal to the mean download time of the nodes in the state,

$$\overline{T}_k = \frac{1}{|\mathcal{N}|} \sum_{i \in \mathcal{N}} T_i \ ; \ \boldsymbol{S}_k \in \boldsymbol{S}^a$$

where T_i are the individual download times defined in Sect. 4.2.1. The mean download time $\overline{T}$ is the reward of a DTMC obtained by adding a deterministic transition from all the absorbing states to an initial state represented by the empty graph $\emptyset$.

$$\overline{T} = \frac{\sum_{k \in \boldsymbol{S}^a} \overline{T}_k \pi_k}{\sum_{k \in \boldsymbol{S}^a} \pi_k} \ .$$

where π_ks are defined as the steady state probabilities of the support DTMC.

Another performance measure easily defined as a reward is the *wasted upload bandwidth* $\overline{w^u}$ (in percentage). Let $w_i^u = 100 \left(1 - \frac{\max(r_i^u)}{b_i^u}\right)$ be the wasted upload bandwidth of node i, where $\max(r_i^u)$ is the maximum upload rate ever reached by the node in any visited state. Considering again the modified DTMC and letting $\mathbf{S}_a$ be the set of absorbing states in the unmodified DTMC we have

$$\overline{w_k^u} = \frac{1}{|\mathcal{N}|} \sum_{i \in \mathcal{N}} w_i^u \ ; \ \mathbf{S}_k \in \mathbf{S}_a$$

and

$$\overline{w^u} = \frac{\sum_{k \in \mathbf{S}_a} \overline{w_k^u} \pi_k}{\sum_{k \in \mathbf{S}_a} \pi_k} \ .$$

If we fix the number of nodes and let the time to go to infinity, we obtain the probability density function (pdf) of the download rates and the number of nodes that download at each possible rate normalized by the total number of nodes. This representation enables the comparison of the input pdf of the bandwidths with the output pdf of the rates, obtaining a direct measure of the impact of different policies (e.g., constraints on outdegree) on the performance.

4.4.3 The Numerical Solver

The numerical solver GROOVER (stochastic Graph pROcess sOlVER) implements a Monte Carlo integration in form of an algorithmic implementation of the content distribution processes defined as CD-CSGPs. It simulates the stochastic process and computes multiple realizations of that process. GROOVER has several input parameters and the outputs are the rewards associated with the process.

Algorithm 1 gives a high level view of GROOVER for the case of a tree-based architecture as defined in CD-CSGP 1. GROOVER first builds the diffusion sub-trees and then analyzes the mesh approach starting from the leaf nodes of each diffusion tree. It is possible to block the connections among diffusion trees, obtaining the performance of tree based architectures. The following paragraphs give a detailed description of the basic behavior.

Input parameters

The main input is the pdf of the node bandwidths. Other input parameters of interest are the number of nodes in $\mathcal{N}$, the number of neighbors in $\mathcal{B}_i$, the

Algorithm 1 Basic structure of GROOVER for tree based architectures

input: pdf of node bandwidth; total number of nodes; desired confidence level
output: pdf of download times;
initialization;
repeat
  while (#nodes < tot.nodes) **do** {build (diffusion) trees}
    extract the node i with $(t_i^{\text{el}} = \min_j(t_j^{\text{el}}))$ **and** (#levels < max level);
    repeat
      select randomly a neighbor from the $\mathcal{B}_i$ not yet reached by any other node;
    until node i is saturated
    if (#children = 0) **or** (node not saturated) **then**
      put node in leaf set
    else
      compute the rate r_k to each children according to
      the *max min fairness criterion*;
      for $k = 1$ to #children **do**
        assign t_k^{el};
        compute $t_i^{\text{download}} = t_k^{\text{el}} + \frac{\mathcal{F}}{C}(C-1)/r_k$;
      end for
    end if
  end while
  update histograms (time, wasted bandwidth, etc.);
until stop criterion not met

stop criterion:
update the confidence interval for the histogram including the last realization;
stop if desired confidence level is reached;

outdegree constraints $k_i^{\max}$ and $k_i^{\min}$, and the number of chunks. Finally, it is possible to specify the maximum step distance allowed for the architecture. Imposing a strict bound on the distance, we can obtain hop-driven trees.

Initialization

The root and the first generation of children are deterministically assigned the highest bandwidth. If $\mathcal{F}$ is distributed in k_o different orders to obtain a mesh, then the root selects exactly k_o children to fairly compare results with different k_o. For each child, GROOVER computes the eligibility time and assigns the download rate.

Diffusion

Each node i at level 1 (level 0 is the server) randomly selects nodes to upload to in $\mathcal{B}_i$ until its upload bandwidth is saturated, i.e., the sum of the download bandwidths of its children is greater than 80% of its upload bandwidth, or no nodes without chunks are left, provided that the constraints $k_i^{\max}$ and $k_i^{\min}$ are met. We use the threshold of 80% to avoid the selection of a new node, that can be a fast node, when the node resources are sufficiently exploited (otherwise the sum of the downloading bandwidths would be much greater than the upload bandwidth and each children would receive a small fraction of it.) Once the list of children is created, the ancestor calculates for each child i the eligibility time t_i^{el} and the rate r_i (the dimension of a chunk divided the time necessary to download it) according to the max-min fairness criterion. From the eligibility time (i.e., the time a peer finishes to download the first chunk) and the rate, we can compute the total download time of each child i:

$$t_i^{\text{download}} = t_i^{\text{el}} + \frac{\mathcal{F}}{C}(C-1)r_i \ .$$

If the node has no children, it is placed in a list for next rounds analysis.

Cross connections

Leaf nodes in the diffusion subtree are those that find no untouched nodes in $\mathcal{B}_i$. Leaves start to help their neighbors in the overlay, provided that they belong to different diffusion subtrees. The process of neighbor selection is done as in the previous case, but here the *spare* upload bandwidth of the ancestor and the *spare* download bandwidths of the neighbors are considered. For each neighbor, knowing the eligibility time and the additional rate r_i^{add}, we can calculate the new, reduced download time

$$t_i^{\text{download}} = t_i^{\text{el}} + \frac{\mathcal{F}}{C}(C-1)(r_i + r_i^{\text{add}}) \ .$$

We suppose C sufficiently high so that the difference among the starts of different contributions is not significant. Additional cross connections are realized respecting the usual constraints.

End of the realization

The realization stops when an adsorbing state is reached. In this state all the download times can be computed, as well as other rewards.

Stop criterion

The performance indices at the end of each realization are samples of known i.i.d. random variables and standard techniques can be used to estimate the confidence intervals of the whole histograms (see for instance [33]). GROOVER stops when all bins of the histograms have a $\pm 10\%$ relative confidence interval with a 0.99 confidence level.

Solution Complexity

We have not attempted to compute a-priori the number of necessary realizations to obtain the given confidence level and interval, although it

may be possible exploiting the properties of the DTMC. We use instead a-posteriori evaluation of the distributions accuracy using standard methods (see [33]). As expected, the number of realizations needed to reach a desired accuracy decreases with the number of nodes.

For instance, in the numerical examples presented in Sect. 4.5 convergence is obtained in less than 1,000 realizations for 10^4 nodes, less than 500 realizations for 10^5 and less than 200 realizations for 10^6. For 10^6 nodes and a mesh-based architecture this means 4-5 hours of CPU on a standard PC, 10–20 minutes for 10^5, while for a smaller number of nodes the execution time becomes negligible.

For tree-based architectures we also implemented a much faster algorithm that exploits the properties of the paths within the tree: we only realize a single path from the source to a leaf within the tree and use the properties of the DTMC to derive the metrics for the entire tree directly in form of aggregate histograms. For more details on this method see [27]. This further reduces the solution time by more than one order of magnitude, allowing the evaluation of tree architectures for up to 10^8 nodes. To the best of our knowledge numerical results on P2P based content distribution networks rarely extend beyond 10^3–10^4 nodes.

4.5 Numerical Results

To obtain our results, we use as input probability density function for the node bandwidth the pdf taken from [34], summarized in Table 4.1.

When reporting results, we normalize the data such that $\frac{|\mathcal{F}|}{\min(b_i)} = 1$ 'round', where $|\mathcal{F}|$ is the content size in bits and $\min_i(b_i)$ is the minimum bandwidth of the input pdf in bits/s. We use a number of chunks C equal to 100, but a sensitivity analysis with different values of C indicates a qualitative behavior independent of C, as long as $C \gg 1$. All results

Table 4.1: Bandwidth distribution used in the examples

Bandwidth	% nodes
56 kbit/s	13%
640 kbit/s	23%
1.2 Mbit/s	64%

have a confidence level 0.99 and confidence interval ±10% on the whole distribution.

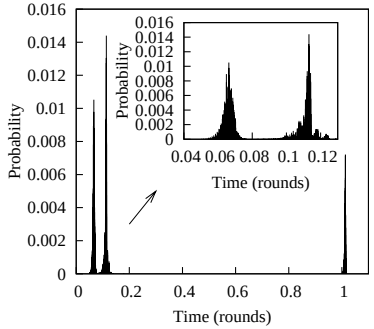


Figure 4.5: Histogram of the estimated pdf of the download time of the nodes with CD-CSGP 3 for 10^4 nodes.

Figure 4.5 shows the histogram of the estimated pdf of the node download times T_i for a network with 10^4 nodes. The distribution architecture is a mesh, modeled with CD-CSGP 3. We see that all nodes complete the download in at most one round.

While distributions like the one depicted in Fig. 4.5 are the prime output of GROOVER, we show in the following mostly aggregate results (means), which are more compact and yet convey the fundamental insights of the results we obtained.

4.5.1 Tree Based Distribution Architectures

As we have seen in Fig. 4.1, the content distribution network is built on top of the overlay network. We start by evaluating the influence of the neighbor set size in the overlay network on the delay in the content distribution network. We see in Fig. 4.6 that a neighbor set size of $|\mathcal{B}_i| = 8$ is sufficient to achieve as good a performance as for the case where in the overlay network every node knows about every other node, which corresponds to the case $|\mathcal{B}_i| = \mathcal{N},\ \forall i$.

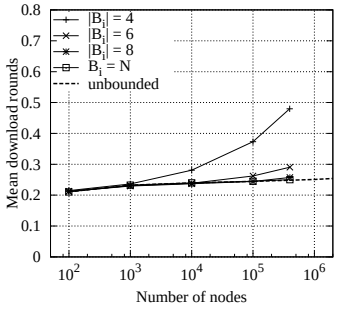


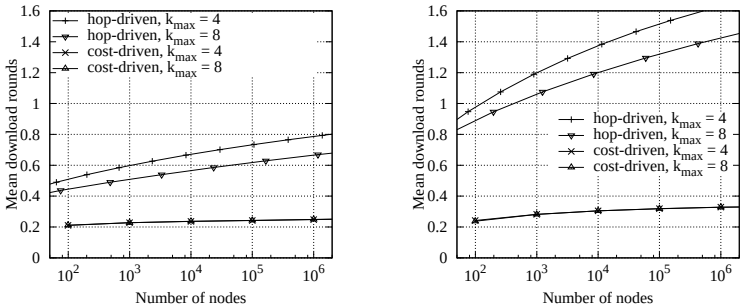
Figure 4.6: $\overline{T}$ for different number of neighbors $|\mathcal{B}_i|$ in the overlay; CD-CSGP 1 with outdegree between 1 and 4.

If the neighbor set is small ($|\mathcal{B}_i| = 4$), the mean download time grows for an increasing number of nodes much faster than for the case where $|\mathcal{B}_i| = \mathcal{N}$.

These observations are valid independent from the outdegree constraints (results are not shown here for space reasons) and the kind of content distribution tree (hop-, cost-driven). For this reason we assume for the rest of the chapter that the neighbor set is sufficiently large, i.e., $|\mathcal{B}_i| \geq 8$.

We first focus on the comparison between hop- and cost-driven trees and. The difference in the way the trees are built as well as different constraints for $k^{\max}$ and $k^{\min}$ have a major impact on the shape of the tree

and on the download performance.



(a) $k^{\min} = 1$. (b) $k^{\min} = 2$.

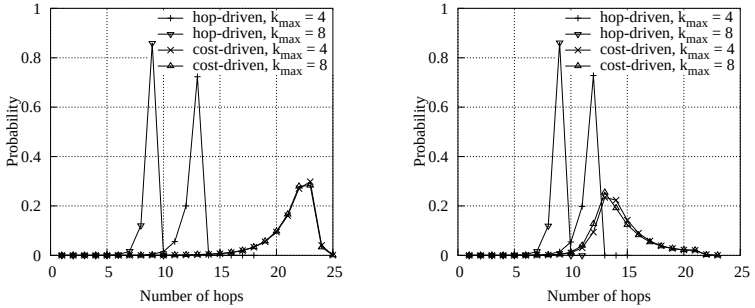
Figure 4.7: Mean download time for cost- and hop-driven trees.

Fig. 4.7 shows the mean download time as a function of the total number of nodes. We see that the mean download time is much lower for cost-driven trees as compared to hop-driven ones. This is due to fact that the cost-driven trees use the lowest eligibility time as criterion for selecting the leaf node to which a new node is attached. Therefore, subtrees with slow nodes as root will grow much slower, i.e. have very few nodes, as compared to subtrees with fast nodes. On the other hand, hop-driven trees try to keep the step distance $d^{(i)}$ of all leaf nodes approximately the same, which means that subtrees with slow nodes as root are potentially much larger than in the case of cost-driven trees.

Another parameter that has a major impact on the performance is the outdegree. We first consider the effect of $k^{\max}$: a node i can increase its number of children only if its upload bandwidth is not saturated. This means that if a node has a high upload bandwidth but receives chunks at a low rate, it can serve as many nodes as its maximum outdegree $k^{\max}$ allows. Since in hop-driven trees it is more frequent that fast nodes receive at a low rate — this concerns all the fast nodes belonging to a subtree rooted at a

slow node — the possibility to increase the outdegree helps as it allows fast nodes to use more of their upload bandwidth. In the case of cost-driven trees, fewer fast nodes will receive chunks at a low rate and the benefit of an increased outdegree is not visible.

Very interesting is also the impact of the minimum outdegree on the download performance. If we impose a minimum outdegree $k^{\min} = 2$, a slow node will divide its low upload bandwidth available by two to serve each of the two children, which will affect the speed of content propagation to the entire subtree rooted at the slow node. If we allow instead a minimum outdegree of one, this will in the case of hop-driven trees cut the mean download time into half (see Fig. 4.7). For cost-driven trees, the value $k^{\min} = 1$ will also reduce the mean download time, but not as dramatically as for hop-driven trees. This is expected, since in cost-driven trees the overall influence of slow nodes on the download times is reduced as already explained. To the best of our knowledge, the impact of the minimum outdegree on the download performance has not been discussed previously in the literature.



(a) Minimum outdegree $k^{\min} = 1$. (b) Minimum outdegree $k^{\min} = 2$.

Figure 4.8: Distribution of the step distance for 10^6 nodes.

The superior download performance of cost-driven trees as compared to

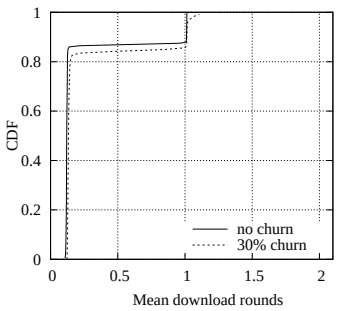
hop-driven trees comes at the cost of a greater step distance. In Fig. 4.8 we report the distribution of the step distance for different cases. As expected, the mean number of hops in cost-driven trees are higher than hop-driven tree. However the individual hops are shorter (smaller chunk download time t^{step}), so the file download time is smaller.

In the case of hop-driven trees, the choice of the minimum outdegree ($k^{\min} = 1$ or $k^{\min} = 2$) has very little influence on the distribution of the step distance, while for cost-driven trees a value of $k^{\min} = 1$ results in a much larger step distance than a value of $k^{\min} = 2$. In hop-driven trees the probability to have many consecutive hops with high rate is very low, so the outdegree of fast nodes increases after few hops and the total number of node reached becomes comparable for $k^{\min} = 1$ or $k^{\min} = 2$.

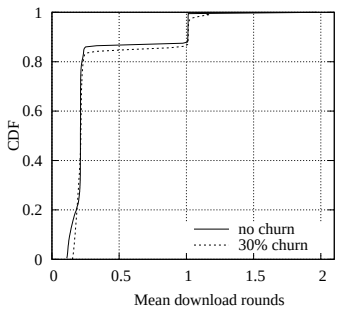
Finally, we consider the impact of churn on the file download times. We assume that when a node starts downloading the file, with a probability p_{churn} it will leave the network during the download. The time until a node leaves is selected uniformly from the interval between the beginning of the download the and estimated end of the download[4]. Fig. 4.9 shows the cumulative distribution function (CDF) of the download times with different outdegree constraints, using a cost-driven tree architecture. The CDF is built considering the nodes that have completed the download. Even in presence of 30% of nodes leaving, the download performance deteriorates only very little.

We can explain this "surprising" result as follows: In a tree structure, a high percentage of nodes are leaf nodes. However only internal nodes that leave will impact on the performance. The download time depends on the initial delay to get the first chunk and the download rate. When

[4]We talk here about estimated download time since the distribution structure is not stable and an ancestor of the node can disappear; this influences the effective download time, so it may happen that a node selects an instant for leaving the network, but in that instant it has already completed the download (if, for instance, during the download it has increased its rate).



(a) outdegree: $k^{\min} = 1$, $k^{\max} = 4$



(b) outdegree: $k^{\min} = 2$, $k^{\max} = 4$

Figure 4.9: CDF of the download times for 10^4 nodes with a churn probability of 30%.

a node i becomes orphan since its father has left, it searches among its neighbors for a node that has upload bandwidth and has downloaded a number of chunks greater than the number of chunks owned by the node i. This very simple policy automatically selects the "faster" nodes among the neighbors, preserving the download rate seen by node i. So, if the new father can offer the same download rate, a small increase in the step distance will not have a great impact. Consider also that, according to the distribution of the step distance, most of the nodes have high step distances, so the absolute difference in terms of delay until the first chunk is received is most likely low.

4.5.2 Mesh Based Distribution Architectures

Trees based architectures have the disadvantage that the leaf nodes do not at all contribute to the distribution of the content. However, when the average outdegree is larger than 2, the leaves make at least 50% of all the nodes. It is therefore natural to consider mesh based architectures. Remember that to construct a mesh, we first let the diffusion trees reach all the nodes, then nodes with spare bandwidth — typically, the leaves of

the diffusion trees — upload to nodes belonging to different diffusion trees. Diffusion trees can be hop- or cost-driven. Due to the poor performance of hop-driven trees, we consider only meshes with cost-driven diffusion trees only.

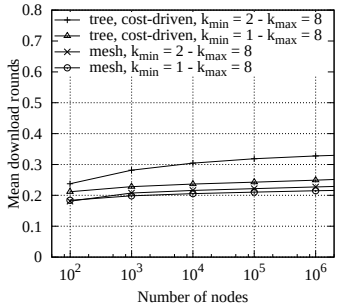


Figure 4.10: Mean download time $\overline{T}$ for cost-driven trees and mesh architectures with different outdegree constraints.

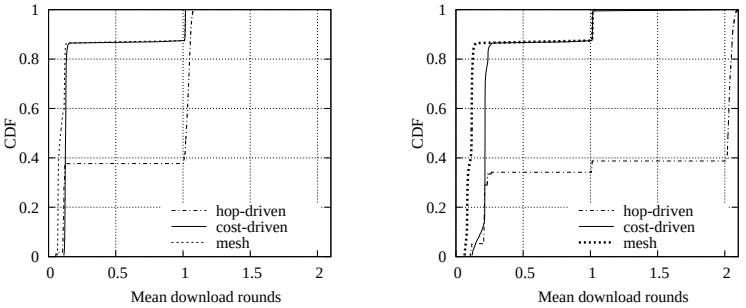
In Fig. 4.10 we compare the mean download time of meshes and cost-driven trees. We see that meshes improve the download time and that the improvement is particularly significant when the minimum outdegree is set to two. We would like to mention that another advantage of meshes over trees is that meshes are more resilient to node failures.

Table 4.2 compares meshes and cost-driven trees in terms of the amount of upload bandwidth wasted ($\overline{w^u}$) and also provides information about the mean tree depth, i.e., the average step distance. A mesh architecture reduces in all the cases we considered the wasted upload bandwidth by more than 50 percent. Another interesting result in Table 4.2 is the increase in mean tree depth by 40 to 50 percent when we reduce the minimum outdegree from two to one. For an operational content distribution system smaller step distances are attractive: At a given node churn rate, smaller step distances reduce the likelihood that a node will be get disconnected

due to such an event. If we take into consideration good download performance and robustness in case of node churn the mesh architecture with minimum outdegree two is very attractive (see also Fig. 4.11).

Table 4.2: Comparison of cost-driven trees and meshes.

		Mean Tree	Upload Bandwidth Wasted	
Outdegree	#Nodes	Depth	Cost-driven Tree	Mesh
1 - 8	10^5	17.2	46.9%	13.3%
1 - 8	10^6	21.4	47.5%	13.1%
2 - 8	10^5	12.2	66.2%	26.8%
2 - 8	10^6	14.3	68.9%	29.3%



(a) Set of 10^5 nodes with outdegree 1-8. (b) Set of 10^5 nodes with outdegree 2-8.

Figure 4.11: Comparison of the Cumulative Distribution Function of the mean download time between hop-driven tree, cost-driven tree and mesh.

Looking not only at the mean download time but also at the distribution of the download times of individual nodes can give valuable insights. We can, for instance, see to what degree the slow nodes affect the download times of the faster nodes. We consider that only 13% of the nodes are slow nodes (see Table 4.1). In Fig. 4.11 we see that for the cost-driven and mesh architecture the slow nodes do not negatively affect the download time of

the other nodes. On the other hand, in the case of hop-driven trees the slow nodes affect negatively the download times of about 50% of the other nodes.

The performance comparison of the different distribution architectures for a heterogeneous peer population gives some very interesting insights that we have not seen published elsewhere.

- Cost-driven trees and meshes allow to avoid any negative impact of slow nodes on the download performance of the other nodes. This is due to the fact that the construction process of a cost-driven tree assures that subtrees rooted at a slow node remain very small.
- A minimum node degree of one allows a slow node to achieve twice the upload rate to its only child as compared to the case where a slow node would upload to two children. While $k^{\min} = 1$ reduces the download times for all three distribution architectures, the improvement is most pronounced for hop-driven trees.

4.6 Practical Aspects

We have studied three different content distribution architectures and evaluated their performance. We think that our findings can greatly help inform and improve the design of content distribution architectures. We therefore discuss briefly how our findings can be put into practice in a real content distribution system where the distribution graph is constructed by a *distributed* algorithm and where some of the assumptions made (e.g., upload/download bandwidth of node is known) may not hold. In the following, we will sketch out the basic operation of such a distributed algorithm for tree construction that is in fact quite simple.

Since the cost-driven tree and its extension to a mesh was the most promising architecture, we will only discuss how to build in a distributed

way a cost-driven tree as defined in CD-CSGP-1 . We assume that the overlay layer (see Fig. 4.1) has been constructed and each node i knows its neighbors $\mathcal{B}_i$. Assume that the root has been selected and the root has chosen some of the nodes in its neighbor set to which it starts transmitting chunks.

If a node i gets contacted and starts receiving the first chunk, it will wait until this chunk is completely received. At that point, node i will select among its neighbors a subset of candidates:
Node i will contact the nodes in $\mathcal{B}_i$ and retain the ones that do not yet receive chunks from any other node in his candidate set $\mathcal{C}_i \subseteq \mathcal{B}_i$. Let assume for the moment that node i knows his upload bandwidth b_i^u and the download bandwidths $b_j^d, j \in \mathcal{C}_i$. Node i will open connections to its neighbors in $\mathcal{C}_i$ until it is saturated (i.e., his upload bandwidth is all used up or the constraint on the maximum number of outgoing connections is reached). It can be easily seen that this algorithm implements CD-CSGP-1, where the next node to extend the tree is the node with the minimal eligibility time, i.e. the node that just finished receiving the first chunk.

There are quite a few cases where the up- and download bandwidth may in fact be known, for instance, in a corporate environment or even in an Internet-wide P2P system. Today, many of the file-sharing tools such as BitTorrent or Edonkey allow the user to *limit* the upload bandwidth. In this case, one can set both, the available up- and download bandwidth to that value.
However, if the available up- and download bandwidth of node i are not known, node i could proceed as follows: It first opens only k^{min} connections and starts transmitting to k^{min} neighbors. If, after some time, node i observes that its upload bandwidth is not fully utilized, it can send down the subtree rooted at node i a message searching for a candidate node j whose download bandwidth is not fully utilized. The tree would be

reorganized as follows: node j disconnects from its current father and becomes a direct child of node i.

4.7 Conclusions

While the use of trees for content distribution has been studied intensively in the literature, important details such as bandwidth heterogeneity and varying node outdegrees as well as different minimum and maximum outdegrees have received very little attention, probably due to the difficulty of finding closed form results. In this chapter we have made several contributions concerning (i) the formalism to describe the content distribution graph, (ii) the methodology to compute the relevant performance metrics, and (iii) the performance results themselves.

We have defined a new formalism called Constrained Stochastic Graph Processes that is suitable for the description of content distribution architectures based on trees or meshes. This formalism specifies the evolution of the content distribution process as a semi-Markov process. To the best of our knowledge, up to now stochastic graph processes were only used to study connectivity properties and have not been applied to the performance analysis of content distribution networks. We have developed a new methodology for the performance evaluation of these Constrained Stochastic Graph Processes. Our numerical solver GROOVER emulates a random walk on the DTMC embedded in the semi-Markov process. GROOVER allows for a very efficient computation of the performance metrics even for very large number of nodes, which is clearly not feasible using standard discrete event simulation. On a PC, we can use GROOVER to evaluate distribution architectures of a million of nodes and more in just a few hours, which makes our approach very attractive for the evaluation of large P2P files distribution systems.

Using GROOVER we have obtained a number of novel results:

1. For tree-based topologies, we show that in case of bandwidth heterogeneity it is important that the nodes with low bandwidth are not at the *root of large subtrees* of nodes, leading to poor efficiency and higher download times. For this purpose, we formally introduce two types of trees, cost-driven and hop-driven trees and show that cost-driven trees perform much better than hop-driven trees, since they succeed at placing slow nodes mainly at the leaves.

2. Another parameter that has not been considered before is the minimum node outdegree. Allowing for a minimum node outdegree of 1 as compared to 2 can cut the download time into half, as it is the case for hop-driven trees.

3. Leaves do not contribute at all the to file distribution and this is normally one of the rationales to use mesh-based topologies. We have extended the tree architecture to a mesh that involves leaf nodes in the file distribution, which helps to significantly reduce the download time of high bandwidth nodes.

Chapter 5

Mesh based Streaming Services

Stochastic graph processes represent a powerful tool in analyzing P2P overlay systems. Within content distribution we can identify streaming systems, where the problem is not the minimum download time, but the distance, in terms of delay, of the nodes from the streaming source. In this chapter we apply the SGP formalism in such a context. The model captures the fundamental properties of the streaming system, such as the number of active connections, the different play-out delays of nodes and the probability of not receiving the stream due to nodes failures/misbehavior. Besides the static properties, the model is able to capture the transient behavior of the distribution graphs, i.e., the evolution of the structure over time, for instance in the initial phase of the distribution process.

5.1 Introduction

The recent success of streaming based on peer-to-peer (P2P) applications seems to achieve what traditional streaming and multicasting applications have never achieved: distributed video-on-demand and live broadcasting on the Internet. Tree based systems [7][8][9] that have been proposed earlier coexist now with more advanced mesh-based systems [10][11][12] that are more resilient to node dynamics and bandwidth variations as seen in the

Internet.

In spite of the success of P2P streaming, the fundamental properties of such systems have not been investigated in depth. In this chapter we develop a mathematical model based on stochastic graph theory that can be used to analyze fundamental performance issues of overlay streaming services. The level of abstraction of the model allows to study the fundamental behavior under different conditions, yet maintaining a limited complexity. Many studies analyze a *static* graphs that captures the properties of a snapshot of the network (see [37] and the references therein). In our work, instead, we study the *dynamics* of the graphs, i.e., the evolution of the structure over time.

We derive the master equations (MEs) that define the evolution of the streaming system in time. The MEs take into account the fundamental characteristics of the streaming protocol as well as the bandwidth available at nodes for the streaming application. The model allows assessing the impact of different protocol choices and of bandwidth heterogeneity on the delivery process and provides insights in how to improve existing streaming strategies.

The results obtained by the systematic study of different configurations show that performance is mainly influenced by the policies related to content format (how much redundant information is sent). Mesh based architectures are very robust to failures, even in presence of high churn and the delay experienced by nodes stays bounded.

The remainder of this chapter is structured as follows. Sect. 5.2 introduces mesh based streaming systems. In Sect. 5.3 we describe in detail the analytical model. Sect. 5.4 discusses the solution approach. We present the results in Sect. 5.5, and we compare them with simulations in Sect. 5.6. We conclude the chapter with some additional discussion in Sect. 5.7.

5.2 Mesh-based Overlay Streaming Systems

We do not consider here any specific system, but we identify *common basic characteristics* of recent proposals, focusing on hybrid approaches, such as [11][12], where the mesh is built as superposition of trees, obtaining a *structured mesh*. Consider an overlay network built by a P2P application. Once the overlay layer is built, paths between the source and the destinations are created following the rules of the distribution protocol. At each hop, nodes both receive the stream and contribute uploading it to other nodes, i.e., they work as content relay. Since nodes in such networks can appear or disappear frequently, the set of nodes from which a node is downloading changes over time.

5.2.1 System parameters

The content is distributed using R different stripes. Each stripe contains part of the stream (coded, for instance, using MDC techniques [39]). A node needs $R' < R$ out of R stripes to achieve a target quality, while the remaining $R - R'$ stripes contain redundant information. We assume that each node downloads a specific stripe R_i from a single node. Downloading the same stripe from multiple parents does not increase the quality of the received stream. Moreover, each node downloads only a single stripe from a given parent, even if the parent could provide multiple stripes, which limits the impact of a parent that leaves.

Even if the structure is a mesh, looking at the system at a specific instant t, it is possible to identify sub-structures inside the mesh. If we consider the graph at time t and we consider the nodes that are downloading stripe R_i, it is possible to construct a tree that connects these nodes. We call such a tree "diffusion tree." The whole mesh can be seen as a set of overlapping diffusion trees, which change over time.

The evolution of the network is subject to two main events: node arrivals and departures. We assume that arrivals and departures are exponentially distributed according to rates $\lambda(t)$ and $\mu(t)$ respectively. The dependence on time makes the model more flexible: for instance, different arrival patterns, such as flash crowds or more smooth arrivals, can be described. Let T_{str} be the duration of the stream and N the mean number of nodes receiving the stream at steady state. We consider a situation where a fraction of the nodes joins the stream at time zero, and there is an *arrival interval* during which $\lambda(t) > \mu(t)$ until steady state is reached. Fig. 5.1 shows a sample arrival pattern.

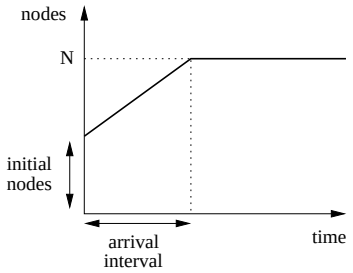


Figure 5.1: Sample arrival pattern of nodes joining a stream.

The departure rate $\mu(t)$ is the inverse of the mean time spent in the system (sojourn time). $\lambda(t)$ at steady state compensates departures. For a given time interval T, the ratio between the cumulative number of nodes that left and the mean number of active nodes during T is defined as the *churn* of the system. A 100% churn means that during T the number of the departed nodes is equal to the mean number of nodes in the system, i.e., there is on average a complete change of the nodes during T.

Nodes are divided into different classes according to their bandwidth. Each class j has an upload bandwidth $b_u^{(j)}$ and a download bandwidth $b_d^{(j)}$, which can be either symmetric, asymmetric or correlated, e.g., $b_i^u + b_i^d$

constant, as in a shared medium access. The bandwidths are random variables described by a probability density function (pdf) that is known (e.g., derived from measurement studies).

The rate of the streaming is r_{str}. We suppose that all nodes have a download bandwidth at least equal to the streaming rate. Each stripe has a rate equal to r_{str}/R', and we assume that the server is able to upload all the R stripes, i.e., it has a bandwidth greater than Rr_{str}/R'. Each node has a constraint on maximum and minimum number of active uploads that limit the possible outdegree of the node: $k^{\max}$ is the *maximum outdegree* and and $k^{\min}$ is the *minimum outdegree*.

Each node has B overlay neighbors. Among its neighbors the node selects its *parent* nodes, i.e., those from which it downloads. R' parents are called *active*; the remaining are called *standby*, since they are used as a backup in case of an active parent failure.

5.2.2 Join, Update and Leave Procedures

Nodes belonging to the initial set start building a diffusion tree for each stripe. The number of nodes in each diffusion tree depends on the characteristics of the nodes involved such as the bandwidth. Each node is involved in multiple diffusion trees.

When a new node arrives, it randomly chooses an active node as first contact and then builds its neighbor list with the help of this node. From the neighbor list, the node selects its parents and connects to them.

With rate λ_{up} nodes periodically search among their neighbors for new connections in order to increase their indegree. For standby parents, the bandwidth is not reserved, so the total number of parents can exceed the ratio between the stripe rate and the node download bandwidth.

When a node leaves, all the inbound and outbound connections are canceled. Orphan nodes try to replace the parent that has left. If the

parent that has left was in the standby set, the node does not react (it simply loses a backup parent). If the parent that has left was in the active set, the node tries to switch the state of a standby parent, i.e., it starts downloading from the standby parent that has enough available upload bandwidth. If a node has no backup parents, there will be a temporary loss of quality whose extent depends on the time necessary to search for a new parent.

5.3 System Model

The network of contacts among users of a P2P networks can be modeled as a graph, where nodes represent the users and edges the neighborhood relationship. When the users start exchanging data (in our case, they start receiving and distributing the stream) they use a subset of the available outgoing/incoming edges. The number of neighbors that are uploading to a node, for instance, represents the number of parents from which the node downloads the content, and it can be considered as a metric to measure the total rate received, and consequently the quality of the streaming. The focus of our analysis is the characteristics of the *distribution graph* (see Fig. 5.2), i.e., the subgraph of the overlay graph, where edges are the connections effectively used by nodes.

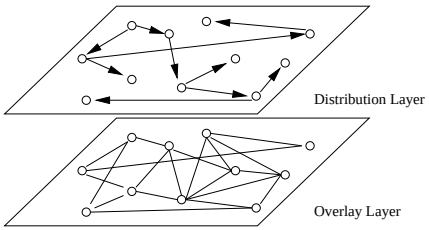


Figure 5.2: Overlay and distribution graphs

In general, the distribution graph is time varying, i.e., nodes and edges can appear or disappear in time. The evolution of the graph can be seen as a stochastic process with Markovian properties, since the graph at time $t + dt$ depends only on the graph at time t and the event (join or leave of a node) occurred during dt.

5.3.1 Formal Description of the System

Considering the system at a specific instant t, we can identify for each node two types of relationship: parents (or, equivalently, children) and neighbors. For parents we can associate an identifier of the stripe exchanged — different identifiers are selected in case of active or standby stripe.

Let α_k and σ_k be the identifier of stripe k, when it is active and standby respectively. Let η be the identifier that indicates that two nodes are neighbors in the overlay. The overlay graph and the distribution graph can thus be described by the connectivity matrix $\boldsymbol{S}$, where each element s_{ij} describes the relationship between node i and node j:

$$s_{ij} = \begin{cases} 0 & \text{nodes } i \text{ and } j \text{ are not neighbors} \\ \eta & \text{nodes } i \text{ and } j \text{ are neighbors and do not exchange stripes} \\ \alpha_k & \text{nodes } i \text{ and } j \text{ are neighbors and } i \text{ is } \textit{active} \text{ parent of } j \\ & \text{with stripe } k \\ \sigma_k & \text{nodes } i \text{ and } j \text{ are neighbors and } i \text{ is } \textit{standby} \text{ parent of } j \\ & \text{with stripe } k \end{cases} \tag{5.1}$$

Since each parent can upload only a *single stripe* to a node, s_{ij} can assume only the above values. Along with the connectivity matrix, each node has a upload bandwidth that can be represented, for a given stripe rate, as the maximum number of active children a node can have.

State transitions are determined by the events join, leave, and update described in Sect. 5.2. We assume that the rates of these events are exponentially distributed with parameters λ, μ and λ_{up} respectively. For each event it is possible to find the transition probabilities from a state $\boldsymbol{S}$ to a state $\boldsymbol{S'}$ that describes the new connectivity matrix. We will see that each event corresponds to a set of operations. As a consequence, the corresponding transition is not simple and more than one element in the connectivity matrix may change.

Join and Update

The join procedure is composed by two steps: node arrival and connection to stripes, with the latest being equivalent to an update procedure.

In the arrival procedure the arriving node b builds the neighbor set. The number of initial neighbors is equal to B. Since neighbors are randomly chosen, the new state will have a new row and a new column filled with η for each neighbor relationship. The possible transitions are given by the combination of B elements out of N total nodes with equal probability.

In the update step, a node receives from each neighbor i a vector $V_{i,b}$ containing the (active) stripes the neighbor can provide. The vector has R elements and element k contains the value of α_k if the neighbor has the stripe, otherwise it contains zero. Within each of the vectors received, the node selects randomly one of the available stripes, independently for each parent. Given the vectors, it is possible to build all the possible combinations of stripes downloadable from the whole neighbors' set. Since the selection is random and done independently neighbor by neighbor, each combination has equal probability. In the Appendix B we give the procedure that finds all the possible combinations. In the following we present a small example with $R = R' = 3$, $\alpha_k = \alpha_1, \alpha_2, \alpha_3$ and a network with 5 nodes. Assume that node 5 has just arrived and built its neighbor set: the overlay and the diffusion trees are the ones depicted in Fig. 5.3.

The connectivity matrix built upon node 5 joining the stream is repre-

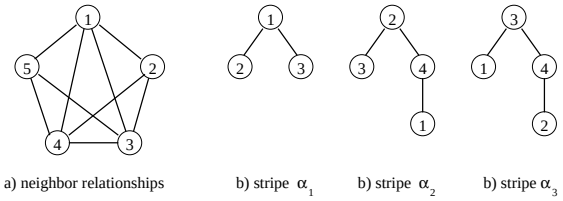


Figure 5.3: Neighbor relationships and diffusion trees of the example when node 5 joins the stream.

sented below, where column 5 identifies that node 5 is still not receiving any stripe.

$$\boldsymbol{S} = \begin{bmatrix} 0 & \alpha_1 & \alpha_1 & \eta & \eta \\ \eta & 0 & \alpha_2 & \alpha_2 & 0 \\ \alpha_3 & \eta & 0 & \alpha_3 & \eta \\ \alpha_2 & \alpha_3 & \eta & 0 & \eta \\ \eta & 0 & \eta & \eta & 0 \end{bmatrix}$$

From the definitions in (5.1), row b represents the children of node b, while column b represents the parents of node b. Node 5 now receives from its neighbors the following vectors: $V_{1,5} = [0, \alpha_2, \alpha_3]$ from node 1, $V_{3,5} = [\alpha_1, \alpha_2, 0]$ from node 3, and $V_{4,5} = [0, \alpha_2, \alpha_3]$ from node 4. Table 5.1 reports in the first row the three vectors $V_{i,5}$ received by node 5 from its neighbors 1, 3, and 4. The following rows in Table 5.1 are the four possible combinations (see Appendix B for details) of downloadable stripes.

The selection strategy depends on the protocol. Here for the sake of simplicity we assume that the node b selects randomly the stripe within the vector $V_{i,b}$ immediately upon receiving it, so the combinations are all equal likely.

Supposing that node 5 selects the last row of Table 5.1, $[0, \alpha_2, \alpha_3]$ (i.e., node 5 will download the stripe α_2 from node 3 and the stripe α_3 from

Table 5.1: Stripes node 5 can select

$[0, \alpha_2, \alpha_3]$ from node 1	$[\alpha_1, \alpha_2, 0]$ from node 3	$[0, \alpha_2, \alpha_3]$ from node 4
α_2	α_1	α_3
α_3	α_1	α_2
α_3	α_2	0
0	α_2	α_3

node 4) then the following state will be $\boldsymbol{S'}$:

$$\boldsymbol{S'} = \begin{bmatrix} 0 & \alpha_1 & \alpha_1 & \eta & \eta \\ \eta & 0 & \alpha_2 & \alpha_2 & 0 \\ \alpha_3 & \eta & 0 & \alpha_3 & \alpha_2 \\ \alpha_2 & \alpha_3 & \eta & 0 & \alpha_3 \\ \eta & 0 & \eta & \eta & 0 \end{bmatrix}$$

The transition probability is $P(\boldsymbol{S}, \boldsymbol{S'}) = \frac{1}{4}$. Defining the other 3 possible states is trivial. The last row of $\boldsymbol{S'}$, without α_k or σ_k identifies 5 as a leaf node, a state (of the node) that can be left only as a consequence of update procedures of other nodes.

Leave

The leave procedure involves multiple interactions of several nodes. When node b leaves the stream it disappears from all neighbors: the connectivity matrix $\boldsymbol{S'}$ will have all zeros in row and column b.

Each node i that was receiving an active stripe k from the node that has left tries to switch to a standby parent j (that has stripe h): the corresponding value of the connectivity matrix will change from σ_h to α_h. This is done if the standby node j can upload more stripes. Otherwise the corresponding value is switched from σ_h to zero. The identifiers α_k or σ_k

of the disappeared stripe will be set to zero wherever it appears in row i. The grandchildren of the node that has left will then try to switch standby parents too. At the end of this composition of steps, the connectivity matrix reaches a stable state that is the final transition. Note that each step is independent, so the final transition probability is easily computed.

5.3.2 Applicability of the Model

The system described in Sect. 5.2 is not meant as a proposal for a new P2P streaming protocol, but it is an abstract representation of the key features of many protocols like [11][12]. In fact all these system share a common idea of "striping" the distribution and building a structured mesh composed of the distribution trees of the single stripes. Moreover, the choice of the neighborhood and of the parents is, up to a certain degree, random, so that considering a purely random choice captures the common behavior and represents (most probably) a lower bound of performances.

The description of the overlay graph through the connectivity matrix is able to capture the essential features of overlay streaming protocols. The details of a protocol may influence different aspects of the model: (i) a protocol may impose different constraints on the structure (e.g., maximum number of neighbors, i.e. maximum number of non null elements per row); (ii) it may influence, given a state, the possible states that can be reached; (iii) it may change the transition rates. Nevertheless, the basic structure of the proposed model, the connectivity matrix and its evolution according to the protocol policies, remains unchanged. By properly translating the protocol policies and constraints into the connectivity matrix properties and transition rates, the methodology is able to provide insights into the fundamental performances that can be obtained by different overlay protocols.

5.3.3 Master Equations

The evolution of the graph that describes the overlay streaming systems is a Markov process with state space and transitions defined above. The temporal behavior can be described using the differential form of the Chapman-Kolmogorov equations, known as *Master Equations* (MEs) [37].

Let $P(\boldsymbol{S_i}, t)$ be the probability to be in state $\boldsymbol{S_i}$ at time t. The variation of the probability $P(\boldsymbol{S_i}, t)$ in time can be expressed as

$$\frac{\partial}{\partial t} P(\boldsymbol{S_i}, t) = \sum_{\boldsymbol{S_j}} w_{\boldsymbol{S_j},\boldsymbol{S_i}}(t) P(\boldsymbol{S_j}, t) \tag{5.2}$$

where $w_{\boldsymbol{S_j},\boldsymbol{S_i}}(t)$ represents the transition rates from the state $\boldsymbol{S_j}$ to the state $\boldsymbol{S_i}$ at time t. The general formulation of the Master Equations must be specialized for our problem: the transition rates are closely related to the streaming protocol policies and can be found as described in the previous section.

5.3.4 Distribution Graph Properties

The information given by the MEs are relative to the whole overlay and distribution graph. In order to analyze the system independently from the size of the network, it is useful to reorganize the information contained in state $\boldsymbol{S}$. We consider the distribution of two main performance indexes that summarize the structural characteristics of a distribution graph: the degree distribution and the delay distribution [37].

The degree distribution $P_b(d, t)$ is the probability that node b has d connections at the distribution layer at time t. We can identify both indegree and outdegree distributions ($P_b(d_i, t)$ and $P_b(d_o, t)$ respectively, with $d_i + d_o = d$), that represent the number of children and the number of parents of node b. The outdegree (indegree) distribution of a node b is

derived from the correspondent row (column) b of the connectivity matrix $\boldsymbol{S}$. We can also determine the total degree distribution defined as

$$P(d,t) = \frac{1}{N(t)} \sum_{b=1}^{N(t)} P_b(d,t) \tag{5.3}$$

where $N(t)$ is the number of nodes attached to the stream at time t.

The delay distribution represents the distance of the node from the source of the stream considering all the active stripes the node is receiving. We define $P_b(\ell,t)$ as the probability that node b is ℓ hops away from the source at time t, where ℓ is the maximum among all the stripes. Similarly to the degree distribution, we can derive the total delay distribution

$$P(\ell,t) = \frac{1}{N(t)} \sum_{b=1}^{N(t)} P_b(\ell,t) \ . \tag{5.4}$$

The probability $P_b(\ell,t)$ can be obtained from $\boldsymbol{S}$ in the following way: given a node b and a stripe k, we can recursively find the parent that is providing the stripe (through σ_k or α_k), counting the number of recursions, until we reach the root. In Appendix B we give the procedure that can be used to obtain the number of steps from $\boldsymbol{S}$.

5.3.5 Rate Equations

The MEs fully determine the evolution in time of the stochastic system. Considering the degree and delay distributions, it is also useful to have the equations for the average value. The correspondent equations are called *Rate Equations* (REs):

$$\frac{\partial}{\partial t}\overline{d} = \frac{\partial}{\partial t} \sum_{d} dP(d,t) \tag{5.5}$$

The REs express deterministically the behavior of the system, since they are a set of differential equations describing the evolution of the mean

properties. Figure 5.4 shows the relationship between the results of the MEs and the result of the REs for a given observed random variable. REs are a fluid approximation of the system. As the time goes to infinity, MEs converge to the steady state distribution if it exists, otherwise yield the transient over any given interval. REs converge to the mean value if steady state exists, otherwise they are meaningless.

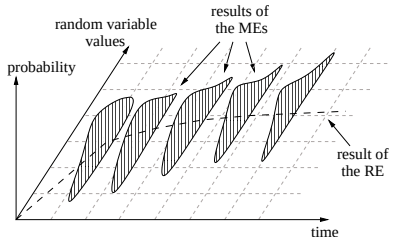


Figure 5.4: Results of the Master Equations and the Rate Equations

The methodology we propose provides the solution for the MEs, and hence the complete system characterization.

5.4 Monte Carlo Integration of the Master Equations

The set of MEs derived in the previous section are solved with the same technique based on Monte Carlo integration presented in Sect. 4.4.

The key strength of the methodology is not only its efficient numerical solution: indeed, this method provides great flexibility in the system description and specification. On the one hand, this is like a generic simulation approach, but, being based on formal definitions, avoids the risk of incomplete or bugged specifications; on the other hand, assumptions made in fluid models can be avoided, since we can describe the system behavior in full detail.

5.4.1 Comparison with Fluid Models

We consider a very simple case in order to show the differences between ME and RE based approaches. Consider the case where a node updates its indegree only during update events. We assume infinite upload and download bandwidths and no constraints on the maximum outdegree. If $d_i(t)$ is the indegree at time t, at every update event the node will add $R - d_i(t)$ parents. In fact, under these assumptions the probability to find all the necessary parents to obtain all the stripes is 1, since there is always a node that is able to provide a connection. The differential equation that describes the evolution can be written as

$$\frac{d}{dt}d_i(t) = \lambda_{\text{up}}(R - d_i(t)) - d_i(t)\mu \tag{5.6}$$

The second term considers the fact that the $d_i(t)$ parents can leave with rate μ each. Actually, not only parents can leave, but any ancestor of the node may disappear, thus the rate μ should consider also this aspect. Since we are looking for a simple closed form solution to this example, we assume that each node is able to build any stripe k starting from the set of stripes it is receiving. This assumption is unrealistic, but it simplifies the example since the failure of a node has impact only on children, not on the whole subtree. Eq.(5.6) describes the evolution of the indegree for this system. We have also modified our numerical solution including the same hypothesis in order to compare the results.

Considering the initial condition $d_i(0) = 1$ (we suppose that all nodes are present at the beginning with exactly one parent each) the solution of (5.6) is

$$d_i(t) = \frac{\lambda_{\text{up}}R}{\lambda_{\text{up}} + \mu}\left(1 - e^{-(\lambda_{\text{up}}+\mu)t}\right) + e^{-(\lambda_{\text{up}}+\mu)t} \tag{5.7}$$

In Fig. 5.5 we compare the analytical solution of this very simple case with the solution of the Rate Equations (5.5) derived from our model. We

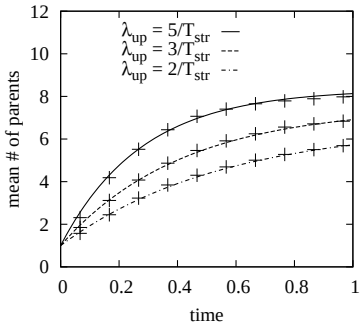
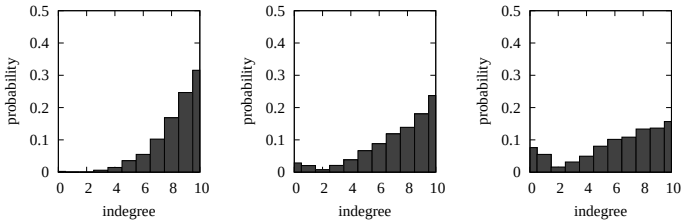


Figure 5.5: Solution of the differential equation and the Rate Equation.

set $R = 10$ stripes, $\mu = 1/T_{\text{str}}$ and $\lambda_{\text{up}} = \frac{5}{T_{\text{str}}}, \frac{3}{T_{\text{str}}}$ and $\frac{2}{T_{\text{str}}}$. We normalize the time with respect to T_{str}. The numerical solution follows closely the analytical one. But the results obtained from our model give more insight. In fact, we can observe how the *full* indegree distribution changes over time.



(a) $\lambda_{\text{up}} = \frac{5}{T_{\text{str}}}$ (b) $\lambda_{\text{up}} = \frac{3}{T_{\text{str}}}$ (c) $\lambda_{\text{up}} = \frac{2}{T_{\text{str}}}$

Figure 5.6: Indegree distribution at time $T_{\text{str}}/2$ obtained from the solution of the MEs.

Fig 5.6 shows for instance the distribution of the number of parents (indegree) at time $T_{\text{str}}/2$ for different values of λ_{up}.

Notice that there is a non-null probability that a node remains without parent, thus being disconnected entirely from the distribution process, a

phenomenon that a fluid approach analyzing the means entirely disregards, while in most cases it is one of the most important performance.

5.5 Application of the Methodology

5.5.1 System Description

We use a configuration with $N = 10^4$ nodes, but we have also checked some configurations with 10^5 nodes obtaining similar results. We use the input bandwidth distributions reported in Tables 5.2 and 5.3; bandwidths are expressed as a multiple of the streaming rate r_{str}. The streaming rate is divided into R' stripes and the source generates R stripes. Results are obtained for $R = 12$ and $R' = 3, 6, 9$.

Table 5.2: Upload bandwidth distribution A (normalized w.r.t. r_{str})

Bandwidth	% nodes
1	20%
2	40%
5	40%

Table 5.3: Upload bandwidth distribution B (normalized w.r.t. r_{str})

Bandwidth	% nodes
1	40%
2	60%

We consider an observation time equal to T_{str} (stream length). We consider two arrival patterns, with an initial number of nodes equal to $\frac{N}{10}$ and $\frac{N}{2}$ respectively; the remaining nodes arrive within $T_{\text{str}}/5$. The mean sojourn time is set to $0.5T_{\text{str}}$, T_{str}, and $2T_{\text{str}}$.

Each node can have up to 60 neighbors in the overlay graph (the actual number of neighbors depends on dynamics of the nodes); among these relationships, while uploading a node can have a maximum outdegree that is limited only by its bandwidth[1].

The stream is chunk based (e.g., few video frames or a slice of a few tens of milliseconds of sound) and we normalize the dimension of the chunk, U, such that $\frac{U}{r_{\text{str}}} = 1$ unit. A node can upload the content after a delay equal to the download time of a single chunk. So the delay can be considered as the "distance" (relative delay) of the node from the source of the stream. The length of the stream, T_{str}, is set to $10000\frac{U}{r_{\text{str}}} = 10000$ units.

Besides degree and delay properties, we consider also the *quality of the mesh*: when a node remains orphan of an active parent, it switches to one of its standby parents: if they have enough bandwidth to help the node, the node has no service disruption; if no standby parent is able to help the node, it must search for a new parent, with a possible service disruption. We measure the quality of the mesh as the percentage of nodes that successfully switch to standby parents.

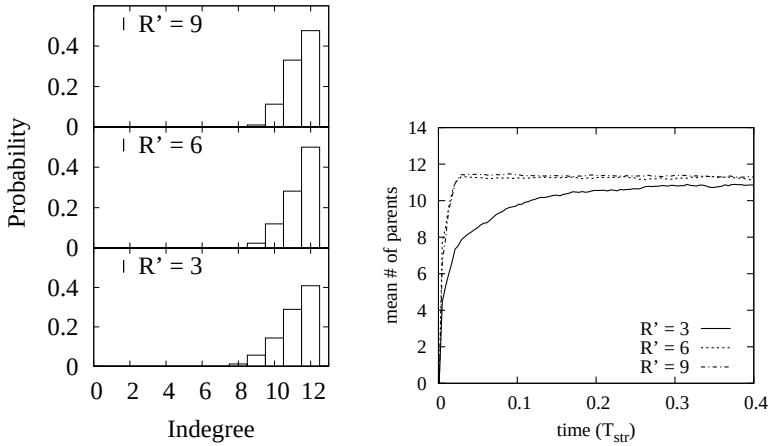
5.5.2 Analysis of the Indegree

Analyzing the indegree we examine whether the subdivision in stripes helps the distribution process or not. On one hand, more stripes means that each stripe has a lower rate, so the loss of a single stripe has less impact. On the other hand, each node must maintain more active connections and the probability that any one of these connections fails increases.

Figure 5.7(a) shows the indegree distribution of the nodes at time $t = T_{\text{str}}$, computed with Eq. (5.3) using distribution A. In this case we have an initial number of nodes equal to $N/10$ and mean sojourn time T_{str}. The

[1]The bandwidth is not necessarily the physical bandwidth, but can be the amount of resources willingly shared with the stream.

distribution tends to peak around R independently from R'. This means that all the nodes in the network are able to receive the full quality, since the degree is always greater or equal to R'. Note that with $R' = 9$ there is a fraction of the nodes with exactly 9 parents: this means that, in case of one parent that has left, the quality received by the node may be temporarily affected. For smaller values of R' all nodes always receive at least one redundant stripe, which makes them less vulnerable to disruptions.



(a) Prob. Distr. at T_{str} (b) Evolution in time

Figure 5.7: Solution of the MEs for the indegree (initial number of nodes: $N/10$; sojourn time: T_{str}, bandwidth distribution A).

The temporal behavior of the indegree can be analyzed looking at the results of the rate equations (Fig. 5.7(b)) computed with Eq. (5.5). A stable value is reached quickly with the only exception of $R' = 3$: this means that the structure, even in presence of high churn is able to maintain a high quality of the stream.

Figure 5.8 shows the indegree distribution of the nodes at time $t = T_{\text{str}}$ using distribution B. Besides the case with $R' = 3$, the distribution is

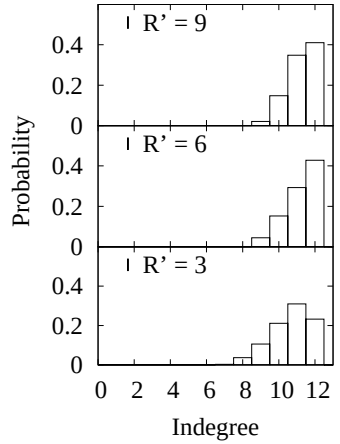


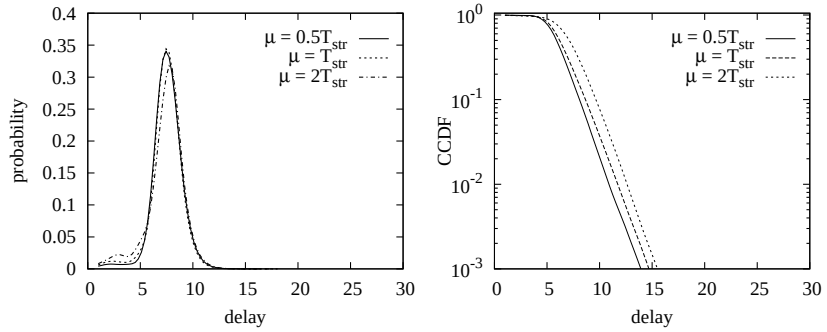
Figure 5.8: Probability Distribution of the indegree at T_{str} for different R' (initial number of nodes: $N/10$; sojourn time: T_{str}, bandwidth distribution B).

very similar to the case with input bandwidth distribution A; this means that the architecture is very robust with respect to the input bandwidth distribution.

5.5.3 Analysis of the Delay

The delay, expressed as time units, represents the number of hops from the source. We plot the probability density function of the delay. We consider $R' = 6$ and we set different sojourn times (μ). Fig. 5.9(a) shows the case of initial number of nodes equal to $N/2$ and bandwidth distribution A. The distribution is not affected by the different values of μ. Similar results are obtained with the other configurations.

In Fig. 5.9(b) we show in details the tails of the distributions (the figure shows $R' = 3$, but similar results are obtained for different configurations): to this aim we plot the Complementary Cumulative Distribution Function



(a) $R' = 6$, distribution (b) $R' = 3$, CCDF

Figure 5.9: Distribution of the delay with different sojourn times (initial number of nodes: $N/2$, bandwidth distribution A).

(CCDF). The behavior of the tails does not depend on the sojourn time.

In Fig. 5.10 we show the impact of R' on the delay. Increasing the number of stripes has a price: since each node needs all the R' stripes to correctly play the stream, the absolute delay is given by maximum delay among the stripes. By increasing the number of stripes, the probability to have higher delays increases, since we have to compute the maximum among an increased number of stripes.

The same analysis is made for input bandwidth distribution B. In Fig. 5.11 we show in details the CCDF for different values of R'. It is possible to note that results are similar to those obtained with input bandwidth distribution A.

5.5.4 Analysis of the Quality

Aggregate results for the indegree and the delay are not able to capture all the aspects related to the quality of the received stream by a generic node i. In Table 5.4 we summarize other results that can be obtained from

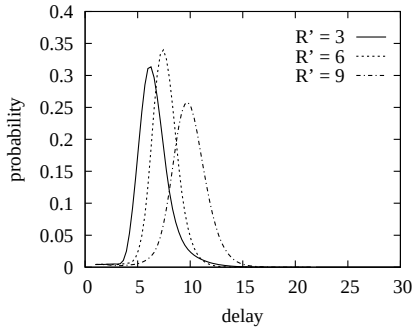
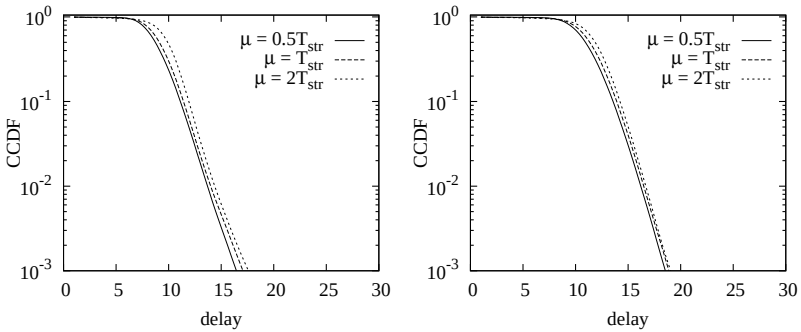


Figure 5.10: Distribution of the delay for different R' (initial number of nodes: $N/2$, bandwidth distribution A).



(a) $R' = 6$ (b) $R' = 9$

Figure 5.11: CCDF of the delay with different sojourn times (initial number of nodes: $N/2$, bandwidth distribution B).

the solution of the MEs. The value of the *churn* is computed according to the arrival pattern: arrivals and departures are Poisson processes with rate $\lambda(t)$ and $\mu(t)$ respectively, so we can calculate the cumulative number nodes that have left at time T_{str} and consequently the value of churn.

Table 5.4: Other statistics.

		distr. A		distr. B	
R'	$1/\mu$	% Churn	%Switch	% Churn	%Switch
6	$0.5T_{\text{str}}$	186.8%	99.6%	186.3%	97.8%
6	T_{str}	93.5%	99.7%	93.5%	97.9%
6	$2T_{\text{str}}$	46.8%	99.8%	46.8%	97.5%
9	$0.5T_{\text{str}}$	187.3%	94.7%	186.4%	87.2%
9	T_{str}	93.2%	97.9%	93.4%	90.1%
9	$2T_{\text{str}}$	46.7%	99.1%	46.8%	91.1%

One of the performance index monitored is the probability to switch to a standby parent if an active parent leaves. This is given by $p(k_i, t)$ with $k_i < R'$. Integrating over time t we are able to compute the switch probability (see columns 4 and 6 of Table 5.4). With a small R', the percentage of switches is very close to 1, i.e., the received stream is stable. On the other hand, with R' near to R, with high churn, if the number of parents of a node n drops below R', the probability to switch to a standby parent is 94% for bandwidth distribution A, and 87% for bandwidth distribution B. This means that the quality temporarily decreases, as expected looking at degree distribution (Fig. 5.7(a) for bandwidth distribution A). Note that, with $R' = 9$, even with stable nodes, the configuration with bandwidth distribution B has a probability to switch significantly below 1.

5.6 Comparison with Simulations

In order to validate our analytic model and our assumptions — mainly exponential distributed times — we implement a simple overlay streaming protocol on top of the PeerSim P2P network simulator [42]. PeerSim is a Java based simulator that consists of many configurable components: it has two types of engines, cycle-based and event-driven, and different modules that manage the overlay building process and the transport characteristics. For a more detailed description of PeerSim simulator the interested reader is referred to [42].

5.6.1 Protocol Description and Simulation Set Up

We implemented the overlay streaming protocol using the event-driven engine. The protocol does not contain all the features of a real system, but it captures the essential behavior of the management of the distribution structure.

In the following we give a high level view of the protocol messages, leaving out details about the management of all the situations. The basic control messages exchanged by nodes are:

- *Join*: when a node joins the network, it obtains a list of neighbors from a *rendevouz server* and it sends messages to a subset of the received list of neighbors asking for stripes; with this message, a node asks to its neighbors to attach to a stripe.

- *Leave*: when a node decides to leave, it informs its neighbors; the messages are sent out after a random interval (different for each neighbor). This behavior is equivalent to have neighbors that periodically send to the node ping messages in order to check if it is still online.

- *Switch*: when a node remains orphan of an active parent, it sends a message to its standby parents asking to switch the status.

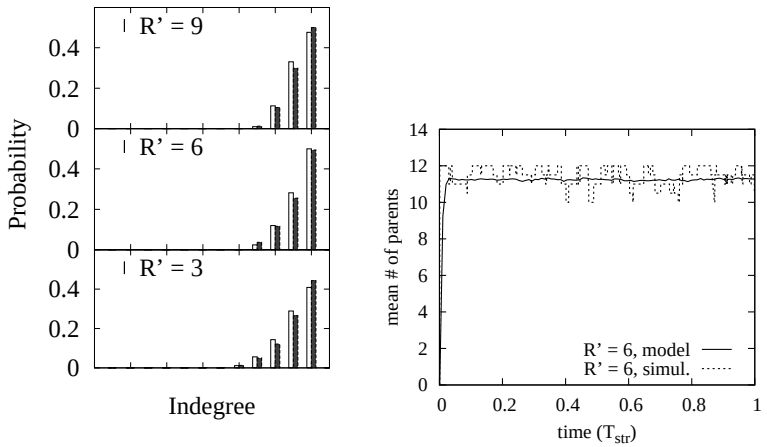
After receiving message, a node processes it determining, for instance, the availability of the bandwidth or the delay from the source, and replies with a message containing the requested parameters. Besides these procedures, a node periodically schedules an *Update* where it sends *Join* messages in order to increase its connectivity.

When a node joins the network, it selects a lifetime uniformly distributed between zero and twice the sojourn time used in the model, i.e. with a mean equal to the sojourn time used in the model. This distribution is used to check the impact of the hypothesis of exponentially distributed sojourn times we made in the model. At the transport layer, each message experiences an end-to-end delay that is uniformly distributed between a minimal and maximal value. The other constraints concerning the minimum and maximum number of children, number of stripes, or initial number of nodes are the same as in the model.

5.6.2 Simulation Results

We consider the impact of the number of stripes on the connectivity, i.e., how many stripes a node is able to receive when stripes have different sizes. Figure 5.12(a) shows the probability density function of the indegree at the end of the stream, compared with the results obtained with the model. We observe that results are very close, i.e., the impact of the assumption we made in the model is low.

Looking at the evolution in time of the outdegree, in Fig. 5.12(b) we compare the mean number of parents obtained from the model and the number of parents of a realization obtained from a simulation. As we can see, the model is able to closely predict the behavior of the system.



(a) Prob. Distr. at T_{str}

(b) Evolution in time ($R' = 6$)

Figure 5.12: Results obtained by simulation for the indegree (initial number of nodes: $N/10$).

We then consider the distribution of the delays, i.e., the distance of the nodes from the source. Figure 5.13 shows the distribution of the delay for different values of R': continuous lines are the results predicted by the model, while dashed lines with points are the results of simulations. It is possible to see that the behavior of the system remains the same.

For all the experiments, we check also configurations where we add a delay in the message transfer uniformly distributed between 0 and 1 (1 unit is the time necessary to transfer a single chunk). This delay (results not shown here) does not have an impact on the final performance. Moreover, we consider different scenarios where a node reacts in different ways in case of an active parent leaves and the node has no standby parents. In this case, the node, instead of waiting for the next *Update* event, can look for new parents. Results shows that the distribution of the number of parents peaks around the maximum number of parents, and the overall quality

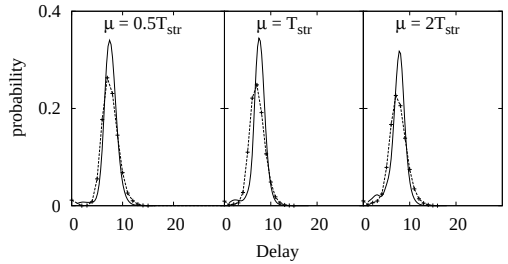


Figure 5.13: Distribution of the delay obtained by simulation with PeerSim (initial number of nodes: $N/2$; $R' = 6$).

increases. Nevertheless, the number of additional messages represents an overhead for the network. From the comparison between simulation and analytic results we can conclude that our analytic model is able to capture the essential performance characteristics of the overlay streaming systems.

5.7 Discussion and Conclusions

In this chapter we introduced a novel methodology for the high-level representation of overlay streaming systems. The proposed methodology does not represent only a model for a specific protocol. Actually, the general definition of the structure is flexible and can be adapted to a *generic* overlay streaming protocol that builds structured mesh.

Based on the use of Master Equations, the solution of the model yields the entire probability distribution and not only the mean, of the metrics of interest (node degree or delay) as well as the temporal (transient) dynamics.

We have modeled some systems proposed recently obtaining novel insights in the dynamics of self-organizing systems for streaming distribution. In the following we summarize the main findings that can help in designing

better P2P streaming systems.

- Redundant stripes play a fundamental role in obtaining good performances. Recent proposals [11][13] consider only a small fraction of redundant information so, in case of node departures, the streaming is vulnerable to disruptions.
- The delay is influenced by stripe 'size': the greater R' (smaller stripes) the higher the delay. The number of necessary stripes R' should be kept low to keep a low delay. The delay remains low independently from the dynamics of the network.
- Under medium to high churn, nodes may experience a poor quality. Only stable nodes can prevent this behavior. This performance measure cannot be computed with any methodology that only yields averages.

The model can be extended in different ways in order to study different scenarios. For instance, we can consider *bandwidth fluctuations*, that model the unstable behavior of nodes. When the bandwidth decreases, a node simply drops some of the stripes. When the bandwidth increases, the node can accept new children when other nodes perform the update procedure. Another interesting extension is considering different policies for selecting the stripes from neighbors: instead of choosing randomly neighbor by neighbor, a node may collect all the stripes a set of nodes can give, selecting the combination of nodes that maximize the number of received stripes.

Chapter 6

Conclusions and Perspectives

The *self-scaling* and *self-organizing* properties of peer-to-peer networks allow the quick and efficient distribution of content to large client populations. Cooperative distribution techniques capitalize on the bandwidth of every peer to offer a service capacity that grows linearly with the number of peers, provided the blocks among the peers are exchanged in such a way that the peers are busy most of the time.

In this thesis we have analyzed the case of file distribution and streaming services. In the following we briefly summarize the contribution of the thesis.

- In case of ideal conditions — with stable peers that know the address and the bandwidth of the other peers — our results 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 and PTree 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.

- In case of distribution architectures with constraints, when the neighbor bandwidth is not known, we introduced an analytical model that provides a closed form solution assuming no correlation among successive distribution steps. The model proved to be very accurate due to a weak correlation structure of the distribution process. The results obtained have been validated against simulations. Both the analytical model and the simulations yield the pdf of the distribution process, as a function of time and as a function of the number of nodes reached. The impact of dynamically modifying the degree of the distribution tree has been quantified as a function of the upper and lower bounds of the degree and a thorough analysis has been carried out.

- The most important contribution of the thesis is the definition of a new analytical technique called Constrained Stochastic Graph Processes, which can be applied to general distribution architectures, different protocols, and heterogeneous scenarios to analyze properties and performances. This formalism specifies the evolution of the content distribution process as a semi-Markov process. To the best of our knowledge, stochastic graph processes were only used to study connectivity properties and have not been applied to the performance analysis of content distribution networks. We have developed a new methodology for the performance evaluation of these Constrained Stochastic Graph Processes. Our numerical solver GROOVER emulates a random walk on the DTMC embedded in the semi-Markov process. GROOVER allows for a very efficient computation of the performance metrics even for very large number of nodes, which is clearly not feasible using

standard discrete event simulation.

- We successfully applied the Stochastic Graph process formalisms to streaming systems: based on the use of Master Equations, the solution of the model yields the entire probability distribution of the performance metrics (not only the mean values), as well as the temporal (transient) dynamics. We have modeled some systems proposed recently obtaining novel insights in the dynamics of self-organizing systems for streaming video.

Appendix A

Stochastic Graph Processes and Chemical Kinetic Systems

A.1 Similarities with Chemical and Physical Systems and Convergence Properties

Monte Carlo integration method has been widely used in physical and chemical sciences. Looking at the problems studied by these communities we can find many similarities with the file distribution problem, opening new possibilities for the study of complex distributed systems.

Consider for instance these two different systems: a chemical kinetic system that contains millions of molecules, and a physical system described by the positions and the spin of a material composed by millions of particles.

In the chemical system, assume that the system is stable and we introduce a new type of molecule that is able to react with the molecules present in the system. The perturbation of the system starts to involve more and more molecules and we reach a new stable situation when all the reactions have occurred. Besides the macroscopic description of the new stable state, physical chemistry is interested in the characterization of the evolution of the system over time. The speed of the evolution depends, for instance, on physical properties of the molecules and on the single reaction

speed.

In the physical system, assume that all particles have random spin and we introduce a small magnetic field in a particular point of the system space. Under this magnetic field, the spin of the adjacent particles will start changing the polarization. This in turn will modify the magnetic field around them and the result is a perturbation that will eventually reach all the particles. Again, besides the final configuration of the system, in physics it is also interesting to study the evolution of the polarization. The speed of the magnetic perturbation depends, for instance, on physical characteristics of the particles that determine how they interact one another.

Both examples show how a system with million of elements, molecules or particles, switches from one stable state to another state and this change can be observed at microscopic level.

The distribution of a file among users can be modeled as the propagation of the file over the network. All the nodes will eventually receive the file, but the interesting problem is the characterization of the time evolution of the process. We compare the node that has not yet received the file to the molecule that is not yet involved in the reaction or to the particle that is not yet reached by the magnetic field.

The methodologies used to analyze chemical and physical systems can also be applied for the file distribution process. The informal explanation on how these systems can be compared is made formal in the following paragraphs. The basic idea is to write formally the state space that describes the stochastic process. The fact that we are able to obtain a formal description of the system equivalent to the formal description of a physical or chemical system makes the systems equivalent and so solvable with the same methodologies.

Formal Description of the System

In order to simplify the description, we made some hypotheses on the model. We suppose that the network is fully connected, i.e., every node can reach all the other nodes. We have shown that, if the number of neighbors is above 10, the performance are equivalent, so this assumption does not have a great impact.

We discretize the time into equal intervals of width Δt, which are labeled with index k (t_k is the k-th interval).

We define $N(t_k, b_i, r_j)$ as the number of nodes that in interval t_k has bandwidth b_i (we consider here the symmetric case), complete the download of the first chunk with rate r_j and has not yet tried to upload to any other nodes (*ready to start uploading*). We define $N'(t_k, b_i, r_j)$ as the number of nodes with the same characteristics as for $N(t_k, b_i, r_j)$, but they have already started to upload the content (*uploading nodes*). As we did for the time, we discretize the rate, from 0 to b_i and we refer to r_j as the j-th rate.

We identify the interval t_k with a vector $\vec{r}_{t_k}$ that collects all the values $N(t_k, b_i, r_j)$ and $N'(t_k, b_i, r_j)$ (ordered according to a simple scheme, from the lowest b_i and r_j to the biggest ones).

The complete state space $\boldsymbol{S}$ that describes entirely the system is the set of vectors that identify intervals, i.e.,

$$\boldsymbol{S} = \{\vec{r}_{t_0}, \vec{r}_{t_1}, ...\vec{r}_{t_{k_{\text{MAX}}}}\}$$

It is easy to show that this state is a description of the weighted graph of tree based architectures (with nodes that do not leave), where the weights are the eligibility times, here discretized with the intervals t_k. Even if connections among nodes are not taken into account in the description of the state, there is no loss of information since the connections among internal nodes do not influence the possible transition among states: in

fact, the evolution of the chain depends on the leaf nodes that, when they become eligible, involve other nodes. The already established connections do not influence this evolution.

Let $P(\boldsymbol{S}_i)$ be the probability to be in state $\boldsymbol{S}_i$. We consider the transition probability matrix Q whose elements $Q(\boldsymbol{S}_i, \boldsymbol{S}_j)$ represent the transition probability from state $\boldsymbol{S}_i$ to state $\boldsymbol{S}_j$.

The Chapman-Kolmogorov equations, which are called Master Equations in chemical and physical sciences, of the system are:

$$P(\boldsymbol{S_i}, t+1) = P(\boldsymbol{S_i}, t)\left[1 - \sum_{\substack{k=1 \\ k\neq i}}^{M} Q(\boldsymbol{S_i}, \boldsymbol{S_k})\right] + \sum_{\substack{j=1 \\ j\neq i}}^{M} P(\boldsymbol{S_j}, t)Q(\boldsymbol{S_j}, \boldsymbol{S_i}) \quad \text{(A.1)}$$

In each state, the one-step transition probabilities are determined by the building rules (in- out- degree constraints, etc.). The transition probability matrix Q is the collection of all the state transition probabilities.

For a given state $\boldsymbol{S}_i$, for each of the vectors $\vec{r}_{t_k}$ with the smallest t_k, we can find all possible transitions, with the corresponding probabilities. In order to show how these probabilities can be found, we introduce the following operator W derived from [30]

$$W^l_{N(t_k,b_i,r_j)}\boldsymbol{S} = W^l_{N(t_k,b_i,r_j)}(\ldots, N(t_k, b_i, r_j), \ldots) = (\ldots, N(t_k, b_i, r_j) + l, \ldots)$$

which means that the number of nodes with bandwidth b_i that at time t_k are downloading at rate r_i increases by l (l can be positive or negative). As an example, consider a two-class network with minimum and maximum outdegree equal to $k^{\min} = 1$ and $k^{\max} = 2$. Denote $r_1 = b_F$ (fast) , $r_2 = b_F - b_S$ (medium), and $r_3 = b_S$ (slow) . Assume that $\frac{\mathcal{F}}{Cr_i} = \delta_i$, where $\mathcal{F}$ is the file size, C is the number of chunks, and δ_i is the time necessary for downloading the chunk at rate r_i, i.e., the one step eligibility time at rate r_i. Consider this transition

$$W^{+1}_{N(t_k+\delta_1,b_F,r_1)} W^{+1}_{N'(t_k,b_F,r_1)} W^{-1}_{N(t_k,b_F,r_1)} \boldsymbol{S}$$

This means that a fast node successfully uploads to another fast node: there is a decrease in the number of N and an increase in the number of N' and a new fast node that will become ready to start uploading at time $t_k + \delta_1$.

For completeness, we write here all the remaining transitions, omitting the transition to non-active nodes for notation simplicity.

$$
\begin{aligned}
W^{+1}_{N(t_k+\delta_3,b_S,r_3)} W^{+1}_{N(t_k+\delta_2,b_F,r_2)} W^{-1}_{N(t_k,b_F,r_1)} \boldsymbol{S} \\
W^{+2}_{N(t_k+\delta_3,b_S,r_3)} W^{-1}_{N(t_k,b_F,r_1)} \boldsymbol{S} \\
W^{+1}_{N(t_k+\delta_2,b_F,r_2)} W^{-1}_{N(t_k,b_F,r_2)} \boldsymbol{S} \\
W^{+1}_{N(t_k+\delta_3,b_S,r_3)} W^{+1}_{N(t_k+\delta_2,b_F,r_2)} W^{-1}_{N(t_k,b_F,r_2)} \boldsymbol{S} \\
W^{+2}_{N(t_k+\delta_3,b_S,r_3)} W^{-1}_{N(t_k,b_F,r_2)} \boldsymbol{S} \\
W^{+2}_{N(t_k+\delta_3,b_F,r_3)} W^{-1}_{N(t_k,b_F,r_3)} \boldsymbol{S} \\
W^{+1}_{N(t_k+\delta_3,b_S,r_3)} W^{+1}_{N(t_k+\delta_3,b_F,r_3)} W^{-1}_{N(t_k,b_F,r_3)} \boldsymbol{S} \\
W^{+2}_{N(t_k+\delta_3,b_S,r_3)} W^{-1}_{N(t_k,b_F,r_3)} \boldsymbol{S} \\
W^{+1}_{N(t_k+\delta_3,b_F,r_3)} W^{-1}_{N(t_k,b_S,r_3)} \boldsymbol{S} \\
W^{+1}_{N(t_k+\delta_3,b_S,r_3)} W^{-1}_{N(t_k,b_S,r_3)} \boldsymbol{S}
\end{aligned}
$$

The probability associated to these transitions can be simply derived from the input pdf of the bandwidths. If more than one node is active in the same interval t_k we have a complex simultaneous transition toward all possible states generated by the nodes. Nevertheless, since each transition is independent from the others, we can compose them using simple transition considering a single node at a time.

The above formulation of the system completely describes the stochastic process. The interested reader can verify that, by assigning different variable meanings, the problem formulation becomes one that has been studied in natural sciences. We use a similar notation and approach to system description as presented in [30] (with some little extensions) so it is simple to compare the two systems.

Appendix B

Overlay Streaming Systems: Procedures

This appendix contains the formal definition of the procedures used to find the transitions between a state $\boldsymbol{S}$ and a new state $\boldsymbol{S'}$ in case of *Join* or *Update* event, as well as the procedure to find the delay of a node b given a connectivity matrix $\boldsymbol{S}$.

B.1 Join and Update

A node receives the vectors containing the stripes of its neighbors. The output of the decision process is a vector of R elements (R is the number of stripes), where element k contains the neighbor from which the node download stripe k (the first R' will be active, the remaining standby).

In order to find all the possible combinations of neighbors that can provide the stripes, we use as a basic building block the procedure that is able to find all the permutations of R objects taken from a set of B objects, where B is the number of neighbors. The number of available permutations is $(B)_R = B!/(B-R)!$, so the output is a matrix E with $(B)_R$ rows and R columns. Each element e_{ij} contains the neighbor that will provide stripe j in the combination i. If, looking at the vector provided by neighbor e_{ij},

we found a zero at position j, then the element e_{ij} is set to NULL. After this operation we obtain a new matrix M'.

The matrix M' is then reduced eliminating all the rows that are *equivalent* or *contained* in other rows. Row a is *equivalent* to row b if they have exactly the same non-null elements in the same positions. Row a is *contained* in row b if row b has the same non null elements of row a (in the same positions) and one or more other non null elements that row a does not have.

The final matrix M'' contains all the possible combinations neighbor-stripe that the node can select. Note that, with the Monte Carlo integration methodology we use, it is not necessary to compute the entire matrix M'' (it would be computationally expensive): since we perform a realization of the process, it is sufficient to generate a random row of the matrix for each realization[1].

B.2 Computing the Delay

In order to find the delay, in terms of number of steps from the root, we start from the connectivity matrix $\boldsymbol{S}$. Given a node i the procedure to find the delay is described in Algorithm 2. The procedure "find_stripes(stripe_id, column_index)" returns the row index where the stripe is. The source node has index zero, so when we reach the source the procedure stops. This procedure is done for all nodes except the source.

[1]Consider the equivalent problem of generating all the possible permutations of a set of B elements, compared to the cost of generating a random permutation.

Algorithm 2 Procedure for finding the number of steps

```
input: connectivity matrix S, initial node i;
output: number of steps from root;
#steps = 0;
#parents = 0;
for α_k = α_1, α_2, ..., α_R do
   j = find_stripe(α_k, i);
   while j ≥ 0 do
      #parents++;
      i = j;
      j = find_stripe(α_k, i);
   end while
   #steps = max(#steps, #parents);
end for
return(#steps);
```

Bibliography

[1] A. Parker, "True Picture of File Sharing", Cache Logic Report 2004, Available: `http://www.cachelogic.com/home/pages/research/p2p2004.php`

[2] Z. Ge, D. Figueiredo, S. Jaiswal, J. Kurose, and D. Towsley, "Modeling Peer-Peer File Sharing Systems," in *Proc. IEEE INFOCOM*, San Francisco, California, USA, Mar. 2003.

[3] X. Yang and G. de Veciana, "Service Capacity of Peer-to-Peer Networks," in *Proc. IEEE INFOCOM*, Hong Kong, Mar. 2004.

[4] F. Clevenot and P. Nain, "A Simple Fluid Model for the Analysis of the Squirrel Peer-to-Peer Caching System," in *Proc. IEEE INFOCOM*, Hong Kong, Mar. 2004.

[5] S. Iyer, A. Rowstron, and P. Druschel, "Squirrel: a Decentralized Peer-to-Peer Web Cache," in *Proc. ACM Symposium on Principles of Distributed Computing (PODC 02)*, Monterey, California, 2002.

[6] D. Qiu and R. Srikant, "Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks," in *Proc. ACM SIGCOMM*, Portland, OR, Sept. 2004.

[7] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "ALMI: An Application Level Multicast Infrastructure," in *Proc. of the 3rd Usenix Symposium on Internet Technologies & Systems (USITS)*, Mar. 2001.

[8] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," in *Proc. SIGCOMM 2002*, Aug. 2002.

[9] AD. A. Tran, K. A. Hua, and T. T. Do, "A Peer-to-Peer Architecture for Media Streaming," in *IEEE JSAC: Special Issue on Advances in Overlay Networks*, Vol.22, N.1, Jan. 2004.

[10] Y.-H. Chu, S. G. Rao, and H. Zang, "A Case for End System Multicast," in *Proc. of ACM SIGMETRICS 2000*, June 2000.

[11] X. Zhang, J. Liu, B. Li, and T. S. P. Yum, "DONet/CoolStreaming: A Data-driven Overlay Network for Live Media Streaming," in *Proc. IEEE INFOCOM 2005*, Miami, Mar. 2005.

[12] N. Magharei, R. Rejaie, "Understanding Mesh-based Peer-to-Peer Streaming," in *Proc. NOSSDAV 2006*, Newport, Rhode Island, May 2006.

[13] F. Pianese, J. Keller, and E. W. Biersack, "PULSE, a Flexible P2P Live Streaming System," in *Proc. 9th IEEE Global Internet Symposium 2006*, Bascelona, Spain, Apr. 2006.

[14] M. Castro, P. Druschel, A. M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: Highbandwidth Multicast in a Cooperative Environment," in *Proc. ACM Symposium on Operating Systems Principles (SOSP 03)*, The Sagamore, New York, USA, Oct. 2003.

[15] L. Massoulie and M. Vojnovic, "Coupon Replication Systems," in *Proc. ACM Sigmetrics 2005*, Banff, Alberta, Canada, June 2005.

[16] D. Stutzbach, D. Zappala, and R. Rejaie, "The Scalability of Swarming Peer-to-Peer Content Delivery," in *Proc. of Networking 2005*, Waterloo, Ontario, Canada, May 2005.

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

[18] E. W. Biersack, D. Carra, R. Lo Cigno, P. Rodriguez, and P. Felber, "Overlay Architectures for File Distribution: Fundamental Performance Analysis for Homogeneous and Heterogeneous Cases" in *Computer Networks Journal - Elsevier*, to appear.

[19] F. Baccelli, A. Chaintreau, Z. Liu, A. Riabov, S. Sahu "Scalability of Reliable Group Communication Using Overlays," in *Proc. IEEE INFOCOM 2004*, Hong Kong, Mar. 2004.

[20] S. Nikoletseas, J. Reif, P. Spirakis and M. Young, "Stochastic Graphs Have Short Memory: Fully Dynamic Connectivity in Poly-Log Expected Time," in *Proc. of the 22nd ICALP*, pp. 159-170, 1995

[21] P. Erdös and A. Renyi, "On random graphs," *Publ. Math.* 6:290–297, 1959.

[22] The Octave Web Page. Available: http://www.octave.org.

[23] Top applications (bytes) for subinterface: Sd-nap traffic, in CA/DA workload analysis of SD-NAP data. Available: http://www.caida.org/analysis/workload/byapplication/ sd-nap/index.xml, 2002.

[24] E. Adar, and B. A. Huberman, "Free Riding on Gnutella," in First Monday, Vol. 5, No. 10, October 2000.

[25] D. Carra, and R. Lo Cigno, "Stochastic Analysis of Chain Based File Distribution Architectures with Heterogeneous Peers," in *Proc. WCW 2005 (WCW 2005)*, Sophia Antipolis, Nice, France, Sept. 2005.

[26] D. Carra, R. Lo Cigno, and E. W. Biersack, "Content Delivery in Overlay Networks: a Stochastic Graph Processes Perspective," in *Proc. IEEE GLOBECOM 2006*, Nov. 27 – Dec. 1, San Francisco, CA, USA.

[27] D. Carra, R. Lo Cigno, and E. W. Biersack, "Fast Stochastic Exploration of P2P File Distribution Architectures," in *Proc. IEEE GLOBECOM 2006*, Nov. 27 – Dec. 1, San Francisco, CA, USA.

[28] D. Carra, R. Lo Cigno, and E. W. Biersack, "Stochastic Graph Processes for Performance Evaluation of Content Delivery Applications in Overlay Networks," submitted for publication. Available: `www.dit.unitn.it/locigno/preprints/CaLoBi_TPDS_V1.0.pdf`

[29] D. Carra, R. Lo Cigno, and E. W. Biersack, "On the Fundamental Properties of Mesh-Based Overlay Streaming Systems," Technical Report DIT-06-043, Univ. of Trento, July 2006. Available: `http://www.dit.unitn.it/locigno/preprints/DIT-06-043.pdf`

[30] H. P. Breuer and F. Petruccione "On the numerical integration of Burgers' equation by stochastic simulation methods," *Computer Physics Communications*, Vol. 77, Pages 207-218, 1993.

[31] J. Honerkamp, "Stochastic Dynamical Systems: Concepts, Numerical Methods, Data Analysis," 1994, VCH, New York.

[32] D.T. Gillespie, "Exact Stochastic Simulation of Coupled Chemical Reactions," *Journal of Physical Chemistry*, Vol. 63, Issue 25, Pages 2340-2361, 1977.

[33] S. M. Ross, "Introduction to Probability and Statistics for Engineers and Scientists," Academic Press, 8th edition, Dec. 2002.

[34] R. Gaeta, M. Gribaudo, D. Manini, and M. Sereno, "Analysis of Resource Transfer in Peer-to-Peer File Sharing Applications using Fluid

Models," *Performance Evaluation: an International Journal. Peer-to-Peer Computing Systems*, Vol. 63, Issue 3, Pages 147-264.

[35] C. Gkantsidis, and P. Rodriguez, "Network Coding for Large Scale Content Distribution," in *Proc. IEEE INFOCOM 2005*, Miami, Mar. 2005.

[36] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh," in *Proc. SOSP 2003*, Oct. 2003.

[37] S. N. Dorogovtsev, and J. F. F. Mendes, "Evolution of Networks: From Biological Nets to the Internet and WWW," Oxford University Press, Oxford, January 2003.

[38] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations," in *Proc. 11th ACM SIGKDD 2005*, Chicago, IL, USA, Aug. 2005.

[39] V. K. Goyal, "Multiple Description Coding: Compression Meets the Network," in *IEEE Signal Processing Magazine*, pp. 7493, Sept. 2001.

[40] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava, "PROMISE: peer-to-peer media streaming using Collect-Cast," in *Proc. ACM 2003*, Berkeley, CA, Aug. 2003.

[41] B. Cohen, "Incentives build robustness in BitTorrent," 2003.

[42] PeerSim: A Peer-to-Peer Simulator, `http://peersim.sourceforge.net/`