UNIVERSITÉ DE NEUCHÂTEL

Architectures for Peer-to-Peer Media Streaming in Large Scale Systems

par

Marc Schiely

Institut d'informatique

Thèse présentée le 8 Décembre 2009 à la Faculté des Sciences
pour l'obtention du grade de Docteur ès Sciences

Acceptée sur proposition du jury:

Prof. Pascal Felber, directeur de thèse
Université de Neuchâtel, Suisse
Prof. Peter Kropf, rapporteur
Université de Neuchâtel, Suisse
Prof. Laszlo Böszörmenyi, rapporteur
Université de Klagenfurt, Autriche
Prof. Benoît Garbinato, rapporteur
Université de Lausanne, Suisse

Université de Neuchâtel

FACULTE DES SCIENCES
Secrétariat-Décanat de la faculté
- Rue Emile-Argand 11
- CP 158
- CH-2009 Neuchâtel

IMPRIMATUR POUR LA THESE

Architectures for Peer-to-Peer Media Streaming in Large Scale Systems

Marc Schiely

UNIVERSITE DE NEUCHATEL

FACULTE DES SCIENCES

La Faculté des Sciences de l'Université de Neuchâtel,
sur le rapport des membres du jury

MM. P. Felber (directeur de thèse), P. Kropf,
B. Garbinato (Université de Lausanne),
et Laszlo Böszörmenyi (Université de Klagenfurt, A)

autorise l'impression de la présente thèse.

Neuchâtel, le 8 juin 2010

Le doyen :
F. Kessler

UNIVERSITE DE NEUCHATEL
FACULTE DES SCIENCES
Secrétariat - décanat de la faculté
Rue Emile-Argand 11 - CP 158
CH-2009 Neuchâtel
Felix Kessler

Acknowledgements

I rarely had the opportunity to thank persons who helped me to complete my thesis in the last few years. Many of my friends were always ready to discuss problems with me and tried to understand me.

First of all I want to thank Silvia who was sharing the office and a great time with many deadlines with me. Also all my other friends from the institute were giving me a lot of energy and joy, namely Sabina, Raphaël, Leo, Fred, Steve, Heiko, Claire, Christian, Olena and Walther.

Also many thanks to Pascal and Peter for their support all the years and the possibility to work with them and to participate in many winter schools and conferences.

Thanks a lot to Professor Böszörmenyi and Professor Garbinato for their valuable work.

My family was also motivating me throughout the years and helped me to finish my thesis. Many thanks to my parents Christa and Markus for the patience they had with me and the support they offered me. Also my sisters Andrea and Corinne and my brother Dominic were there when I needed them. Also my grandmother was a important person keeping asking me about my work and motivating me.

There were many other important friends present at my side which I do not explicitly list here. Thanks to all of them.

Abstract

Keywords: Peer-to-Peer, Media Streaming, Tree-based Architectures, Dynamic Adaptation Algorithms, Tit-for-Tat.
Mots-Clés: Pair-à-Pair, diffusion de flux, arbres de diffusion, algorithmes d'adaptation dynamique, Tit-for-Tat.

The distribution of substantial amounts of data to a large number of clients is a problem that is often tackled by peer-to-peer (P2P) architectures. Bottlenecks are alleviated by distributing the work of the server to all participating peers. Content is no longer passed directly from the server to all clients but only to a small subset of peers from where it is forwarded to a different subset of clients. These basic P2P ideas can also be applied to the distribution of live content, such as video streams. Additional timing constraints and bandwidth requirements of this application context lead to new challenges. Peer failures or late arriving packets directly influence the user perception, which is not the case in simple file distribution scenarios.

This thesis first analyzes some of the major problems faced by P2P live media streaming, and then presents a new architecture to address these challenges. Starting from a tree-based approach, the architecture is enhanced with adaptation algorithms to finally evolve in a mesh-based system. The in-depth analysis of tree-based architectures shows that it is important to adapt a node's position in the tree according to its bandwidth capacity. The same analysis is conducted for mesh-based architectures and it is shown that the position on the distribution path has a significant influence on performance. Another important problem concerns the fairness aspect in terms of collaborators and so-called "free-riders". A P2P system works best if all peers contribute with their resources. This can be ensured by tit-for-tat mechanisms where peers return as much as they get. In this thesis a new kind of tit-for-tat mechanism is developed to combine bandwidth contribution with robustness—the more bandwidth a peer provides the more robust its position on the path becomes.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Context

The typical usage of the Internet has evolved over the last ten years from simple message transfer and Web browsing to applications that involve much larger amount of data, such as video and music streaming. Additionally the number of Internet users has exponentially increased, which made the demand for bandwidth and server capacities explode [24]. Simply adding more servers to deal with this high demand is very expensive as the costs increase linearly with the number of users and additional hardware is needed to balance the load among the servers. Therefore new communication paradigms have been proposed to replace the short-comings of classical server/client architectures.

Instead of directly downloading content from servers, clients are requested to forward content as well and to cooperate such that resources from clients are also integrated in the distribution process. Initially, peer-to-peer (P2P) architectures were mainly used for distributing large files to a big number of users, but they have soon been integrated in other applications such as media streaming. Instead of only supporting pre-generated files, the P2P approach has been applied to live generated content, such as music or video streams that have more stringent requirements in terms of timing. The first live content streaming systems evolved in the late nineties and many have been proposed afterwards.

In this thesis we analyze the problem of P2P media streaming, and we propose solutions and a new architecture based on the results of our study. In the rest of this section we discuss the context of this thesis and summarize our contributions.

1.2 Challenges in P2P Media Streaming

While the use of P2P techniques for media streaming offers many benefits, it also comes with a number of challenges that have to be dealt with. In addition to the problems usually encountered in classical P2P systems, such as high churn rates, additional challenges are specific to media streaming. We take a closer look at these issues and discuss existing solutions.

1.2.1 Scalability

In all live streaming systems a single streaming source exists which produces streaming content. We assume that this central unique instance never fails as the whole streaming system would fail otherwise. We usually distinguish between three types of P2P architectures: (1) systems that rely on a single central instance (different to the source) of a specific component (e.g, the Akamai CDN [53]), (2) fully decentralized architectures where each participating peer has the same role (except the source), and (3) hybrid or hierarchical models with some peers (usually called "superpeers") having a specific role. Obviously, the scalability of the first approach is limited by the capacity of the central instance, which also represents a single point of failure. The hybrid approach mitigates this problem by essentially replicating the service provided by the superpeers.

Fully decentralized architectures have the greatest potential for scalability and reliability, but they have to face additional complexity in their topology management protocols. As there is no global knowledge about the network, all operations have to rely on local partial knowledge. This may lead to suboptimal performance (when compared to centralized, omniscient algorithms) but this is usually a small price for having a scalable system. Further, as each peer has the same role, failures are not as critical as the failure of a

centralized component or a super peer in a hybrid model. In environments with high churn, where many nodes join and leave the system, this property is essential.

1.2.2 Robustness

One of the major advantages of the P2P paradigm is also one of its biggest problem: the peers are acting in an unpredictable manner and independently of each other. The rate at which peers join and leave the system can be very high. Worse, departures may be ungraceful in the sense that peers fail or leave without prior notification. As a consequence, P2P systems must incorporate some form of self-healing mechanisms and construct robust topologies (e.g., using redundant paths).

In general, the robustness of P2P networks can be increased by using either data redundancy or path redundancy. The most widely used techniques for data redundancy are *forward error correction* (FEC) [45], *layered coding* and *multiple description coding* (MDC) [10]. FEC uses encoding techniques, such as Reed-Solomon codes, to encode a number of packets n into m packets where $m > n$. Any subset k ($k \geq n$) of these m packets is enough to reconstruct the n original packets.

In layered coding, the original media stream is split into different layers of quality. The base layer is the most important and must be received in any case. Each additional layer improves the quality of the stream.

Finally, MDC divides a stream into different descriptions (substreams) where each description can be distributed on a different network path for avoiding network failures or congestion. Any subset of descriptions can be used to decode the original stream. The more descriptions are available the higher the streaming quality is. The error resilience is therefore higher than it is in layered coding as the loss of a description does not lead to a streaming interruption but only to a temporarily decreased streaming quality.

Redundancy in data alone does not help if a peer has only one other peer serving the data. If this single data source fails then the receiving peer does not get any data. A better strategy is to have multiple sources that serve different parts of the stream such that, if a subset of the neighbors fail,

the remaining peers can still serve the data needed to play back the stream. Most existing P2P media streaming systems provide such support for path diversity using redundant distribution trees or mesh-based topologies [34].

1.2.3 Latency

P2P media streaming architectures are based on application-level overlay networks. This means that messages from the source to a given peer typically follow a longer route than IP's shortest path. In order to minimize the network's stretch and the end-to-end latency, peers that are physically close should be neighbors in the logical overlay. Thus, the construction of the distribution architecture does not only need to take into account robustness, but also performance metrics [42]. The neighbors of a node have to be selected in an intelligent way to optimize the chosen metrics.

1.2.4 Throughput Optimization

In traditional P2P file sharing systems, each peer tries to download content as fast as possible, i.e., maximize its effective bandwidth. In contrast, media streaming architectures must provide a timely constrained download rate for a smooth playback of the stream. Multiple cooperating peers are needed to balance out bandwidth fluctuations, so that the loss or degradation of service from one peer can be compensated by other peers.

The service capacity of a P2P system consists of the aggregate upload bandwidth of all participating nodes. As this bandwidth is a scarce resource its usage must be optimized. In the optimal case, each peer can obtain a peak service capacity equal to the aggregate bandwidth divided by the number of nodes.

1.2.5 Timing Constraints

For distributing a large file to a high number of clients it is typically split into a number of equal-sized chunks. Each chunk can be distributed independently and the file can be reassembled at the peers. A participating node requests missing chunks from neighbors based on a chunk-selection strategy

(e.g., rarest-chunk-first strategy) until it has the complete file. The order has no importance as the file is only used when its download is completed.

Live streaming systems are very different in this respect: chunks are only useful for a peer if it arrives before its scheduled playback time. Chunks far in the future cannot be requested as they often do not exist yet. These timing constraints make the trading window of a peer very narrow and impose additional mechanisms to handle bandwidth degradations and peer failures.

As we cannot rely on guarantees from the transport layer that a packet arrives on time, we need some over-provisioning of bandwidth to deal with small delays and limit the risk of missing some deadlines. This requirement is tightly coupled with the throughput optimization problem. The more download bandwidth each peer gets, the higher the probability that all blocks arrive on time.

1.2.6 Fairness

An important observation made on current peer-to-peer file distribution systems is the existence of selfish peers, so-called freeriders [1]. These peers try to download from the system without serving other nodes. As peer-to-peer systems live from cooperation, this is an important problem that needs to be addressed.

Without any central instance that controls data flows in the system, the peers must have the ability to penalize peers that are unfair. Ideally, each peer should only get as much as it contributes, except the initialization phase where a peer is not able to provide data, but this objective is not compatible with the uniform bandwidth requirement of media streaming. Yet, a peer that contributes shall get more advantages than a node that does not contribute at all, for instance lower delays or higher reliability.

1.3 Research Goal

The goal of this thesis is to study cooperative distribution of streaming media and large files from a networking perspective. The problem should be addressed by analysis, modeling, prototype implementation and experimen-

tal validation. The algorithms presented should be practical enough to be implemented.

The solutions developed in this thesis should not depend on special peers with dedicated roles (e.g., centralized nodes, super-peers). Instead all peers should be assumed to be equal in terms of their role in the network but heterogeneous in terms of bandwidth. The presented architectures should further be resilient to failures and provide simple incentives for peers to contribute to stream distribution.

1.4 Contributions

We briefly summarize below how the different problems identified are solved in this thesis.

1.4.1 Throughput Optimization

In todays Internet users are typically connected by asymmetric links, such as ADSL where upload bandwidth is smaller than the download bandwidth. Therefore in this work the assumption is made that the upload bandwidth is the limiting resource (usually bandwidth is also more constrained compared to CPU and disk space). The peers must obviously have a download capacity that is at least as high as the maximum streaming rate to be able to receive the stream.

Under the assumption that all peers want to consume a stream with a given rate, one must develop architectures able to guarantee the same minimum download rate across the whole network. In an environment where peers have different upload capacities, the position of each node has a high impact on overall download performance, e.g., if a peer with high-bandwidth capacity is placed at the end of a distribution chain then its upload bandwidth is wasted. In Chapter 3 we present our different approaches for architectures providing uniform download rates and their analysis.

1.4.2 Adaptation Algorithms

The architectures developed in Chapter 3 can only be constructed with global knowledge of all peers. Maintaining such information would necessitate non-scalable solutions, e.g., a central tracker node. Therefore, algorithms to dynamically approximate optimal distribution architecture have been developed in Chapter 4. Peers initially join the system at any position where they are accepted (each node has a maximal number of neighbors). Starting from this configuration, nodes dynamically adapt their position according to their upload bandwidth. The higher its upload capacity the nearer to the source a peer should be. It is analytically shown that this adaptation leads to a significantly higher throughput.

1.4.3 Fairness and Robustness

The position of the peers in distribution trees does not only affect overall performance, but it also lets peers at leafs operate as "free-riders", i.e., consume without contributing. To prevent peers from being selfish, additional fairness mechanisms have to be implemented. Systems like BitTorrent [17] use a tit-for-tat mechanism where peers only return as many chunks as they receive (except for the bootstrap phase). This approach can slow down the system as peers are waiting for data before they send chunks. In a live streaming system this delay can have a strong impact on the stream quality and may break down the system. Therefore a new fairness approach based on robustness has been developed and is presented in Chapter 5.

The more children a peer p serves, the higher the number of backup links it gets. In the case of a failure of one of its parents, the peer can choose a new source among all its backups. A peer that is not contributing (has no children) will strongly be affected by a failure and has to rejoin the system—an operation that is time-consuming and produces gaps in the stream. On the other hand, a cooperating peer has a high chance to find a stable backup peer with sufficient upload capacity.

1.4.4 Mesh-based Architectures

As opposed to tree-based systems (studied in the first chapters of the thesis), in mesh-based systems peers have more flexibility in choosing neighbors. Instead of having a fixed position and forwarding chunks from its parent to its children, a peer p trades chunks with a much larger set of neighbors whose composition evolves dynamically. If one of p's active neighbors fails then p simply discards it from its neighbor set and chooses another neighbor to trade chunks with. This mechanism makes mesh-based systems much more robust than tree-based architectures where a parent failure affects the child and its whole sub-tree. Mesh-based architectures are analyzed in Chapter 6 by considering them as a collection of distribution trees.

1.5 Structure

The thesis is divided into 7 chapters where Chapter 2 discusses related work, Chapters 3 to Chapter 5 form the main technical part and Chapter 7 concludes the thesis and outlines future work.

Outline. Chapter 3 analyzes different architectures which have all the same property of providing a uniform download rate to all participating peers. Parts of the chapter have been published in [48]. Chapter 4 presents an adaptation algorithm which can further enhance the throughput of distribution architectures. The content has been partially published in [51].

In Chapter 5 a new P2P media streaming architecture, integrating the results from the previous chapters, is presented. This chapter has partially been published in [49] and in [50]. Chapter 6 analyzes the structure of mesh-based systems and makes the link from the tree-based CrossFlux design to mesh-based architectures. Parts of this chapter have been published in [6].

Chapter 2

Related Work

2.1 Overview of Existing Systems

Although many proposals for P2P media streaming architectures exist, only few were implemented and deployed. One very popular system which is widely used in Europe is Zattoo [60]. It became available in 2006 and quickly attracted a large user base (more than 20% of all Swiss Internet users). Zattoo offers dozens of TV channels and operates in several countries. Unfortunately no technical details are available. The same holds for PPLive [43], which has only been studied by performance measurements [23].

We can identify the following reasons for the slow growth of P2P media streaming systems:

1. A critical mass of users is needed for cooperation to be effective. If there are only few participants, then the media server can use traditional multicast communication.

2. Most existing systems fail short of providing the properties users expect from media streaming architectures: (1) no interruptions and no jitter; (2) fast startup of the stream; and (3) quick recovery of failures such that the stream is played back continuously also under high churn.

3. P2P systems have to deal with legal and political issues. The owners of a stream lose the control of how the stream is being distributed. Fur-

ther, any user with video recording equipment is able to serve streams, which opens the door to copyright infringement.

We describe below some of the most well known streaming systems and highlight the differences with our CrossFlux architecture described in Chapter 5. Given the large number of proposals found in the literature, this list is not exhaustive.

CoolStreaming. One of the most widely used systems is CoolStreaming [61], which has been deployed with up to 30,000 distinct users in 2004 but has been stopped due to copyright issues in 2005. Unlike many other systems, CoolStreaming is data-driven, considers bandwidth heterogeneity, and tries to reduce latency between pairs of peers. A set of backup nodes is maintained to deal with failures and adapt to changing network properties. Backup nodes are periodically contacted to see if they provide higher performance than certain nodes currently serving the stream; in that case, nodes may be exchanged. A limitation of this optimization strategy is that it is restricted to the nodes of the backup set. In contrast, the algorithms used in CrossFlux allow to perform "transitive" optimizations, i.e., not limited to the exchanges with direct neighbors.

Chunkyspread. Chunkyspread [56] is an unstructured approach to media streaming. It uses a multi-tree (multi-description) based structure. The structure is very dynamic as each peer periodically looks for new partners in its local environment. It exchanges information (load, latency, creation of loops) with each neighbor to search for the best parent-child pairs for each tree. The constraints on these relationships are (1) avoid loops, (2) satisfy any tit-for-tat constraints, (3) adapt load (shall be in a per peer defined range) and (4) reduce latency. In contrast to Chunkyspread, CrossFlux combines fairness with robustness. Trees are built in a more structured way to include backup links.

End System Multicast. End System Multicast (ESM) [14] is a P2P media streaming solution that provides several desirable properties. An overlay

mesh is initially constructed and multiple spanning trees, rooted at each possible source, are constructed on top of it. The trees are then incrementally enhanced by adding or dropping additional links depending on a utility function. In CrossFlux, we try to construct a good mesh from the beginning and incorporate performance metrics during the joining process of new nodes. In addition, CrossFlux introduces a notion of fairness by using links between nodes in one direction to serve streaming data and in the other direction as backup link.

PeerStreaming. PeerStreaming [30] differs from other systems in that it adapts the streaming bit-rate dynamically to the available bandwidth, which directly depends on the number of serving peers. The clients reading the stream receive different parts from multiple altruistic serving peers. A new node joins the system by asking a list of serving peers and connects to a number of them. The main drawback is that, unlike CrossFlux, there is no incentive for the serving peers to participate in the system and to help distribute the stream.

GnuStream. GnuStream [27] is built on top of the Gnutella P2P substrate [47, 20]. A peer in GnuStream queries the Gnutella network to locate multiple parents that have part of the stream. Parts of the stream are then requested from these parents and aggregated in the peer for playback. As GnuStream relies upon Gnutella, its implementation is very simple: joins and searches are mapped to the underlying protocols, while failure recovery is achieved by simply exchanging a failed source with another one. This simplicity comes at the price of some performance loss. Gnutella is not optimized for live media streaming and, therefore, may not perform as good as a system that has been designed specifically for that purpose, as CrossFlux is.

SplitStream. SplitStream [8] is a P2P media streaming architecture that focuses on robustness. As in our model, the stream is split into multiple stripes that can be distributed independently. A distinct tree is constructed for each of these stripes spanning over all participating peers. The robustness

in SplitStream comes from the fact that each node is inner node in at most one tree and leaf in all the other trees. Thus, if a peer fails, only one distribution tree is affected and has to be rebuilt. In CrossFlux, peers may be placed as interior nodes in more than one tree but quick recovery from a peer failure p is achieved by using backup paths which do not inlcude p. Additionally, fairness is introduced in the tree architecture in rewarding forwarding peers with higher robustness.

CollectCast. The CollectCast [22] architecture is built on top of a P2P DHT substrate, such as Chord, CAN, or Pastry. Failures or stream degradations are handled by exchanging active senders. Further, CollectCast tries to optimize the download rate at each peer by selecting the best performing peers out of a candidate set. In contrast, CrossFlux does not rely on fixed candidate sets but performs a more global optimization by moving peers across the trees.

CoopNet. CoopNet [36] combines a classical client-server model with a P2P architecture. The server is responsible for directing joining nodes to potential parents and for reconnecting peers upon failure of their parents. The central instance obviously limits scalability and represents a single point of failure whereas in CrossFlux there is no central component.

NICE. NICE [2] uses a hybrid architecture in which peers are clustered in a hierarchical layer structure. Each cluster has a leader, which also belongs to the next layer above. Latency can be optimized by selecting as leader a peer that is close to the center of the cluster. The system focuses on low-bandwidth streams distributed to a large receiver set. Thus, optimization of the available bandwidth is not a major objective of NICE and has not been explicitly addressed.

ZIGZAG. ZIGZAG [54] is another layer-based architecture. Like NICE [2], it constructs clusters that are grouped in a hierarchical structure. Unlike NICE, ZIGZAG dynamically adapts to the load of the cluster heads: if a node has too many children or no sufficient bandwidth capacity, it can dis-

tribute the load by reconfiguring the cluster. ZIGZAG does not use path redundancy and it is not clear how well it scales when distributing high-bandwidth streams.

2.2 Analytical Models for Download Rate

Two main approaches exist for dealing with differences in uplink bandwidth in overlay multicast systems. Narada [12], CollectCast [22] and GnuStream [27] use bandwidth measurements to improve the overlay structure by dynamically replacing links. In contrast Scattercast [11], SplitStream [8], Overcast [26] and ALMI [38] use degree-constrained structures to deal with heterogeneity. If a peer's degree is saturated when a new peer wants to connect, then some reorganization needs to take place. CoopNet [36] uses both of these techniques. It deploys multiple parallel trees and reorganizes them based on performance feedbacks.

All of these systems do not try to uniformly distribute the download rate to all peers. Instead, they send distinct streams at different rates, or they consider bounded streams and use buffers to deal with timing problems. Our goal is to minimize the buffer requirements by evening out the download rate at all peers.

In [44], the authors investigate the impact of heterogeneous uplink bandwidth capacities on Scribe [9]. Their experiments show that heterogeneity may create distribution trees with high depths, which is not desirable. After proposing several ways to address the problem they conclude that heterogeneity in DHT-based multicast protocols remains a challenging open problem.

Analytical models have been proposed for peers with homogeneous bandwidth capacities [3, 59], as well as for heterogeneous peers but for non-uniform download rates [7]. Different architectures for homogeneous and heterogeneous bandwidth constraints are analyzed. In contrast to this work, the authors make the assumption that the downlink and uplink capacities are symmetric and do not consider uniform download rates.

2.3 Adaptation Algorithms

Many architectures for content distribution have been proposed. Most of these systems build an overlay network that is kept throughout the distribution process. Links are only changed if either a neighbor fails or the performance heavily degrades. Affected nodes then simply rejoin the tree starting at the root. Most architectures do not actively reconfigure links before a degradation occurs.

CollectCast [22] is an example of such a passive system. The authors propose an architecture that works on two different sets of nodes for media streaming. From a set of potential senders the best ones are taken and form the active set. The other potential senders are kept in a standby set. During the streaming process peers do passively measure bandwidth and latency. If the quality of the media streaming falls below a threshold, a peer from the active set is exchanged with one from the standby set. A similar exchange technique has been proposed in GnuStream [27] for use with the Gnutella system.

Other systems like Scattercast [11] try to construct near-optimal distribution trees in advance. A set of agents is deployed across the network. The agents together provide a multicast service. The number of clients that join an agent is limited by its bandwidth capacity. The goal of Scattercast is to construct a degree-constrained spanning tree across all agents and keeping the average delay between the source and all destinations at a minimum. This problem is known to be NP-hard.

One system which dynamically adapts to the network conditions was presented with TMesh [58]. The architecture aims at reducing latencies between nodes in a multicast group. Based on a set of heuristics, new links are added to the existing tree or mesh. If the new link reduces the overall latency then it is kept; otherwise, it is dropped.

2.4 Fairness and Robustness

The impact of fairness on download performance has been studied in [15]. A framework is presented to evaluate maximum achievable download rate

of receivers as a function of altruism. The results show that fairness has a high impact on the performance of receivers and that a small degree of altruism brings significant benefit. In [13] a taxation model is presented in which peers with higher upload capacity help compensating bandwidth of peers with lower bandwidth.

A framework based on game theory is presented in [31]. In this paper incentive-based strategies to enforce peer cooperation are evaluated and compared.

In [57] it is shown that not only a high number of users is necessary to build a robust system, but the main contribution to the system is provided by some stable peers with high upload capacity. This confirms our approach of propagating well-performing peers toward the root and enhancing their robustness by additional backup links.

2.5 Mesh-based Approaches

Many mesh-based P2P streaming systems have been proposed in the last few years [37, 33, 40], but none of them has been formally analyzed due to their complexity. Mainly these architectures have been studied by means of simulations [18, 32] or experimental evaluation [41].

A comparative study of tree- and mesh-based approaches for media streaming is presented in [34]. The authors first propose an organized view of data delivery in mesh overlays, which consists of data diffusion and swarming phases, and later introduce delivery trees, which they discover in mesh overlays in a similar fashion to diffusion trees described in our thesis. This work is different in that it focuses on formally analyzing properties of diffusion trees rather than evaluating them by simulation. Further an overlay adaptation algorithm that improves properties of these trees is proposed.

A different approach to analyzing P2P media streaming systems are fluid models. In [29] the authors present a stochastic fluid model that takes into account peer churn, heterogeneous peer upload capacities, peer buffering and delays. In this thesis the distribution trees created in a mesh are analyzed such that known adaptations for tree-based approaches can be applied to meshes.

2.6 Summary

The systems presented in Section 2.1 study different important aspects of media streaming in P2P systems. They focus mostly on isolated problems and their solutions. The architectures proposed in this thesis are not only adapted during data dissemination but also are designed from the beginning to meet the challenges of P2P media streaming.

Several analytical models developed for P2P media streaming are discussed in Section 2.2. Many of them focus on systems composed of peers with homogeneous upload bandwidths, where only few consider heterogenous upload bandwidths. The present thesis studies a model that assumes peers with heterogeneous up- and download bandwidths, with the objective of providing all peers throughout the system with sufficient download bandwidth for receiving the media stream with the same streaming rate.

Section 2.3 presents systems based on adaptation algorithms. In many of them, peers use a backup set for replacing underperforming peers from the current neighbor set by more powerful ones. Such an approach could also be used in our architectures, leading to a more robust system. In contrast, the adaptation algorithms presented in this work are designed to optimize bandwidth usage instead of reducing latency. They are apt to both tree- and mesh-based architectures.

Section 2.4 discusses the problem of fairness in P2P media streaming. Literature studies this topic in detail, mainly through experimental approaches. All those studies conclude that the problem of free-riders is an important issue in P2P systems. The architectures presented in this thesis introduces a novel approach to fairness, by trading bandwidth contribution against increased robustness.

Finally, mesh-based models are discussed in Section 2.5. This thesis presents a novel analytical approach to study tree-based diffusion patterns in mesh-based architectures. With this approach, existing algorithms and models for tree-based architectures can be applied to mesh-based systems.

Chapter 3

Peer-to-peer Distribution Architectures providing Uniform Download Rates

3.1 Introduction

Early studies of content distribution architectures have primarily focused on homogeneous systems where the bandwidth capacities of all peers are similar, or simple heterogeneous scenarios where different classes of peers with symmetric bandwidth try to minimize the average download duration. Such settings are not representative for real-world streaming networks.

In this chapter, we study the problem of content distribution under the assumption that peers have heterogeneous and asymmetric bandwidth (typical for ADSL connections), with the objective to provide uniform download rates to all peers—a desirable property for distributing streaming content. Our goal is to propose and analyze different architectures for peer-to-peer networks that are able to sustain large populations of clients while delivering

Parts of this chapter have been published in: M. Schiely and P. Felber. Peer-to-peer Distribution Architectures providing Uniform Download Rates. *Proceedings of the International Symposium on Distributed Objects and Applications (DOA'05)*, October 2005.

a given stream to all of them, under the simplifying assumption that peers are fair and no failures occur.

Unlike previous studies, we assume that the peers have heterogeneous and asymmetric bandwidth (typical for ADSL connections) and we aim at providing a uniform download rate to each of them. This property is crucial for applications like media streaming, for which users expect an uninterrupted stream of data.

We consider simple models with two classes of peers that differ in their uplink capacities. We study several architectures that achieve optimal utilization of the aggregate uplink capacity of the system and share it equally between all the peers. It obviously follows that fast peers must share more bandwidth than they receive, but this unfairness can be balanced by placing them nearer to the source for increased reliability and shorter latency.

The analytical models developed in this chapter provide interesting insights on the performance of content distribution architectures with uniform download rates in various configurations. By comparing them with other architectures providing non-uniform rates, we conclude that uniformity can be achieved with little additional complexity and no performance penalty.

The rest of the chapter is organized as follows: we first present the system model in Section 3.2. Then, we analyze three different architectures providing uniform download rates in Section 3.3 and compare them in Section 3.4. Finally, Section 3.5 summarizes the chapter.

3.2 System Model and Definitions

For the rest of this chapter we use the following model. We assume that nodes in the network have different upload capacities. We analyze content distribution architectures with two classes of nodes, referred to as *fast* and *slow* peers according to their upload bandwidth. All nodes in a class have the same bandwidth. The data stream is sent by a single source which has the same bandwidth as fast nodes. To simplify the analysis, we assume that the source receives the data at the same uniform rate as the other peers before distributing it within the content distribution network. We shall ignore latency in our model.

As is the case for typical ADSL connections, we assume that the slow peers are essentially limited by their uplink capacity and have sufficient download bandwidth to receive the data at the same uniform rate as the other peers.[1] We consider N_f fast peers in class F with upload bandwidth B_f and N_s slow peers in class S with upload bandwidth B_s ($B_f > B_s$). For the sake of simplicity, we assume in our analysis that $B_s = \frac{B_f}{k}$ with k being an integer value. The total number of peers is $N = N_f + N_s$.

We analyze the behavior of different architectures when transmitting a large content. We assume that the file being transmitted is split into C chunks that can be sent independently: as soon as a peer has received a chunk, it can start sending it to another peer. We consider one unit of time to be the time necessary for transmitting the whole content at the uniform rate r that is provided to all peers. Each chunk is thus received in $\frac{1}{C}$ unit of time. For clarity, we shall describe the different architectures with the assumption that we transmit the whole file at once and we shall introduce chunks later in the analysis. As total download time is a function of the number of chunks, our main objective of supporting streaming data corresponds to situations where $C \to \infty$.

A peer may receive chunks from the source via different paths. For instance, in the case of SplitStream [8], the source splits the content into several layers and sends each of them along distinct trees spanning all the nodes. Two chunks sent *at the same time* by the source may thus traverse a different number of peers and be received at different times. This implies that each peer may have to buffer some chunks until all of those sent at the same time have been received. We compute δ_T as the maximal difference in distance between a peer and the closest common node along the paths to the source via distinct incoming links. This value indicates the buffer space needed at the peer. For instance, in Figure 3.1, the first node of the right chain receives chunks from the source in 1 (directly), 2 (via one peer), and 3 (via two peers) units of time and we have $\delta_T = 3 - 1 = 2$. Clearly, small values of δ_T are desirable and we shall also compare the different architectures with respect to this property.

[1] As we shall see, this rate is not higher than the uplink capacity of the fast peers.

3.2.1 Uniform Rate

As previously mentioned, our goal is to provide the same download rate to all peers in the network. Obviously, the maximal rate r that can be achieved corresponds to the aggregate upload bandwidth of all nodes divided by the number of peers ($B_s < r < B_f$). It is easy to see that a tree cannot be used to fulfill this goal because a slow node does not have enough upload bandwidth to serve even a single other peer at rate $r > B_s$.

A trivial approach is to form chains of peers, in which a combination of slow and fast peers team up and share their bandwidths at each level of the chain. Figure 3.1 shows such an architecture with 50% fast nodes and 50% slow nodes ($N_f = N_s = \frac{N}{2}$), and slow nodes having half of the upload bandwidth of fast nodes ($B_s = \frac{B_f}{2}$). The source is the topmost node and the numbers show the transmission rate on the corresponding link, as a fraction of B_f. Fast nodes are displayed in gray. The time units indicated in the figure do not explicitly take chunks into account: at $t = 1$, the second peer in the left chain has received the content at rate $\frac{3}{4}$; at $t = 3$, the first peer in the right chain has received the content via three links, each at rate $\frac{1}{4}$; etc. All time units should be divided by C when considering chunks. The download rate r is calculated as:

$$r = \frac{1}{N}(\frac{N}{2}B_f + \frac{N}{2}\frac{B_f}{2}) = \frac{3}{4}B_f$$

We can observe that an unused upload bandwidth of $\frac{3}{4}B_f$ remains because the source does not download any content. We shall ignore this in the rest of the chapter.

If we generalize the upload bandwidth of the slow peers to a fraction of the upload bandwidth of the fast peers $B_s = \frac{B_f}{k}$ and compute the download rate r for a scenario where $N_f = N_s = \frac{N}{2}$, we obtain:

$$r = \frac{1}{N}(\frac{N}{2}B_f + \frac{N}{2}\frac{B_f}{k}) = \frac{k+1}{2k}B_f$$

We now relax the assumptions on the distribution of fast and slow nodes. If the number of fast peers is N_f, then the number of slow peers is $N_s = N - N_f$. Again the upload bandwidth of the slow peers is a fraction k of the

upload bandwidth of the fast peers. The optimal download rate is then:

$$\begin{aligned} r &= \frac{1}{N}(N_f B_f + (N - N_f)\frac{B_f}{k}) \\ &= (\frac{N_f}{N} + \frac{1}{k}(1 - \frac{N_f}{N}))B_f \end{aligned} \tag{3.1}$$

If slow peers do not serve any content, i.e., $k \to \infty$, then Equation (3.1) becomes:

$$\lim_{k\to\infty}(\frac{N_f}{N} + \frac{1}{k}(1 - \frac{N_f}{N}))B_f = B_f\frac{N_f}{N}$$

In a scenario where $N_f = N_s$ this leads to a binary tree where the slow nodes are the leaves and the fast nodes are the inner nodes, each serving two other nodes at rate $r = \frac{B_f}{2}$ (as studied in [3]).

In the rest of the chapter, unless explicitly mentioned, we consider equal populations of slow and fast peers ($N_s = N_f$).

3.3 A Study of Three Architectures

We now study and compare three different architectures that provide a uniform download rate to all peers.

3.3.1 Linear Chain Architecture

The first architecture considered in this chapter consists of multiple linear chains of peers. Several independent linear chains of peers are forked and used to distribute content in parallel. In a chain each peer has exactly one parent to receive data from and one child to send data to. The chains are constructed in three phases.

Phase 1 - Growing phase. The objective of the growing phase is to serve several peers (say m, $m > 1$) in parallel starting from a single source. Obviously, an expansion (i.e., forking of chains) can only be achieved by fast peers, as they have more upload capacity than the target download rate

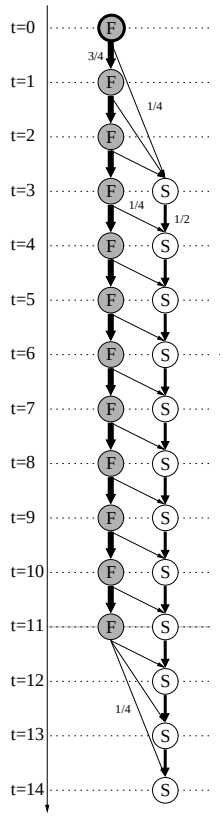


Figure 3.1: Linear chains with one expansion step and $k = 2$ ($B_s = \frac{B_f}{2}$).

r (where r is the aggregate upload bandwidth of all peers divided by the number of peers). Using this free capacity allows us to build the service capacity mr necessary to serve m peers in parallel.

Informally, the growing phase proceeds as follows. The first fast node (the source) starts a chain by serving one other fast peer with rate r. The remaining bandwidth $B_f - r$ will be used in another chain. The second fast peer again serves another fast peer with rate r, which also leaves it with $B_f - r$ remaining bandwidth. This process continues until the sum of the remaining bandwidths of the first p fast nodes is sufficient to serve another peer, i.e., $p(B_f - r) \geq r$. Given that $B_s = \frac{1}{k}B_f$, p can be computed as:

$$p = \left\lceil \frac{k+1}{k-1} \right\rceil$$

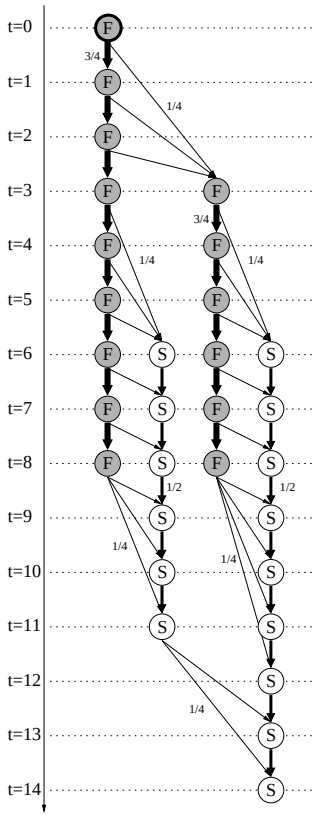


Figure 3.2: Linear chains with two expansion steps and $k = 2$ ($B_s = \frac{B_f}{2}$).

In the formula above, depending on the value of k, some bandwidth may be lost in the integer conversion. This can be avoided by expanding to k nodes at once. The number of peers p_k necessary for this expansion can be computed by solving $p_k(B_f - r) = r(k-1)$, which gives:

$$p_k = k + 1$$

In the rest of the chapter, we shall assume expansions to k chains using p_k peers (instead of 2 chains using p peers). Each fast peer can in turn fork another k chains with the help of $p_k - 1$ other fast peers. By repeating this process, the number of chains can be multiplied by k every iteration. Each expansion obviously requires p_k units of time. Examples with $k = 2$ ($r = \frac{3}{4}B_f$) and $k = 4$ ($r = \frac{5}{8}B_f$) are shown in Figures 3.1, 3.2, and 3.3. It

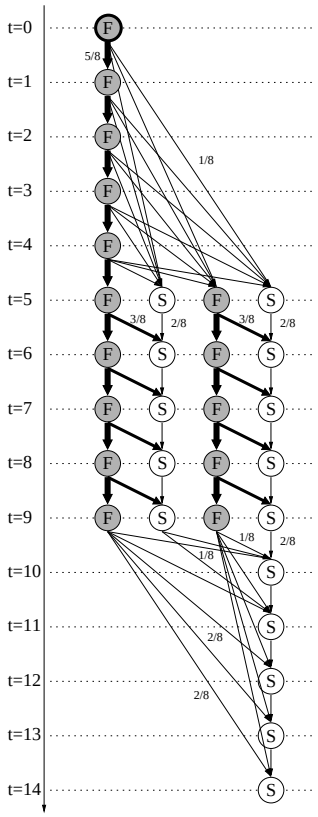


Figure 3.3: Linear chains with one expansion step and $k = 4$ ($B_s = \frac{B_f}{4}$).

is important to note that the peers are organized as a directed acyclic graph (DAG).

Phase 2 - Parallel phase. The parallel phase starts when the growing phase has finished its expansion to m peers. It constructs two sets of $\frac{m}{2}$ linear chains, composed respectively of fast and slow peers. Each chain of slow peers is combined with a chain of fast peers. A slow peer serves its successor at rate B_f/k. A fast peer serves its successor at rate r and the next slow peer in the companion chain at rate $B_f - r$. Thus, each peer is served at rate r. Phase 2 proceeds until all fast peers are being served (see Figures 3.1, 3.2, and 3.3).

Phase 3 - Shrinking phase. In the last phase, we are left with a set of slow peers to serve at rate r. As a slow peer cannot serve another peer by itself, the bandwidth of several peers must be combined, which leads to shrinking down the number of parallel chains. This phase is almost symmetrical to the growing phase, in that we can serve p_k slow peers from each set of k chains. We repeat this process until all slow peers have been served (see Figures 3.1, 3.2, and 3.3).

3.3.2 Analysis

We can easily notice that delays of $\delta_T = k$ are encountered during the growing phase. The case of the shrinking phase is more subtle, as δ_T grows larger if we keep it perfectly symmetric to the growing phase. By allowing some asymmetry, we can both bound the delays by the same value $\delta_T = k$ and reduce the total length of the shrinking phase.

We now compute the number of peers that can be served within a given time interval. After p_k steps, k peers can start again another chain. If we define s as the number of expansion steps, we can calculate the number of peers in the first phase N_1 to be:

$$N_1 = \sum_{i=0}^{s-1} k^i p_k = p_k \frac{k^s - 1}{k - 1}$$

The shrinking phase is built in a symmetric manner. Therefore the number of nodes N_3 in the third phase is the same as in the growing phase: $N_3 = N_1$. Given the constraint that $N_1 + N_3 \leq N$, the maximal value of s is:

$$s_{max} = log_k \left(N \frac{k-1}{2p_k} + 1 \right)$$

The number of nodes N_2 that can be served in phase 2 in a given time interval T is:

$$N_2 = k^s (T - 2sp_k + 1)$$

Indeed, there are k^s parallel nodes and phase 2 lasts for the given time interval minus the duration of the growing and shrinking phases. The number of peers served in a time interval T with s growing steps ($1 \leq s \leq \lfloor s_{max} \rfloor$) is then:

$$N(T, s, k) = 2p_k \frac{k^s - 1}{k - 1} + k^s(T - 2sp_k + 1)$$

We observe that the number of peers served in a given time interval grows with s, producing thus more efficient content distribution architectures (compare $N(14, 1, 2) = 24$ in Figure 3.1 and $N(14, 2, 2) = 30$ in Figure 3.3).

Solving the equation for T gives the number of units of time necessary to serve N peers:

$$T(N, s, k) = \frac{N(k - 1) - 2p_k(k^s - 1)}{k^s(k - 1)} + 2sp_k - 1 \tag{3.2}$$

Assuming that the content is split into chunks, the total download time for the complete file is then $1 + \frac{1}{C}T(N, s, k)$, i.e., the time necessary to transmit the whole file at rate r plus the propagation time of the chunks through the content distribution network. Using Equation (3.2) leads to:

$$T(N, s, k, C) = 1 + \frac{1}{C}\left(\frac{N(k - 1) - 2p_k(k^s - 1)}{k^s(k - 1)} + 2sp_k - 1\right) \tag{3.3}$$

Figure 3.4 shows the time necessary to complete the download with the linear chain architecture for different values of k and C. We observe that performance improves with larger numbers of chunks, because all peers can be active most of the time. In contrast, with few chunks only a fraction of the peers will be uploading at any point in time, while the others have either already forwarded the entire file or not yet received a single chunk. Therefore, the value of k, which influences the depth of the content distribution architecture, has more impact on performance when the number of chunks is small. In Figure 3.4 one can notice indeed that the reduction of download times starts earlier with small values of k because they yield deeper architectures.

Figure 3.5 compares the download times for different values of s (the

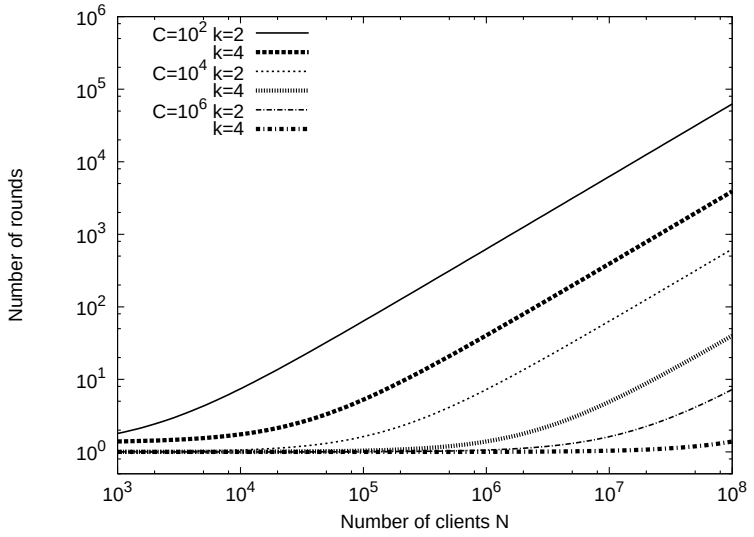


Figure 3.4: Download time (in rounds) of the linear chain architecture for different values of k and C ($s = 4$).

value s_{max} corresponds to the maximal number of expansion possible with the given peer population). As expected, performance improves with higher values of s because they produce architectures which have shorter paths from the source to all other peers. The optimal value s_{max} exhibits extremely good scalability.

3.3.3 Mesh Architecture

The linear chains architecture can be improved in several ways if we allow peers to be organized as a directed graph with cycles. We can reduce the duration of the growing phase and thus the length of the paths (and consequently the latency); we can simplify network management by only using connections with identical bandwidth capacities; and we can limit the size of buffers at each peer to a constant value.

The resulting mesh architecture is shown in Figure 3.6 (for $k = 2$ and one expansion step) and Figure 3.7 (for a general value of k and two expansion steps). The download bandwidth for each peer is the aggregate upload bandwidth divided by the number of peers ($\frac{B_s+B_f}{N}$). In the mesh a node does

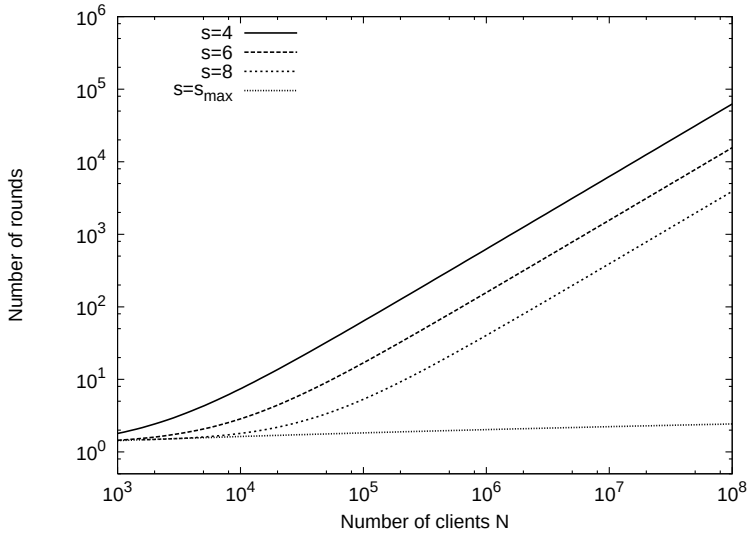


Figure 3.5: Download time (in rounds) of the linear chain architecture for different values of s ($k = 2$, $C = 10^2$).

not only receive data from its parent, but also from its siblings. The source has $2k$ fast peers as children and sends data at rate $\frac{B_f}{2k}$ to each of them; the remaining missing bandwidth $\frac{B_f}{2}$ is provided by their siblings. The first-level fast peers together serve k^2 children with their remaining bandwidth of $\frac{B_f}{2}$; again, the remaining bandwidth $\frac{k-1}{2k}B_f$ is provided by the siblings. Second-level peers have enough bandwidth to completely serve k^2 children. Each third-level child can in turn expand to k^2 peers in three steps.

As in the previous architecture, one can build linear chains after the expansion phase before reducing the architecture to one peer. The shrinking phase is symmetric to the growing phase, as shown in Figure 3.6.

Using only connections with identical rate $\frac{B_f}{2k}$ simplifies significantly the management of the architecture. The throughput is controlled by the source and peers only differ in their number of outgoing connections: the outdegree is always $2k$ for fast nodes and 2 for slow nodes. All peers have an indegree of $k + 1$.

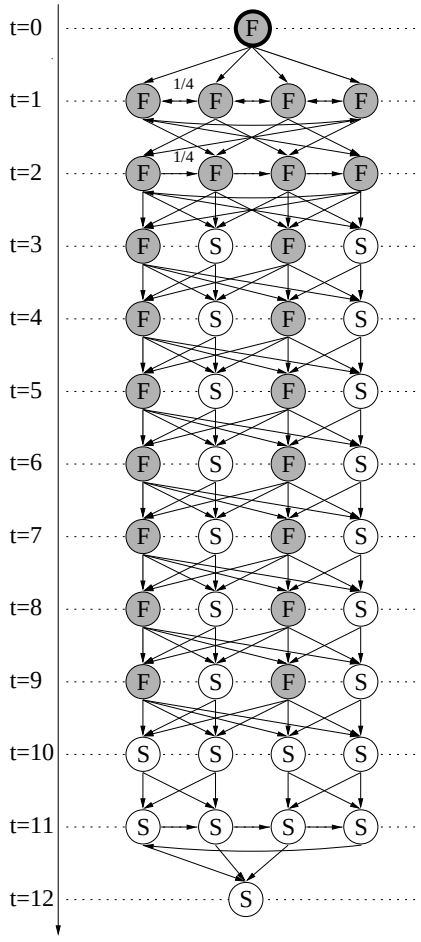


Figure 3.6: Mesh with one expansion step and $k = 2$ ($B_s = \frac{B_f}{2}$).

3.3.4 Analysis

One can note in Figure 3.6 that the first level fast peers receive chunks from the source at $t = 1$ and from their sibling at $t = 2$; similarly, second level peers receive chunks at $t = 2$ and $t = 3$; on the third level, all chunks are received simultaneously at $t = 3$. A similar observation can be made with the shrinking phase and it follows that constant delays of $\delta_T = 1$ are encountered in this content distribution architecture.

For computing the number of nodes which can be served in time T we again analyze the three phases. As we have seen, a fast peer can expand to k^2 peers in three units of time with the help of $2k + k^2$ other fast peers. If

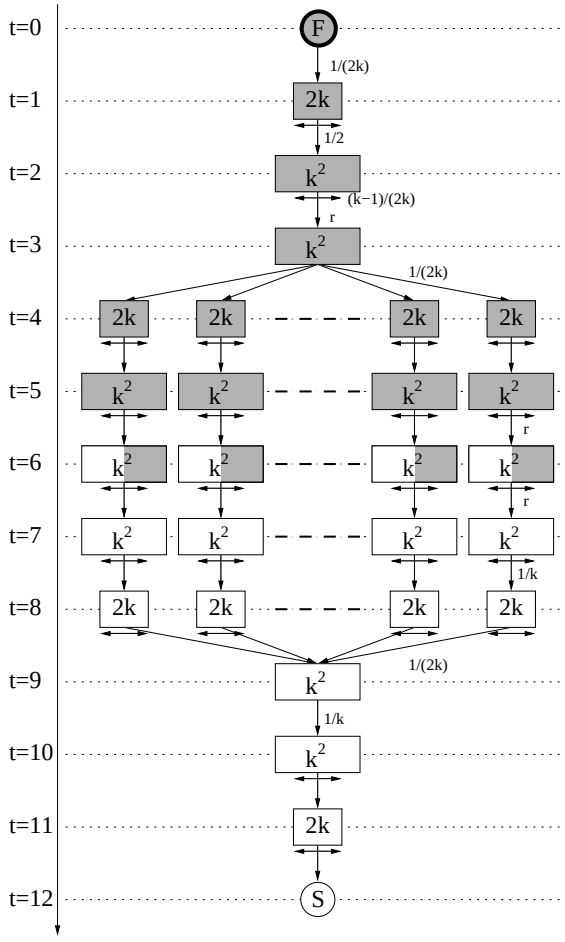


Figure 3.7: Mesh with two expansion steps and any k ($B_s = \frac{B_f}{k}$).

we define s to be the number of expansion steps, then the number of peers served in the first phase is:

$$N_1 = 1 + (2k + 2k^2) \sum_{i=0}^{s-1} k^{2i} = 1 + 2k \frac{k^{2s} - 1}{k - 1}$$

The shrinking phase again is symmetric in the number of nodes so the number of nodes in the third phase N_3 is equal to N_1, thus $N_3 = N_1$. Given the constraint that $N_1 + N_3 \leq N$ we can compute the maximal value of s:

$$s_{max} = \frac{1}{2} log_k \left(\frac{(N-2)(k-1)}{4k} + 1 \right)$$

In phase 2, $k^2 k^{2(s-1)}$ parallel nodes can be served in the remaining time $T - 6s - 1$. In total the number of peers served within T units of time for a given number of s expansion steps $1 \leq s \leq \lfloor s_{max} \rfloor$ is then:

$$N(T, s, k) = 2 + 4k \frac{k^{2s} - 1}{k - 1} + k^{2s}(T - 6s - 1)$$

Solving the equation for T and introducing the number of chunks C gives:

$$T(N, s, k, C) = 1 + \frac{1}{C} \left(\frac{1}{k^{2s}} \left(N - 2 - 4k \frac{k^{2s} - 1}{k - 1} \right) + 6s + 1 \right)$$

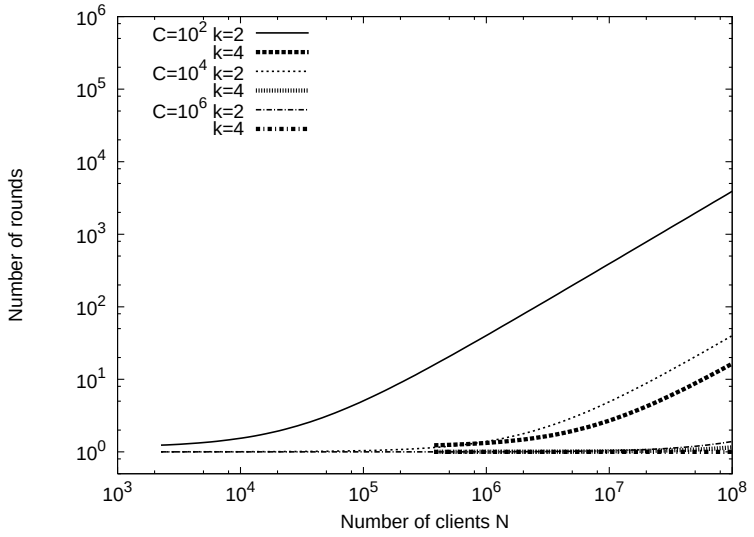


Figure 3.8: Download time of the mesh architecture for different values of C ($k = 2$, $s = 4$).

Figure 3.8 shows the time necessary to complete the download with the use of the mesh architecture for different values of C and k. As expected, the download times follow the same general shape as for the linear chains architecture in Figure 3.4 but performance is significantly improved due to

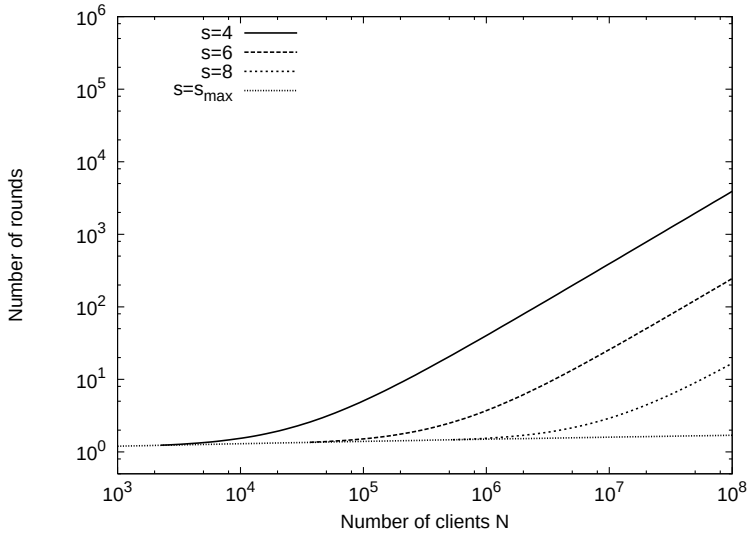


Figure 3.9: Download time of the mesh architecture for different values of s ($k = 2$, $C = 10^2$).

the faster expansion of the mesh architecture. We can observe in Figure 3.9 that a higher number of expansion steps s also produces flatter architectures and therefore reduces the download time. The maximal expansion for a given peer population s_{max} yields the best download times, which is almost constant, independent of the population size.

3.3.5 Parallel Trees

The third architecture studied in this chapter consists in constructing multiple trees spanning all the nodes and sending a separate part of the content in parallel to each tree similarly to SplitStream [8] and PTree^k [3] (as $N_f = N_s$, we shall use binary trees). If we construct $k+1$ trees that distribute content at rate $\frac{B_f}{2k}$, then every peer will receive data at the same uniform rate r.

We construct parallel trees by placing each fast peer (except the source) as interior node in k trees. Fast nodes will thus serve $2k$ other peers at rate $\frac{B_f}{2k}k$ (i.e., at aggregate rate B_f). The slow nodes are placed as interior nodes in a single tree and must thus serve two other nodes at rate $\frac{B_f}{2k}$ (i.e., at aggregate rate $\frac{B_f}{k}$). As the number of leaves in a complete binary tree

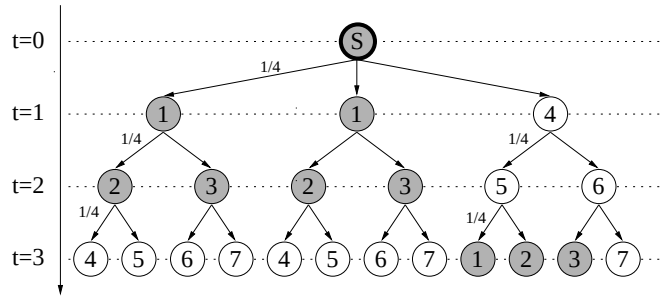


Figure 3.10: Parallel trees with $N = 8$ and $k = 2$ ($B_s = \frac{B_f}{2}$).

is equal to the number of interior nodes plus one and the source is a fast node, the constraint $N_f = N_s$ is met. Figure 3.10 illustrates the parallel tree architecture (peers are numbered for clarity). Note that every peer except the source appears in all trees.

3.3.6 Analysis

We first need to determine the depth d of the trees. At each level i in the tree, we have 2^i nodes (the root is at level 0). Thus, the number of nodes in a binary tree of depth d is $\sum_{i=0}^{d} 2^i = 2^{d+1} - 1$. Considering the special role of the source, the $N - 1$ remaining nodes can be placed in parallel trees of depth $d = \lfloor log_2(N-1) \rfloor$.

It follows from the construction of the trees that delays of $\delta_T = \lfloor log_2(N-1) \rfloor$ are encountered in this content distribution architecture. Delays grow with the number of peers, in contrast to the other architectures studied in this chapter.

The number of nodes that can be served by the parallel tree architecture in a given time interval T can be computed as follows (the first term represents the source):

$$N(T) = 1 + \sum_{i=0}^{T-1} 2^i = 2^T$$

Solving this equation to T and introducing the number of chunks C leads to the time used to distribute a file to all nodes:

$$T(C,N) = 1 + \frac{1}{C}\lceil log_2 N \rceil$$

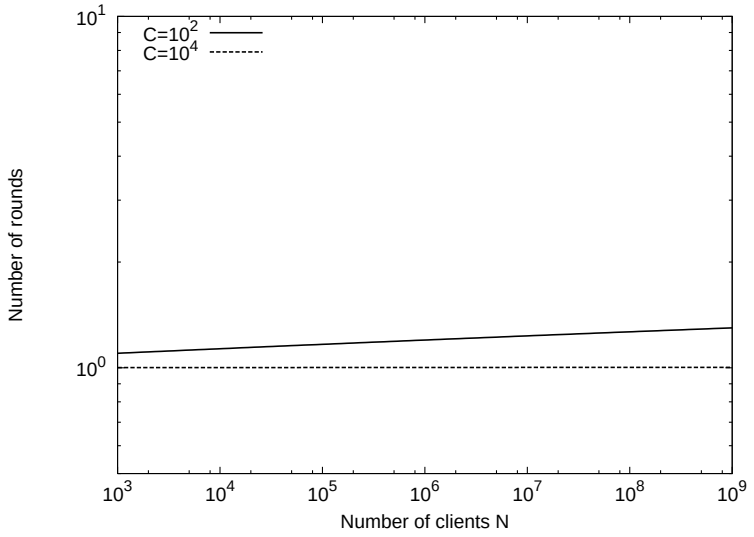


Figure 3.11: Download time for the parallel trees architecture for different values of C.

Figure 3.11 shows the time necessary to complete the download with the parallel tree architecture for two values of C (improvements become unnoticeable when C grows larger). As the download time is a function of the depth of the trees, which increases logarithmically with the number of peers, performance degrades only slowly with the population size.

3.4 Comparative Analysis

In this section we compare the three architectures presented in this chapter with the linear chain architecture analyzed in [7] (referred to as *Linear*). In contrast to our architectures, in *Linear* the peers have symmetric bandwidth capacities. The peers are organized in separate chains according to their bandwidth capacity and there is no cooperation between fast and slow nodes. Fast peers can therefore finish the download faster.

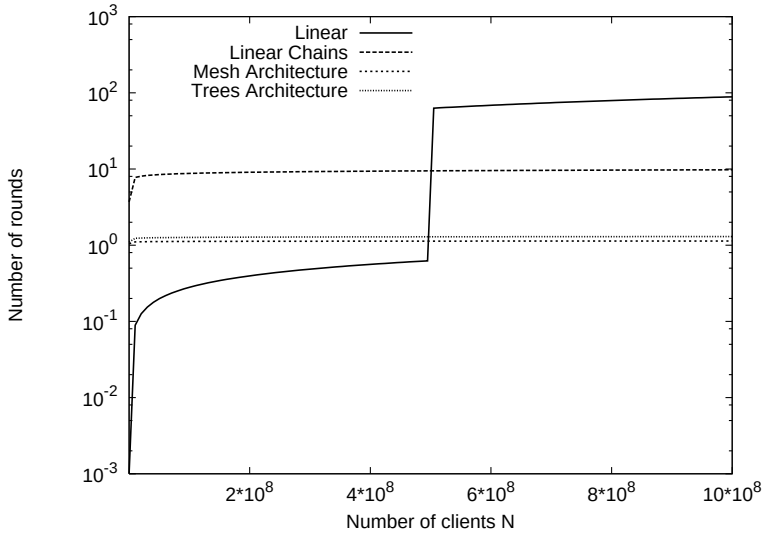


Figure 3.12: Download time for different architectures with $k = 100$, $C = 100$ and $s = s_{max}$. *Linear* shows the completion times for a population of 10^9 peers with symmetric bandwidth.

As we can see in Figure 3.12, this difference leads to a stepwise function with the fast nodes completing their download faster than the slow nodes ($N_f = N_s$). In contrast, the uniform architectures all scale well and yield an almost constant download rate independent of the population size. As expected, uniform linear chains are less efficient than the mesh and parallel tree architectures due to the longer paths.

In Figure 3.13 we can observe that with a smaller difference between fast and slow peers (lower value of k) the download time of *Linear* grows, whereas it decreases for the linear chains and the mesh architecture (remember that a unit of time is defined as a function of the uniform rate r). We can further see that the mesh architecture performs slightly better than parallel trees in Figure 3.12, unlike in Figure 3.13. This is due to the fact that the mesh architecture expands as a function of k^{2s} whereas the expansion of parallel trees does not depend on k. Thus the expansion in the mesh will grow faster when k is large. Higher values of C do not produce interesting results as the difference between the various architectures quickly becomes unnoticeable.

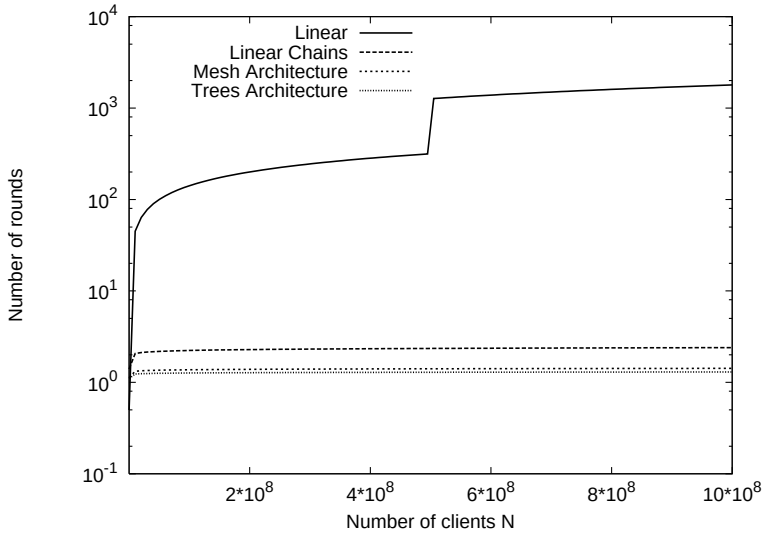


Figure 3.13: Download time for different architectures with $k = 4$, $C = 100$ and $s = s_{max}$. *Linear* shows the completion times for a population of 10^9 peers with symmetric bandwidth.

3.5 Summary

In this chapter, we have studied the problem of providing uniform download rates to a population of peers with asymmetric and heterogeneous bandwidth capacities. The architectures that best achieve this goal among those studied in the chapter are the mesh and the parallel tree, but the latter requires peers to buffer data for a duration proportional to the depth of the trees. As the number of chunks grows, i.e., when the stream duration becomes very long, the differences between all the architectures become insignificant.

Although we only focused on analytical models for simple content distribution architectures, we believe that our analysis provides some important insights as how to set up peer-to-peer networks for distributing streaming data. It can also guide the design of cooperative applications that organize the nodes in a more dynamic manner than chains or trees. In particular, the system needs to build up upload capacity as fast as possible (which corresponds to maximizing the number of expansion steps) and the content should be partitioned into a large number of chunks (but not too many chunks as

each one adds some coordination and connection overhead). By properly combining high and low capacity nodes, one can provide a high initial quality of service to every peer and even out their differences in a truly cooperative manner.

The models used in this chapter assumed that no nodes fail and that the capacity remains stable over time. In a real environment peers are not stable and often fail or refuse to participate in distribution. Further network capacities change over time due to change of usage. Both of these challenges are analyzed in the following chapters based on the models introduced in this chapter.

Chapter 4

Self-organization in Cooperative Content Distribution Networks

4.1 Introduction

The architectures presented in Chapter 3 are constructed in one pass and are not adapted to conditions that change afterwards. The environment can change due to peers failing or bandwidth that is fluctuating because of varying link usage.

In contrast to Chapter 3, in this chapter, we aim at providing techniques that are *efficient* in heterogeneous settings, *adaptive* so as to tolerate run-time changes like bandwidth fluctuations, and *practical* enough to be implementable in real systems. For the sake of simplicity, our study mostly focuses on architectures with binary trees; the principles and algorithms presented here do, however, also apply to other architectures, as will be discussed later.

The main metric we consider is the average time for each of the clients to

Parts of this chapter have been published in: M. Schiely, L. Renfer and P. Felber. Self-organization in Cooperative Content Distribution Networks. *Proceedings of the IEEE International Symposium on Network Computing and Applications (NCA'05)*, July 2005.

receive the complete content. Earlier studies [3, 59] have developed analytical models and indicated theoretical limits for this problem, but they only considered homogeneous scenarios where all the peers have identical bandwidth. In particular, a comparison of several distribution architectures based on linear chains, trees, and parallel trees, has indicated that performance can be maximized if all the peers can use their upload capacity and the content is split in enough small blocks so that the peers are all active at the same time.

The contributions of this chapter are as follows: We first analyze the problem of cooperative distribution of content from a single source to a large number of heterogeneous clients and we identify the limitations of existing solutions. We propose techniques and algorithms that dynamically optimize the distribution network, based on the observed effective bandwidth capacities, in order to avoid bottlenecks and improve global throughput. These algorithms have several desirable features. Most notably, they are fully decentralized and work by only performing local reorganizations; as such, they might stop short of producing an optimal configuration, but perform extremely well under the aforementioned constraints. We analyze the properties of our algorithms and we evaluate them by the means of simulations, as well as experimentally in a LAN and in the Internet using the PlanetLab [39] testbed.

The chapter is organized as follows: We first present classical tree-based distribution architectures and analyze their shortcomings in Section 4.2. Section 4.3 introduces the principles, mechanisms, and algorithms proposed to dynamically improve the efficiency of tree-based content distribution. Section 4.4 presents results from simulations and experimental evaluation, and Section 4.5 summarizes the chapter.

4.2 P2P Content Distribution

Tree-based Architectures. Different architectures have been developed for organizing clients in a P2P fashion for cooperatively distributing content, e.g., a large file. The key idea is to have clients that have already downloaded the file help redistribute it to other clients, instead of relying on a single

source. The time necessary to send the file to all peers is not anymore proportional to the number of clients in the network as for classical client-server distribution, but proportional to the logarithm of the number of peers.

As an example, consider the situation where a server must replicate a critical file, e.g., an antivirus update, to all $100,000$ machines of a large company. Given a file size of 4 MB and a server (client) bandwidth capacity of 100 Mb/s (10 Mb/s) with 90% link utilization, a classical client/server distribution protocol would distribute the file by iteratively serving groups of 10 simultaneous clients in $u = \frac{32 \text{ Mb}}{9 \text{ Mb/s}} = 3.55$ seconds. Updating $100,000$ clients would thus necessitate $\frac{100,000}{10}u$, i.e., almost 10 hours.

In contrast, cooperative distribution leverages the bandwidth of the nodes that have already obtained the file, thus dynamically increasing the service capacity of the system as the file propagates to the clients. As each client that has already received the file can serve another client while the server updates 10 new clients, we can compute the number of clients updated at time t as $n(t) = 2n(t-u) + 10 = 2^{\lfloor t/u \rfloor}10 - 10$. Updating $100,000$ clients would thus necessitate less than 1 minute. The exponential increase of the number of served peers provides a sharp contrast with the linear progression of traditional client/server distribution (see [19] for a more detailed analysis).

The simplest architecture for cooperative content distribution consists in forming a chain (or pipeline) in which each client downloads the file from one peer and uploads it to another peer. The file is divided into small blocks of a given size that can be transmitted independently from each other: as soon as a block is received at one peer, it is forwarded to the next peer. This architecture leads to impressively short distribution times in high speed networks with full duplex connectivity. The total distribution time is essentially the time to send the whole file to the first node plus the delay for the first block to reach the last node.

If each peer serves more than one other peer, we obtain trees instead of linear chains. As the bandwidths of upload connections have to be shared between several downloaders, such architectures are best adapted in settings where peers (especially those close to the source) have large upload capacities.

Chains and tree architectures have the disadvantage that the failure of a node adversely impacts the whole subtree rooted at that node. Indeed, once

the only link to the subtree is broken, no data can flow to any of its peers. To address this problem, one can organize the peers into multiple spanning trees, with each peer belonging to all the trees and being interior node of at most one of them, and have the source send distinct blocks to each tree. Such architectures based on parallel trees have been used in SplitStream [8] to improve bandwidth efficiency and increase robustness. Obviously, the failure of a peer will affect at most one of the distribution trees and leave the rest operational. Analytical models and analysis of these architectures in homogeneous settings can be found in [3]. We shall primarily focus on architectures based on a single binary tree in the rest of the chapter, although we shall briefly discuss extensions for n-ary and parallel trees.

Dealing with Heterogeneity. The performance of content distribution using a single tree composed of peers with heterogeneous bandwidth directly depends on the organization of the nodes in the tree. One slow peer p_s can increase the average reception time of all the peers in the subtree rooted at p_s, even if they have more bandwidth and computational power than p_s.

To show the effect of a single slow peer p_s in a balanced binary distribution tree of n nodes, we compute the average reception time depending on the position of p_s in the tree. We assume a symmetric bandwidth of B_f for the fast peers and the source S, and $B_s < \frac{B_f}{2}$ for the slow peer. To distribute a file of size F, we divide it into blocks and send them along the tree as a continuous stream of data. We shall neglect the delay of the first block to reach the bottom of the tree, as its impact on the average reception time of the file by the peers is negligible. We also assume that each peer stores the blocks locally and, hence, does not need to buffer communication flows. If we construct a tree composed only of fast nodes, each of them downloads the file in time:

$$T = \frac{2F}{B_f}$$

Distribution occurs at half the available bandwidth because each peer has to serve two other peers on a single link. T, in this case, also equals to the average download time $\overline{T}$ among all the peers.

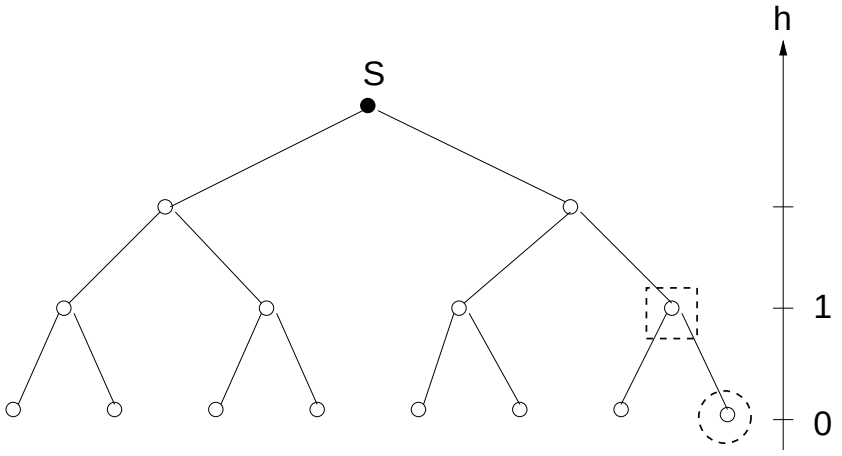


Figure 4.1: Positions of a slow node in the binary tree.

If we have now one slow peer at the bottom of the tree (node surrounded by a circle in Figure 4.1), then $\overline{T}$ becomes:

$$\overline{T}(n) = \frac{n-2}{n}\frac{2F}{B_f} + \frac{1}{n}\frac{F}{B_f - B_s} + \frac{1}{n}\frac{F}{B_s}$$

The first term in the equation refers to the peers which are not affected by the bandwidth limitation of the slow peer. The second and the third term correspond to the download times of the sibling of the slow peer and the slow peer, respectively.

If the slow peer is at the second level from the bottom of the tree (node surrounded by a square in Figure 4.1), $\overline{T}$ now becomes:

$$\overline{T}(n) = \frac{n-4}{n}\frac{2F}{B_f} + \frac{1}{n}\frac{F}{B_f - B_s} + \frac{1}{n}\frac{F}{B_s} + \frac{2}{n}\frac{2F}{B_s}$$

Again, we have the unaffected peers in the first term, and the second and third terms refer to the sibling of the slow peer and the slow peer itself. The last term corresponds to the download time for the children of the slow peer.

If we generalize the average download time per peer depending on the height h (from the bottom of the tree) of the slow peer, we get the following expression for $\overline{T}$:

$$\overline{T}(n,h) = \frac{F}{n}\Big(\frac{(n-2^{h+1})2}{B_f} + \frac{1}{B_f - B_s} + \frac{1}{B_s} + \frac{(2^{h+1}-2)2}{B_s}\Big)$$

As previously mentioned, the download time can be improved when using parallel trees, with each peer being interior node of at most one of the trees (only one peer can be a leaf of all trees). In such architectures, the position as interior node of the slow peer will also affect the average performance of file distribution. With a parallel binary tree configuration, half of the file is sent in parallel to each tree and the download performance is obviously limited by the tree in which the slow peer is an interior node, i.e., has the highest position. In that case, we can compute the average download time as:

$$\overline{T}_{\parallel}(n,h) = \frac{F}{n}\Big(\frac{n-2^{h+1}+1}{B_f} + \frac{2^{h+1}-1}{B_s}\Big)$$

In Figure 4.2 we can see the effect in binary tree configurations of one or two slow peers depending on their height. Computations were performed with $B_f = 100$ Mb/s, $B_s = 10$ Mb/s and 1 Mb/s, $F = 650$ MB, and $n = 2^{17}-1$ peers (including the source). In settings with two slow peers, each of them was in a different subtree from the source. Figure 4.2 shows a clear exponential increase in the average reception time $\overline{T}$, both with single and parallel trees, after the height of the slow nodes reaches approximately half the depth of the tree. This clearly demonstrates the necessity of dynamically reorganizing distribution trees to adapt to the effective bandwidth of the peers.

4.3 Dynamic Reorganization Algorithm

Motivations and Design Guidelines. In the previous section we have shown that, in heterogeneous settings, slow nodes should be located as deep

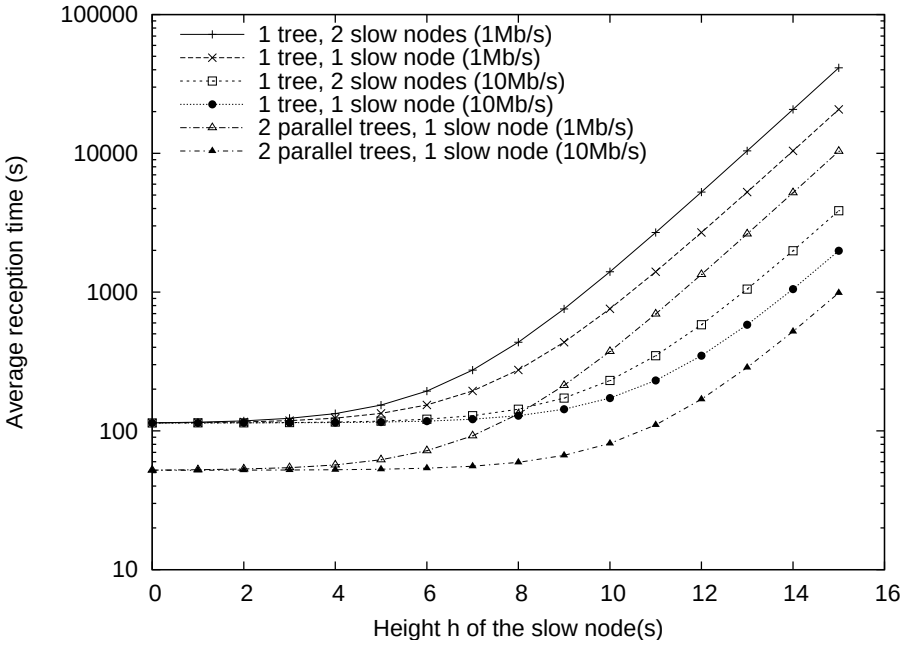


Figure 4.2: Average reception time depending on the height of slow nodes.

as possible in tree-based content distribution architectures. Indeed, a slow node is a bottleneck for its whole subtree and the higher the position of the slow node in the tree is, the more peers its subtree contains.

Therefore, our goal is to design an algorithm that dynamically optimizes the distribution tree by reorganizing peers according to their effective bandwidth. This directly raises the problem of estimating bandwidth capacities and moving peers at runtime in a practical and efficient manner.

Our algorithm was designed according to several guidelines: it should be fully distributed and symmetric, and not rely on a centralized entity (besides the data source that has a specific role); all operations and reorganizations should be performed locally or in the close neighborhood of a peer; decisions should be based on local information and no global knowledge should be necessary; the algorithm should be able to adapt dynamically to changes in the network; and the complexity and overhead should remain as low as possible.

These guidelines comply with the P2P design philosophy and are key

to achieve high scalability. A consequence of the constraints they impose is that our algorithm may not yield an optimal configuration, which would necessitate non-local information and operations, as we shall discuss shortly.

Bandwidth Measurements. The limiting factor in most file distribution networks is the upload capacity of the nodes, which is typically lower than their download capacity (e.g., ADSL). Therefore we based our algorithm on the upload capacities of the nodes and we reorganize the peers when we detect nodes that have lower upload capacities than some of their children.

Each peer p must be able to estimate its upload capacity u. To that end, a node actively or passively measures the throughput u_i achieved when uploading data only to child i, and the throughput u_n obtained when uploading data simultaneously to all m children (see Figure 4.3). Further, let $d_i > 0$ be the download capacity of child i.

Based on these measurements, we can distinguish two cases:

1. $u_n < \sum_{i=1}^{m} u_i$ with $u_j = u_n$ for some nodes j and $u_k < u_n$ for some nodes k

2. $u_n = \sum_{i=1}^{m} u_i$

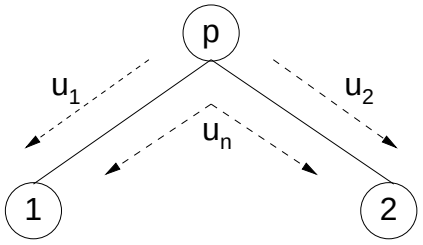


Figure 4.3: Throughput measured to estimate effective bandwidth.

In case 1 the transfer bandwidth is limited by the upload capacity of peer p. The upload capacity to all nodes u_n is not higher than the upload rate to a subset of its children. We estimate the upload rate of p to be $u = u_n$. We also know that each child j has a download rate of $d_j \geq u_j$ and each child k has a download rate of $d_k = u_k$.

Case 2 occurs if the upload capacity u of p is not the limiting factor. The children are all downloading at their limits. We have $u \geq u_n$ and we know that each child i has a download rate of $d_i = u_i$.

Based on these estimations, a peer can easily compare its upload capacity with that of its direct neighbors to determine whether local reorganizations are necessary.

The *HeapTop* Algorithm. *HeapTop* is remotely inspired from the well-known HeapSort algorithm, where the nodes of a tree are reorganized by exchanging selected father-child pairs. The goal is to move the nodes with highest upload bandwidth closest to the root of the tree. The property maintained by our algorithm is that, for every node p other than the root and every child c of p, we have $u_p \geq u_c$ (with u_p and u_c being the effective upload bandwidth of p and c, respectively).

As we only want to perform local operations, the only way we can reorganize the tree is by exchanging the position of a node with its parent. This operation can be easily implemented because both nodes are directly connected with each other and they essentially have to exchange their respective neighbors.

The algorithm starts with a random initial tree. We assume that all nodes in the tree can estimate their bandwidth capacity and that of their parent, as previously discussed.

Algorithm 1 *HeapTop* algorithm at peer p

```
1: loop
2:    q ← Parent(p)
3:    if q ≠ root and Bandwidth(q) < Bandwidth(p) then
4:       Exchange positions of p and q
5:    end if
6: end loop
```

Each node continuously executes the trivial operations shown in Algorithm 1. Peer p periodically compares its bandwidth capacity with that of its parent. If p's bandwidth is strictly bigger than its parent's bandwidth, then they switch positions, i.e., they exchange their neighbors. This operation can be performed efficiently as it is essentially local to p and its parent.

The algorithm preserves the structure of the initial tree (even if it is not balanced), but the position of the nodes evolves over time.

For avoiding pairwise exchanges resulting from short bandwidth fluctuations, the estimations are based on a weighted moving average computed using the following formula:

$$u(t) = (1 - \alpha) \cdot u(t-1) + \alpha \cdot u$$

The average bandwidth at time t is obtained by combining the latest sample u with the previous average value. The constant $\alpha \leq 1$ (typically $\frac{1}{8}$) is a smoothing factor that puts more weight on recent samples than on old samples and smooths out variations.

In addition, in order to prevent unnecessary reorganizations of peers with similar bandwidth capacities, we shall only exchange the position of a peer p and its parent q if $u_q < \beta \cdot u_p$, with $\beta \leq 1$ (typically $\frac{9}{10}$).

Note there is no synchronization between the peers (except between pairs of neighbors when positions need to be exchanged). This implies that nodes can move upward or downward the tree at different speeds, and distinct configurations can be obtained from the same initial tree. Figure 4.5 shows a possible configuration obtained from the execution of the algorithm on the tree in Figure 4.4 (the numbers indicate the bandwidth capacities of the peers: large numbers correspond to high bandwidth).

Given the special role of the root node, it appears clearly that the peers cannot move from one 1^{st}-level subtree to another 1^{st}-level subtree. Further, within any subtree, a node in one branch may be farther from the root than some other node with less bandwidth in another branch (see nodes 9 and 10 in Figure 4.5). As such, the resulting distribution tree may be sub-optimal but performing further optimizations would necessitate non-local operations and higher complexity.

If there is no bandwidth fluctuation, the tree will quickly reach a stable configuration. In the worst case, a node located at depth $d \geq 1$ (the root is at depth 0) can initiate $d - 1$ exchanges. The actual number of exchanges depends on both the initial configuration of the tree and the order in which exchanges are performed.

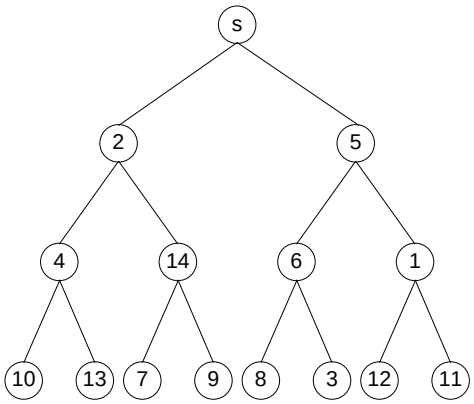


Figure 4.4: Original distribution tree (larger numbers mean more bandwidth).

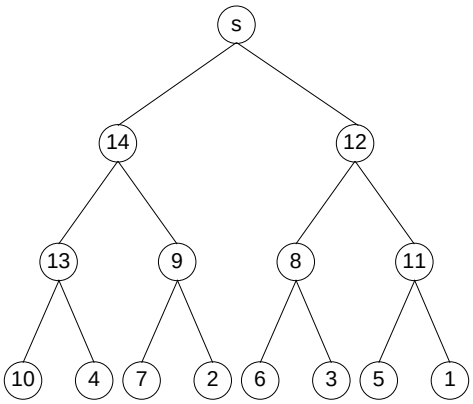


Figure 4.5: One possible configuration obtained from executing the algorithm.

Note that this algorithm can also be used with architectures based on parallel trees. Node exchanges are performed concurrently in each of the trees. If one wishes to meet the robustness property that a peer should be interior node of at most one tree, we lose some flexibility in the way the trees can be organized: exchanges can only be performed if the robustness property still holds after the operation (only interior nodes can be freely exchanged). Although the resulting architecture provides better resilience to failures, it will be sub-optimal in terms of bandwidth efficiency.

4.4 Evaluation

Simulation Setup. For evaluating the behavior of *HeapTop* in different environments, we implemented a Java simulator that faithfully reproduces the operations of the algorithm and evaluates its efficiency. The main criterion considered is the average upload bandwidth capacity using the tree generated by *HeapTop*, as compared with that of the initial randomly generated tree and of an optimal tree.

We have simulated three main classes of peers, chosen to match the observations we have made of real-world populations in an earlier study of the BitTorrent protocol [25]. These classes represent effective connection

throughputs frequently encountered in the Internet:

- F: fast nodes with 1024 Kbit/s upload bandwidth.
- M: medium-speed nodes with 512 Kbit/s upload bandwidth.
- S: slow nodes with 128 Kbit/s upload bandwidth.

As previously mentioned, the upload bandwidth is the limiting factor and we do not explicitly take into account download capacities (peers of classes M and S typically have asymmetric bandwidth).

Each peer has a given probability to fall in one of the considered classes. Binary trees are constructed by iteratively adding each node at a valid position, chosen by traversing the tree from the root until a leaf or a node with a single child is encountered. We experimented with both unbalanced and balanced trees. As the differences in the measurements were negligible, we only show results for balanced trees and note that they are also valid for unbalanced trees.

For comparison with an optimal configuration, a tree was constructed by organizing the nodes from root to leaf in decreasing order of upload capacity. Each result is the average of 50 executions.

Simulation Results. We have first evaluated the improvement factor of *HeapTop* with different population sizes and various proportions of nodes in each class. To that end, we have used the class distributions shown in Table 4.1.

	Class F	Class M	Class S
$D1$	90%	5%	5%
$D2$	60%	30%	10%
$D3$	50%	25%	25%
$D4$	30%	60%	10%
$D5$	25%	25%	50%
$D6$	5%	90%	5%
$D7$	5%	5%	90%

Table 4.1: Distributions of peer classes for the simulations.

The improvement factor f is defined as the ratio of the average bandwidth $\overline{B}_{HT}$ of the tree generated by *HeapTop* to the average bandwidth $\overline{B}_R$ of the random initial tree: $f = \overline{B}_{HT}/\overline{B}_R$.

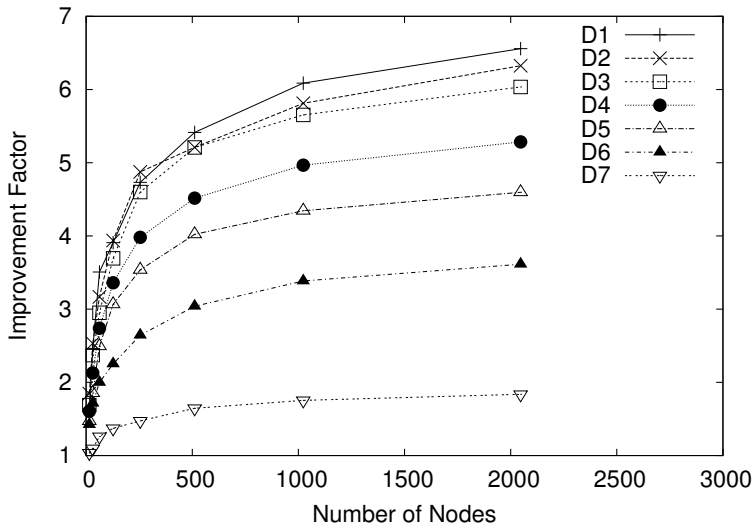


Figure 4.6: Average improvement factor for different population sizes and various class distributions.

Figure 4.6 shows that the improvement factor is significant, with *HeapTop* being as much as 6 times more efficient than the initial tree. Further, it increases with a logarithmic behavior as the number of nodes grows. This can be explained by the analysis of Section 4.2, which showed that with a single slow node located at height h, the performance of the whole network degrades as a function of 2^h. As the height of a binary tree composed of n peers is proportional to $log(n)$, the logarithmic shape of the improvement factor is not surprising.

One can also observe that the difference between *HeapTop* and the initial tree decreases when there are many slow peers, as there is less room for optimization (some slow peers must end up as interior nodes).

Another desirable property of *HeapTop* is to maintain the average number of exchanges per node as small as possible. As one can see in Figure 4.7, this value mostly depends on the class distribution and is not higher than 1.2 exchanges per node. The size of the peer population, i.e., the tree depth, has

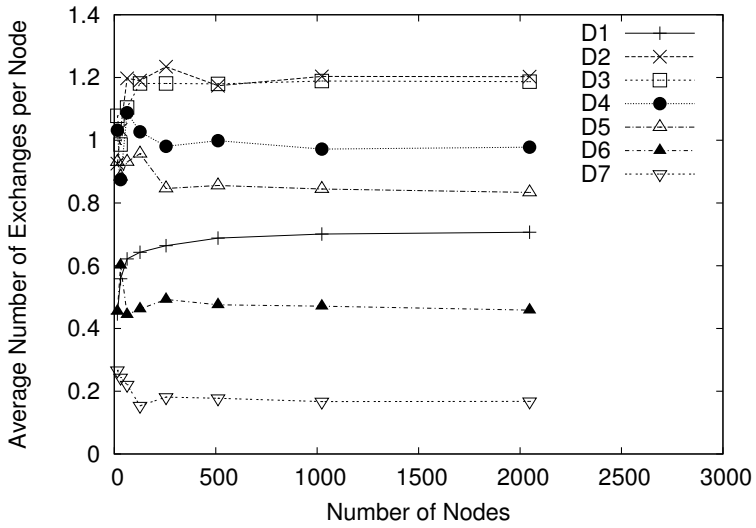


Figure 4.7: Average number of exchanges per node for different population sizes and various class distributions.

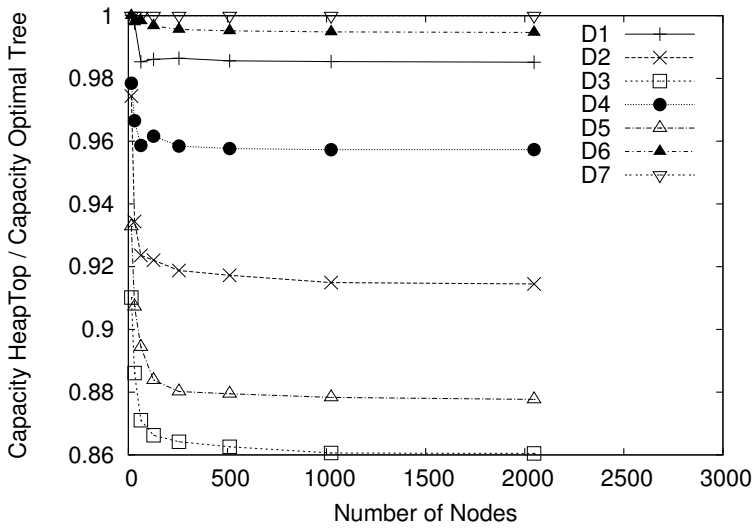


Figure 4.8: Bandwidth capacity of the *HeapTop* tree vs. an optimal binary tree for different population sizes and various class distributions.

little impact on that metric.

As discussed in Section 4.3, *HeapTop* does not generate optimal trees because it only performs local reorganizations. Figure 4.8 shows that the

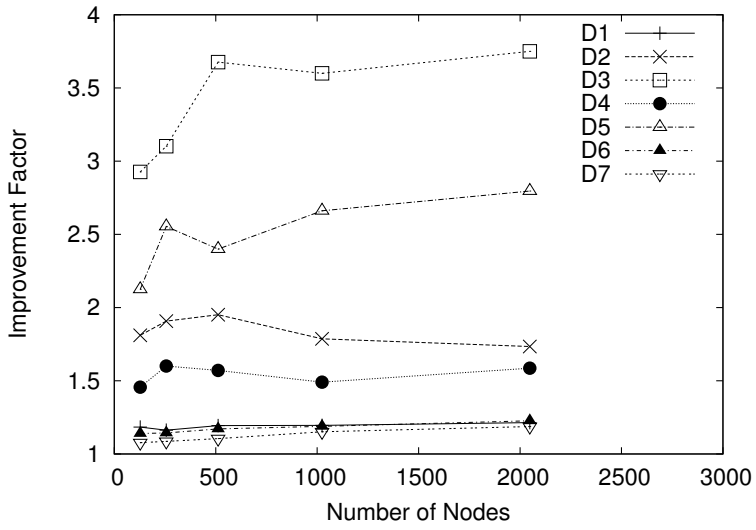


Figure 4.9: Average improvement factor with two parallel trees for different population sizes and various class distributions.

constructed trees are close to the optimum (more than 0.95 for most configurations) and do not depend much on the size of the peer population. Taking into account the simplicity and efficiency of the algorithm, this is clearly an acceptable approximation of the optimal tree.

We also simulated *HeapTop* with an architecture based on two parallel binary trees. As in SplitStream, we enforced each peer to be inner node of at most one of the trees. After generating both trees, *HeapTop* was run on the inner nodes of each tree. Figure 4.9 shows the improvement factor for different population sizes and various class distributions. One can observe that the gain is still significant (up to almost 400%). Further, the relative performance of the class distributions is different than for a single tree because only interior nodes can be reorganized. Figure 4.10 shows the best improvement factor observed during the simulations (up to 750%) and gives a measure of the potential benefits of *HeapTop* for parallel trees.

Experimental Setup. To evaluate experimentally the effect of the *HeapTop* algorithm, we have developed a content distribution tool called `crcp` (cooperative remote copy) that implements P2P replication of files to large

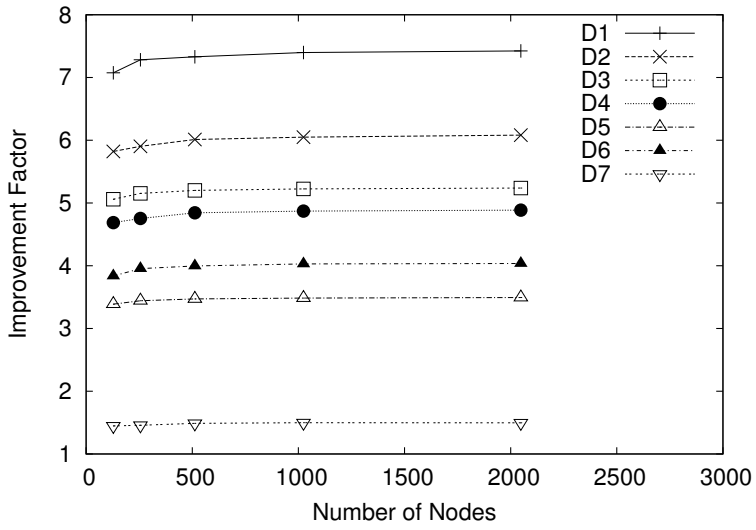


Figure 4.10: Best case improvement factor for two parallel trees for different population sizes and various class distributions.

populations of hosts. Each file is split in blocks that are sent independently. The current version of `crcp` supports linear chain and tree architectures, which are dynamically constructed by the source when initiating file replication.

Experimental Results. We have first evaluated our mechanisms in a local area network (LAN), with 13 Linux computers connected to a switch, one of them acting as the source and the rest as clients. Six of the client peers had network cards configured at 10 Mb/s, the other 6 and the source at 100 Mb/s. The file to distribute had a size of 564 MB. To demonstrate the efficiency of the *HeapTop* algorithm the file was distributed on trees according to the four configurations on Figure 4.11, where a slow node moves down the tree to improve distribution efficiency.

The average reception times are shown in Figure 4.12. As expected, file distribution becomes more efficient when the slow node is deep in the tree. This confirms that the *HeapTop* algorithm achieves better performance for file distribution than a fixed tree construction by moving slow nodes far from the source.

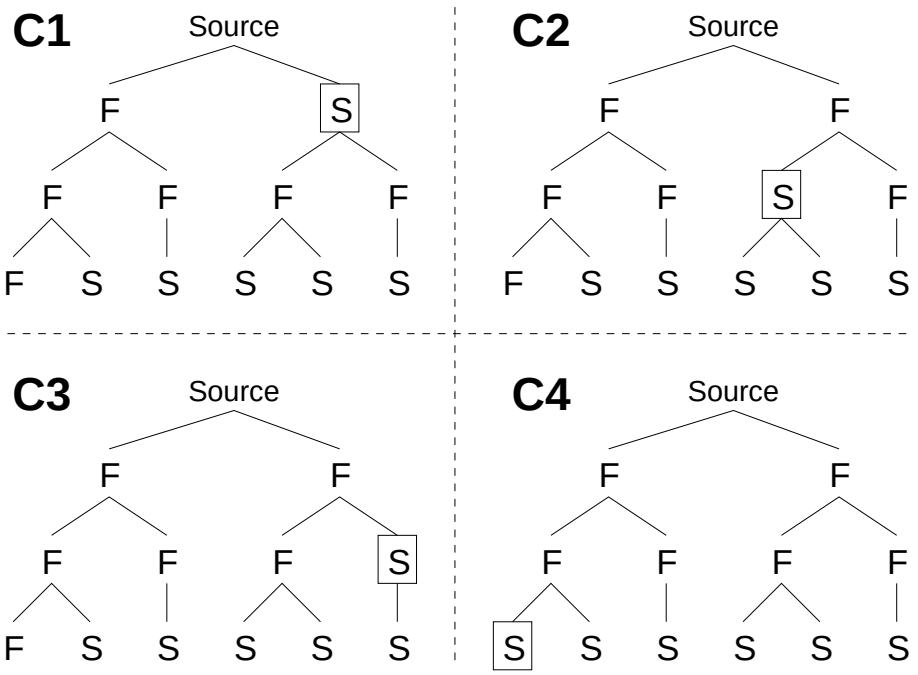


Figure 4.11: Configurations for average reception time tests with `crcp`.

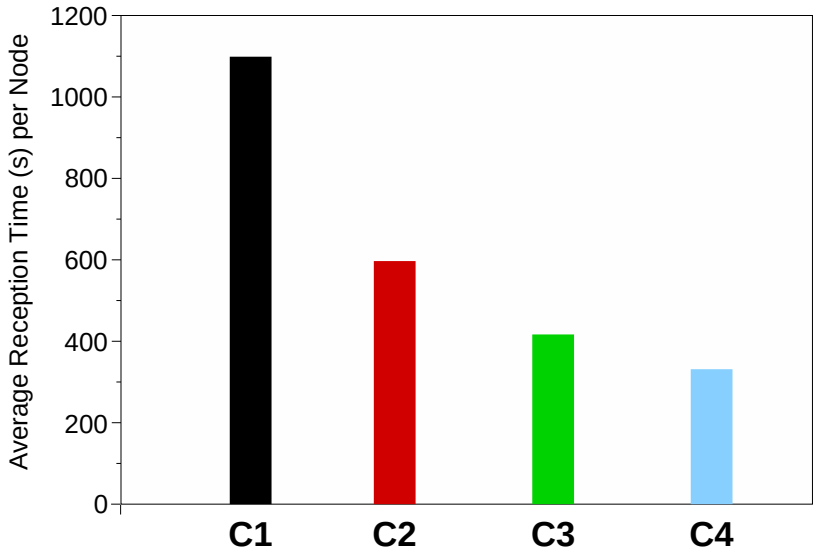


Figure 4.12: Average reception times with a slow node at different positions.

We have then performed large-scale experiments with `crcp` on 25 hosts of the PlanetLab infrastructure and compared the performance of initial random trees and the trees obtained using the *HeapTop* algorithm. Although we observed some variance in the experiments, due to load fluctuations in the network and at the nodes, *HeapTop* produced trees that were systematically faster than the initial configurations, with an average improvement factor of 1.55 and peaks of over 1.70 which corresponds to the simulation results for 25 nodes shown in Figure 4.6.

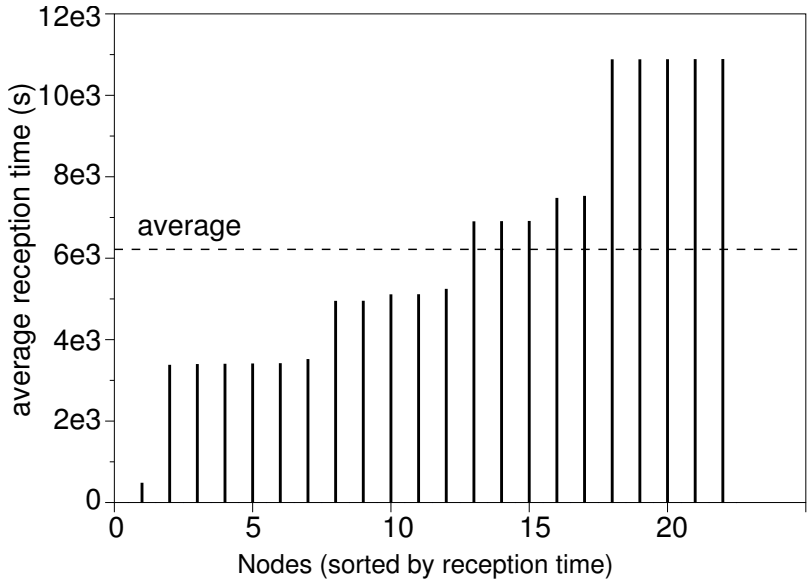


Figure 4.13: Reception times of the peers for the initial random binary tree.

A careful look at the reception times of each of the nodes helps us to understand the reason for this improvement. Figures 4.13 and 4.14 show the performance of individual peers, sorted by reception times, when sending a 29 MB file to 22 hosts, for the initial and *HeapTop* trees respectively. We can observe that, in the former case, low-bandwidth peers slow down their descendant, which produces clear steps in the figure. Such bottlenecks do not appear in the latter case, as many of the peers can download the file with no speed limitations besides their own bandwidth. Further study would be necessary to observe how *HeapTop* dynamically adapts to the bandwidth fluctuations and unpredictability of the Internet, and how it could be ex-

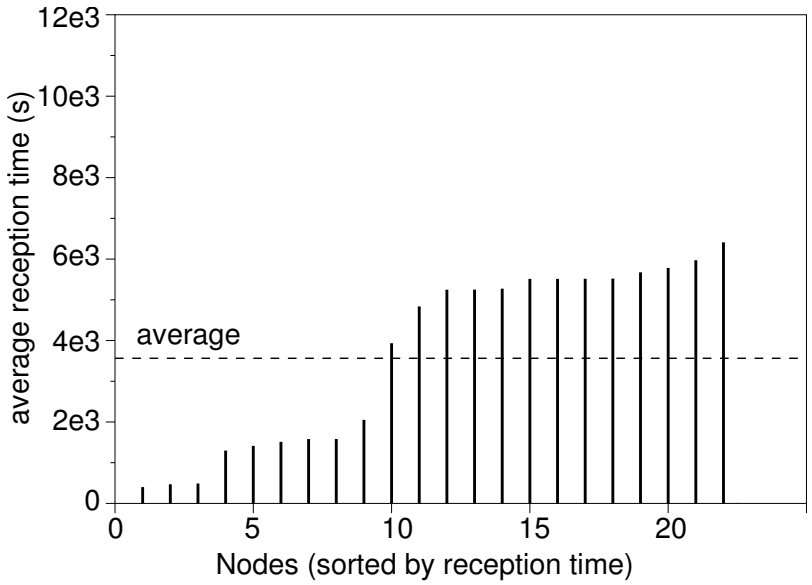


Figure 4.14: Reception times of the peers for the *HeapTop* binary tree.

tended to explicitly deal with failures.

4.5 Summary

In this chapter, we have studied the limitation of classical tree-based architectures when peers have different bandwidth capacities. We have proposed simple and efficient algorithms to dynamically reorganize the peers so as to optimize distribution efficiency. These mechanisms are adaptive, decentralized, and only perform local reorganizations; as such, they follow the P2P design philosophy and are extremely scalable. We have extensively studied their effectiveness by the means of simulations and experimentations and we have observed significant efficiency gains (up to more than 600%) depending on the number of peers and their respective bandwidth. These results demonstrate the importance of explicitly taking into account bandwidth limitations and fluctuations in P2P content distribution architectures, in order to avoid wasting the most essential resources of the network—the service capacity of the peers.

The presented algorithms do not explicitly take into account the problem

of peer failures. If the system is constructed of a high number of unstable peers then the algorithms may not be very efficient as it takes some time for peers to propagate through the tree.

The following chapter will handle the problem of failing peers and propose a mechanism to stabilize a P2P system by rewarding participating peers by more reliability.

Chapter 5

Tit-for-tat Revisited: Trading Bandwidth for Reliability in P2P Media Streaming

5.1 Introduction

The architectures and algorithms presented in Chapters 3 and 4 only work in environments where peers are ready to service other peers with their upload bandwidth and do not fail.

In real networks peers are under the control of randomly acting users and typically consist of low-end hardware, so failures are expected to occur often. Therefore, P2P architectures must be able to withstand a significant number of ungraceful failures and recover with no interruption of the time-sensitive stream. Peers that do not contribute to the system can also cause significant degradation of the streaming quality and must be handled by P2P systems.

Parts of this chapter have been published in: M. Schiely and P. Felber. CrossFlux: An Architecture for Peer-to-Peer Media Streaming. In *R. Baldoni, G. Cortese, F. Davide, A. Melpignano (Eds.): Global Data Management, Volume 8, Emerging Communication: Studies on New Technologies and Practices in Communication*, pp. 342-358, IOSPress, 2006 and in M. Schiely and P. Felber. Tit-for-tat revisited: Trading bandwidth for reliability in p2p media streaming. In *Multiagent and Grid Systems, Volume 5, Number 2*, pp. 197-215, 2009.

In BitTorrent [17], contribution is enforced by a data-based tit-for-tat mechanism. A peer p only sends chunks to peer q if in return it also receives data from q, except for the startup phase to bootstrap the system. This kind of tit-for-tat is not well adapted to media streaming, as it introduces additional delays and may not be feasible as in media streaming chunks are distributed sequentially in time-order. A peer q that receives chunks from peer p may never have useful chunks for p as it is streaming content that p already displayed. Therefore our architecture, CrossFlux, integrates a new approach for tit-for-tat that implements fairness by essentially trading bandwidth for reliability.

In CrossFlux the initial stream is split into multiple stripes, which are distributed across a communication graph interconnecting the peers—actually a forest of overlapping trees, as there is a single source. The tit-for-tat approach uses a link between two peers p and q in two different ways: (1) under normal operation, the link is used to transmit a single stripe s_i from p to q; (2) in case of a failure of a peer on the path from the source to p for stripe s_i, it can also be used in the reverse direction to send other stripes (different from s_i) from q to p (backup link). This mechanism guarantees path redundancy between the source and any peer, and it rewards the peers with a high outdegree (big contributors) by providing them increased reliability (more backup links).

For further enhancing the streaming quality, the construction process of the distribution topology places nodes in branches that are less loaded. The adaptive algorithm of Chapter 4 is later used to better distribute the load among the peers and to give sufficient bandwidth to each of them. Dynamic reorganization is performed locally by the nodes in order to enhance the topology: nodes that have higher bandwidth capacities are moved closer to the source by pairwise exchanges with lower capacity nodes.

The evaluation of CrossFlux shows that recovery of node failures is fast due to the backup links, efficiency is increased with the help of self-adaptive techniques, and the load is well distributed among the peers.

A number of approaches have been proposed to deal with the stringent requirements of media streaming (see Chapter 2). Most of them do, however, suffer from some drawbacks: (1) the architecture is rigid and does not

adapt well to the dynamics of the network; (2) recovery from peer failures is not very fast; or (3) the system is not well adapted to heterogeneity in node bandwidth. CrossFlux uses several novel techniques to address these issues.

This chapter is structured as follows. In Section 5.2 the CrossFlux architecture and its overlay management and content distribution algorithms are described. Then, Section 5.3 presents results from simulations and experimental evaluation on the PlanetLab [39] testbed. Finally, Section 5.4 summarizes the chapter.

5.2 The CrossFlux Architecture

A new approach to fairness in P2P media streaming has been elaborated and implemented in a media streaming system, named CrossFlux. Further, properties of performance, robustness, scalability, and self-adaptability have been incorporated from the ground up in the design of the architecture. These properties are not only considered during the construction phase of the overlay, but are also dynamically adjusted while content is being streamed.

5.2.1 Design Guidelines

The most important guidelines that have driven the design of CrossFlux are briefly discussed below.

Fairness. As P2P systems live from cooperation, it is important to address the problem of selfish peers that do not contribute their bandwidth to the system. So-called freeriders [1] have to be penalized to force them to serve other nodes. Without any central instance that controls data flows in the system, the peers must have the ability to penalize peers that are unfair. Ideally, each peer should only get as much as it contributes (direct tit-for-tat), but this objective is not compatible with the uniform bandwidth requirement of media streaming. Yet, a peer that contributes much upload bandwidth should get more advantages than a node that does not contribute at all. In CrossFlux a node that provides much upload bandwidth to the system should get more backup links than a node that does not upload anything.

Further, a contributing node should be moved closer to the source, which would reduce the latency between the data provider and the node.

Decentralization. Architectures that are fully *decentralized* have the greatest potential for scalability and reliability, but they have to face additional complexity in their topology management protocols. Missing global knowledge may lead to slightly sub-optimal performance (as has been seen for instance in [51]) but this is a small price to pay for having a *scalable* system. Further, as each peer has the same role, peer failures are not as critical as the failure of a centralized component or a super peer in a hybrid model. In environments with high churn, where many nodes join and leave the system, this property is essential. In CrossFlux the only peer with a special role is the streaming source. If the source fails, no more content is produced and distributed and thus the system stops.

Robustness. The robustness of P2P networks can be increased in two ways: by using either data redundancy or path redundancy. The most widely used techniques for data redundancy are *forward error correction* (FEC), *layered coding* and *multiple description coding* (MDC). Redundancy in data alone does not help if a peer has only one neighbor serving the data. A better strategy is to have multiple neighbors that serve different parts of the stream in parallel such that, if a subset of the neighbors fail, the remaining peers can still serve the data needed to play the stream. Most existing P2P media streaming systems provide such support for path diversity, e.g., using redundant distribution trees or mesh-based topologies. CrossFlux must be able to combine both data and path redundancy for increased reliability.

Adaptiveness. The service capacity of the system consists of the aggregate upload bandwidth of all participating nodes. As this bandwidth is a scarce resource, its usage must be optimized. In the optimal case, each peer can obtain a service capacity equal to the aggregate bandwidth divided by the number of nodes. In [48], different architectures that try to achieve optimal usage of the upload bandwidth have been analyzed. In CrossFlux the topology must be constructed in such a way that each peer has enough parents to receive the full stream and as many children as allowed by its upload bandwidth. The use of the *HeapTop* algorithm [51] for local optimizations allows CrossFlux to automatically adapt to bandwidth fluctuations.

5.2.2 Distribution Overlay

First, properties of the distribution overlay in CrossFlux are described, before it is shown how new peers join the system.

The architecture is presented for a single streaming source with only one live media stream. The overlay can be trivially extended to support multiple sources and streams (one distribution overlay is constructed per stream and per source).

In CrossFlux, content is considered as a sequence of chunks that are separated into m groups (stripes) and sent through m distinct spanning trees. The separation of the original file into stripes is performed by the source, which can optionally generate additional stripes for error correction. In its simplest form, the source can construct a "universal backup stripe" as an *xor* of all stripes. Subsequently, the loss of any stripe can be compensated by *xor*-ing all stripes but the missing one with this universal backup stripe. Of course, more sophisticated network coding techniques can be used. Error correction is orthogonal to the content distribution problem and the decision to encode redundant information in the stripes is left to the source: with that respect, the CrossFlux architecture is content agnostic. Obviously, source-driven data redundancy can be combined with CrossFlux' path redundancy to further increase end-to-end reliability.

Instead of using a single distribution tree, CrossFlux uses multiple trees (one per stripe) to cope with the inherent unreliability of peers. There are two types of links that can be distinguished: (1) primary links are used as active connections to send the content across the overlay; and (2) secondary or backup links are used to quickly route around the failure of a primary link. When a node fails, its children only need to switch to a backup link that can provide the missing stripe while the primary link is being repaired. This strategy prevents searching backup peers in the case of a failure and allows minimizing the recovery time.

Links can be used in both directions: a primary link responsible for serving a stripe s_i from peer p to peer q is used as a backup link (secondary link) for other stripes $s_j \neq s_i$ in the opposite direction from q to p in the case of a failure of a peer on the path from the source to peer p for stripe s_i.

Constructing the communication graph in this way provides incentives for contributing to content distribution, as peers get as many backup links as they are serving other peers. A selfish peer that does not serve any other peer has no backup link and is more adversely affected by a failure. Cooperation upon failure is ensured using a simple tit-for-tat mechanism: if backup peer q refuses to serve stripe s_j to p, then p stops serving s_i to q. Peers that do not have sufficient bandwidth to obtain enough secondary connections and guarantee a certain level of robustness can create bi-directional backup links with each others (again, reciprocity is the incentive).

Not all primary links are also valid backup links. A backup link is only *valid* for a given stripe s_i if it allows a peer p to tolerate the failure of its parent in the distribution tree of s_i, i.e., the peer that serves s_i to p. For being a valid backup, a link must satisfy the following property:

Path Diversity Property A link from p to q for stripe s_i is a valid backup link from q to p for another stripe $s_j \neq s_i$ iff p's parent for s_j is not an ancestor of q in the distribution tree of s_j.

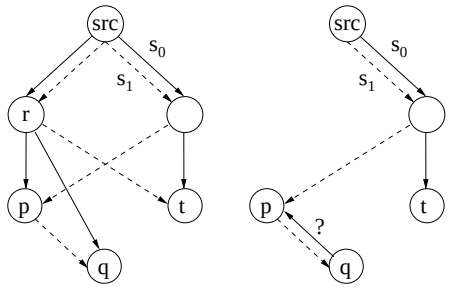


Figure 5.1: Example in which the path diversity property for the link from p to q is not satisfied; before (left) and after failure of r (right).

In Figure 5.1, an example of a forest constructed on 6 nodes and 2 stripes is shown. The solid arrows show the distribution tree for stripe s_0, the dashed ones for s_1. The path diversity property is not satisfied for the link from peer p to q because a failure of p's parent for s_0 (r) also affects q and prevents it from serving s_0 to p in backup mode.

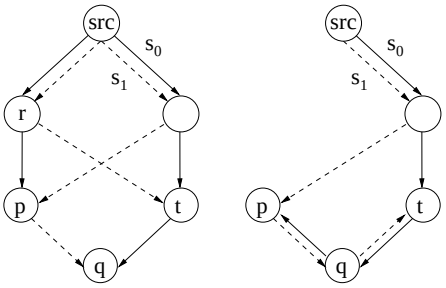


Figure 5.2: Example of content distribution in normal mode (left) and backup mode after failure of node r (right).

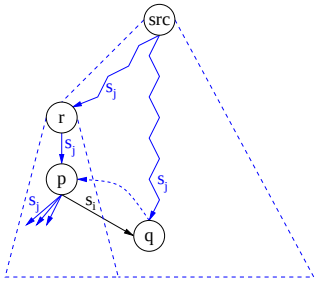


Figure 5.3: Illustration of the path diversity property.

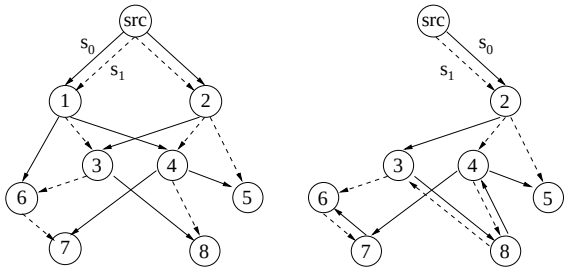


Figure 5.4: Example of content distribution in normal mode (left) and backup mode after failure of node 1 (right).

In Figure 5.2 the link from p to q satisfies the path diversity property for stripe s_0 because the path from the source to q does not traverse p's parent for s_0 (r). In other words, the failure of r will not prevent q from receiving s_0 because q is not part of the distribution subtree for s_0 rooted at r. Thus q can serve s_0 to p in backup mode. The affected nodes p and t use one of their primary links to receive the missing stripe from q.

Consider Figure 5.3. Peer p serves stripe s_i to q. In the reverse direction, the link from q to p is a valid backup for stripe s_j because the path from the source to q does not traverse p's parent for s_j (r). In other words, the failure of r will not prevent q from receiving s_j because q is not part of the distribution subtree for s_j rooted at r.

Figure 5.4 shows another example of a forest constructed on 9 nodes and 2 stripes for both normal mode and backup mode after failure of node 1. The

solid arrows show the distribution tree for stripe s_0, the dashed ones for s_1. In backup mode the affected nodes 3, 4 and 6 use one of their primary links for stripe s_i as backup for stripe $s_j \neq s_i$.

The path diversity property implies that the source needs to serve each stripe to at least two distinct peers, otherwise the grandchildren of the source would not have valid backup links. This requirement is not a major issue because a source typically has enough bandwidth to serve the whole streaming content multiple times in parallel.

The algorithms for joining the system and optimizing the distribution trees guarantee that the path diversity property is always met on a link for at least some of the stripes. This is achieved by a mandatory validation of the property before any link can be established between two peers.

To increase the degree of distribution trees and thus to avoid them to degenerate to linear chains (which would suffer from high latency) the same stripe should be transmitted along multiple connections. Thus a peer should actively transmit a small number of stripes on its outgoing primary connections. On the other hand, transmitting only one stripe along all outgoing primary connections, as in SplitStream [8], implies that no incoming secondary link can be used as a backup for that stripe (lower reliability). Therefore, a good compromise is for a peer to forward at least two stripes and at most $u_p/2$ stripes to its direct neighbors, where u_p is the upload capacity of peer p in terms of stripes. Further if a peer accepts a child for stripe s_i it should reserve at least one upload slot for the same stripe and a different child.

If a peer p has already a high number of backup links for all stripes, it can still benefit from accepting more children even though its robustness is already very high. The more peers p serves the higher is the probability that a high-bandwidth node is in its children set (i.e., backup set). In the case of a failure, p can select the best backup node among all children it serves.

5.2.3 Joining the System

It is assumed that each peer can estimate initially its upstream bandwidth. This value can be set by the user (e.g., by specifying a cap on the bandwidth that the application is allowed to use) or discovered at runtime by

the application (e.g., by sending probe packets or using passive measurement techniques). Peers also keep track of the list of nodes on the path from the source. As the structure of the distribution trees may change due to failures or reorganizations, in each message the sequence of traversed nodes is embedded so that path information can be updated.

The following heuristics are used to find a "good" location where to connect a new peer to: (1) Distribution trees should be balanced, i.e., the paths from the source to the leafs of the trees should have approximately the same length. (2) Peers should preferably join healthy branches with much spare capacity rather than branches with limited growth potential. (3) A newcomer should try to connect to interior nodes first, if they have sufficient spare capacity, in order to maximize their utility and limit the depth of the trees.

The first criteria implies some fairness or randomness in the selection of the subtrees where to connect new nodes. The second criteria avoids directing newcomers toward branches that have limited service capacity, e.g., because they only contain peers with low bandwidth. The third criteria favors nodes high in the tree, given that they still have sufficient service capacity.

For each stripe s_i, each peer p maintains a "healthiness" value $h_p(s_i)$, which it periodically transmits to its parents in the distribution tree of s_i. Healthiness represents above mentioned heuristics and is computed as the average of p's healthiness and the mean of its children's healthiness in the distribution tree of s_i. For a leave, the healthiness is equal to the number of new children it can accept for s_i, i.e., its spare capacity. The healthiness of the nodes evolves over time as a result of peers joining and leaving, as well as reorganization of the distribution trees.

Formally:

$$h_p(s_i) = \begin{cases} \frac{1}{2}\left(f_p(s_i) + \frac{1}{|C_p(s_i)|}\sum_{c \in C_p(s_i)} h_c(s_i)\right) & : \quad |C_p(s_i)| > 0 \\ f_p(s_i) & : \quad |C_p(s_i)| = 0 \end{cases} \tag{5.1}$$

where $f_p(s_i)$ is the number of new children that p can accept for stripe s_i and $C_p(s_i)$ is the set of children of p for stripe s_i. One can note that the

free capacity of p has more weight than that of its children. This allows us to favor new connections to nodes high in the trees. Note that a peer may return 0 for $f_p(s_i)$ if it still has enough upstream capacity but is already interior node of several distribution trees other than s_i.

A new peer starts to join the system at the source. It can obtain information about the source, for instance, from a Web page (e.g., IP address, stream rate). To join a distribution tree, a peer q issues a join request JRQ for each stripe to the source. Join requests traverse the distribution trees using biased random walks (which have only a fraction of the overhead of a broadcast). A join request JRQ for stripe s_i is propagated along the distribution tree of s_i as follows. If the current node p can accept q as a child for s_i, it sends a message (CAN) to q together with its healthiness h_p and the path from the root to p to inform it that p is able to (*can*) accept q as a new child. Then, if p has children in s_i, it forwards the join request to a child chosen at random according to a biased distribution in which the probability of choosing a child is proportional to its healthiness. These messages traverse the associated distribution trees until "enough" potential parents for q are found (trade-off between waiting time and quality of parent position).

During such a random walk, joining node q typically receives several CAN replies. It then selects among the replies the node p closest to the root and, upon tie, the node with the highest healthiness, under the condition that the path diversity property is satisfied for the connection from p to q. Note that the property can be verified using the path information embedded in the CAN message. The join procedure finishes when the new node starts receiving chunks from its parent. If q receives no valid replies, it issues another join request that will likely follow a different path in the distribution tree. Note that q can also request multiple random walks to be conducted in parallel to quickly gather more candidates.

The behavior of the source differs from other peers in that it always tries to have the same number of children for each stripe. The source accepts a new child for stripe s_i if it has sufficient bandwidth *and* no other stripe has less children than are currently registered for s_i.

The source serves directly the first few peers in parallel, during the bootstrap phase. Thereafter, new peers will connect deeper in the distribution

Algorithm 2 Reception of JRQ(s_i, j) at peer p for stripe s_i and new peer j

```
if f_p(s_i) ≥ 1 and j ∉ C_p then
  send CAN(p, h_p(s_i), source → p) to j
end if
if C_p ≠ ∅ then
  c ← biased random node from C_p
  send JRQ(s_i, j) to c
end if
```

trees. This unfairness between early and late joiners is compensated over time by the *HeapTop* algorithm that continuously optimizes the distribution tree and changes the position of the nodes.

A new peer always connects to the distribution trees as a leaf. Therefore, it has no children and no backup links initially. This conscious design decision is motivated by the fact that, in typical P2P systems, many peers remain connected a very short amount of time: the longer a peer has been online, the higher the probability that it remains connected [21], [46]. Therefore, departures among the volatile population of newcomers will have limited impact. As peers remain in the system, they will accept children and consequently acquire backup links. They may also move upward the tree if they have good service capacity. This approach acts as an incentive for peers to remain connected for long periods of time and contribute well to the system.

Note that the heuristics used to meet these criteria will *not* produce optimal distribution trees. The dynamic reorganization of the nodes in the trees has been precisely designed to improve the efficiency of the trees after their construction.

5.2.4 Content Distribution

The chunks of each stripe are forward along the associated distribution trees in a straightforward manner: each inner node of a distribution tree forwards incoming chunks to all of its children in that tree. We assume that the links between nodes are reliable (we use TCP in our implementation).

Peers buffer the chunks for some time, so that they can transmit them to their neighbors over secondary links in case of a failure. To dispose of

buffered chunks, each peer regularly sends a notification to its backup neighbors indicating the last chunk it has received for each relevant stripe. This mechanism allows secondary sources to dispose of the chunks that they buffer for retransmission purposes. If the buffers of a peer are full, it may delete the chunks in its retransmission buffers even if the peers downstream secondary links have not yet acknowledged their reception.

5.2.5 Departures and Failures

Upon a node failure, its children in each stripe must find a new parent. This operation must be very fast to guarantee smooth playback of the media stream. CrossFlux relies on the backup links for quick failover: affected children ask their backup sources to send missing chunks. Failures can be detected by children when a network connection is closed or times out.

By ensuring that each contributing node has at least one valid secondary link for each stripe, the system can be quickly reconfigured after a failure while providing good load balancing: the children of a failed node will request the missing stripes from distinct peers with high probability. Obviously, backup sources must have spare bandwidth to send the missing stripes, even with degraded performance, until the primary link is restored. Typically, peers keep some free bandwidth for dealing with failures, i.e., they underestimate their spare capacity when computing their healthiness. The amount of spare bandwidth can be reduced over time as the probability of a parent failure decreases [21], [46]. The number of parents for a node p is equal to the number of stripes m and corresponds to the maximal number of nodes p has to serve as a backup. This also limits the additional bandwidth used for serving as a backup, in the case of a failure.

After promoting a secondary link to primary, the peers affected by the failure execute the join protocol to find a new parent and revert the status of the secondary link.

5.2.6 Overlay Optimization

When optimizing effective throughput, one needs to take into account the dynamism of the underlying network and the bandwidth heterogeneity. To that

end, *HeapTop* [51] is used in CrossFlux to dynamically move fast nodes upward the trees toward the root. For scalability reasons, reorganization of the tree should affect as few nodes as possible. Exchanging the position of a node with its parent is a local operation that can be easily implemented because both nodes are directly connected with each other and they essentially have to exchange their respective neighbors.

Each node continuously executes the *HeapTop* algorithm. Peer p periodically compares its bandwidth capacity with that of its parent. If p's bandwidth is notably higher than its parent's bandwidth, then they switch positions, i.e., they exchange their parents and children, under the condition that the diversity property is still satisfied. The algorithm preserves the structure of the initial tree (even if it is not balanced), but the position of the nodes evolves over time. For avoiding pairwise exchanges resulting from short bandwidth fluctuations, the estimations are based on a weighted moving average.

Given the special role of the source node, it appears clearly that the peers cannot move from one 1^{st}-level subtree to another 1^{st}-level subtree. As such, the resulting distribution tree may be slightly sub-optimal but performing further optimizations would necessitate non-local operations and higher complexity.

If there is no bandwidth fluctuation, the tree will quickly reach a stable configuration. In the worst case, a node located at depth $d \geq 1$ (the root is at depth 0) can initiate $d - 1$ exchanges. The actual number of exchanges depends on both the initial configuration of the tree and the order in which exchanges are performed.

Several important considerations must be taken into account when using *HeapTop* to optimize the distribution trees in CrossFlux. First, the *HeapTop* algorithm is run independently in the distribution trees of each stripe. Second, leaf nodes are not exchanged with inner nodes if the former is already inner node of several other trees (typically 2, as previously discussed). Finally, before performing any pairwise exchange, it is verified that the path diversity property will be preserved in the new configuration.

5.3 Evaluation

A prototype of CrossFlux has been developed in Java (see Appendix A). The implementation has been designed in such a way that it can be deployed in simulated settings, in controlled environments such as clusters, and in large-scale networks.

Most of the experiments were carried out in a simulator, which allowed us to observe the behavior of the system with large client populations without the need of a critical mass of users. Several experiments have been performed in distributed settings using the Modelnet [55] network simulator and finally the evaluation has been completed using PlanetLab. Some results of this evaluation are discussed in this section.

5.3.1 Simulations

Experimental Setup. Three main classes of peers have been simulated (Fast F: 1024 Kbit/s, Medium M: 512 Kbit/s, Slow S: 128 Kbit/s), chosen to match the observations that have been made of real-world populations in an earlier study of the BitTorrent protocol [25]. Simulated population sizes range from 500 to $4,000$ peers. As the upload bandwidth is usually the limiting factor, download capacities are not explicitly taken into account (peers of classes M and S typically have asymmetric bandwidth). Each simulated peer's class was chosen randomly according to the 6 distributions $D_1, \ldots, D_6$ shown in Table 5.1.

	Class F	Class M	Class S
D_1	90%	5%	5%
D_2	60%	30%	10%
D_3	50%	25%	25%
D_4	30%	60%	10%
D_5	25%	25%	50%
D_6	5%	90%	5%

Table 5.1: Distributions of peer classes for evaluation.

Dynamic Adaptation of the Overlay. First the dynamic optimization of the overlay using the *HeapTop* algorithm has been studied. The main criterion considered is the average upload bandwidth capacity using the tree adapted by *HeapTop*, as compared with that of an initial randomly generated tree. For every considered distribution, binary trees were constructed by iteratively adding each node at a valid position, chosen by traversing the tree from the root until a leaf or a node with a single child is encountered. This join procedure is much simpler than in CrossFlux but makes it easier to observe the effect of *HeapTop* in isolation. Both balanced and unbalanced trees have been experimented with. As the differences between both settings were negligible, only results for balanced trees are shown and it is noted that they are also valid for unbalanced trees.

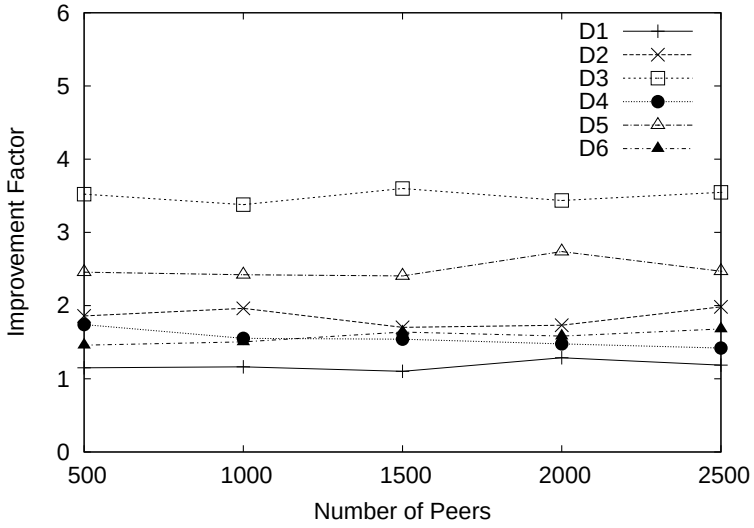


Figure 5.5: Average improvement factor with two stripes for different population sizes and various class distributions.

The improvement factor of *HeapTop* has been evaluated with different population sizes for each node distribution. *HeapTop* has been simulated by running it separately on two stripes, as implemented in CrossFlux. Figure 5.5 shows the improvement factor f, defined as the ratio of the average bandwidth $\overline{B}_{HT}$ of the tree generated by *HeapTop* to the average bandwidth $\overline{B}_R$ of the random initial tree: $f = \overline{B}_{HT}/\overline{B}_R$.

One can observe that the gain is significant (up to 350%). The best improvement factor observed during simulations was 750%, which gives a measure of the potential benefits of dynamic overlay optimizations in CrossFlux.

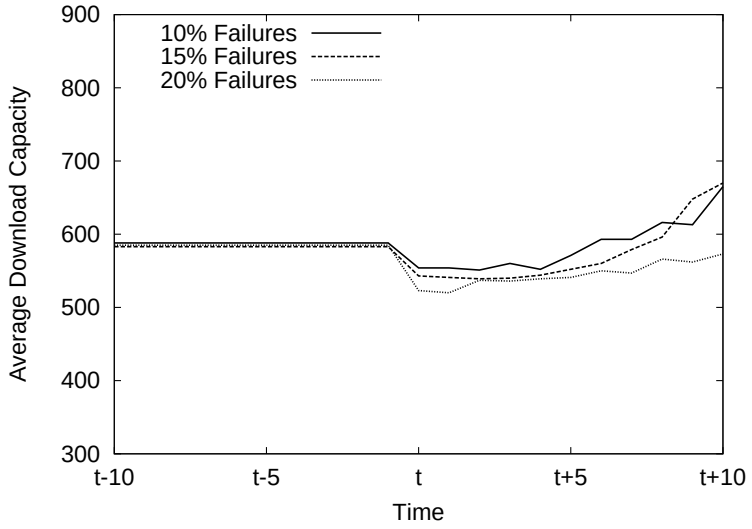


Figure 5.6: Impact of failures on the average download capacity for a population of $2,000$ nodes with distribution $D4$.

Failure Recovery. The ability of CrossFlux to recover from node failures has been tested on a population of $2,000$ nodes with upload bandwidths matching the distribution $D4$ which best reflects real-world scenarios. In a first phase of the simulation, all the nodes were added to the system and the simulation was run for some time without activating *HeapTop* to stabilize the system. Then simultaneously a fraction of nodes chosen randomly was shut down, and *HeapTop* was activated to improve recovery effectiveness. The failures were detected without delay for not depending on timeout parameters.

Figure 5.6 shows the average download capacity as a function of the simulation time (discrete steps). The failure occurred at time t and recovery directly began by switching to backup links. It can be observed that the impact of the failures on the download capacity is very moderate. The subse-

quent improvement, with the download capacity reaching above its previous value, are due to *HeapTop*.

Download Performance. CrossFlux's scalability and performance has been studied by simulating a content distribution network with a single source, 4 stripes, and a streaming rate set to 320 Kbit/s. The simulation was started with only the source, and then iteratively nodes were added using the join algorithm. The upload capacity of each node was chosen randomly according to distributions and classes introduced in Table 5.1. The maximal number of children of a node was determined based on its upload capacity. When all the nodes completed the join procedure for all stripes, the available download capacity of each node (higher than the actual streaming rate) was computed. The experiments were done (1) with *HeapTop* disabled (Figures 5.7 and 5.8) and (2) with *HeapTop* activated (Figures 5.9 and 5.10).

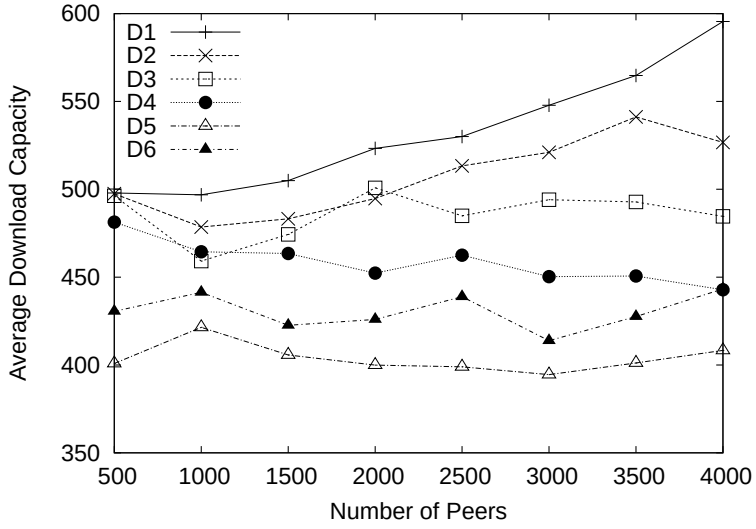


Figure 5.7: Average download capacity as a function of the number of peers (without *HeapTop*).

As can be seen in Figure 5.7, the system scales well with the node population for all distributions: the download capacity does not degrade when adding new peers and is consistently above 320 Kbit/s even for distributions with a high fraction of slow and medium-speed nodes. It can even be

observed that the average download capacity increases with the node population for distributions with many fast nodes (e.g., distribution $D1$). This can be explained by the fact that slow nodes can more significantly impact the performance of small networks, because the restrictions imposed by the path diversity property sometimes enforce placing slow nodes in the interior of the distribution trees. This problem becomes less relevant in large networks where more positions are available for placing nodes.

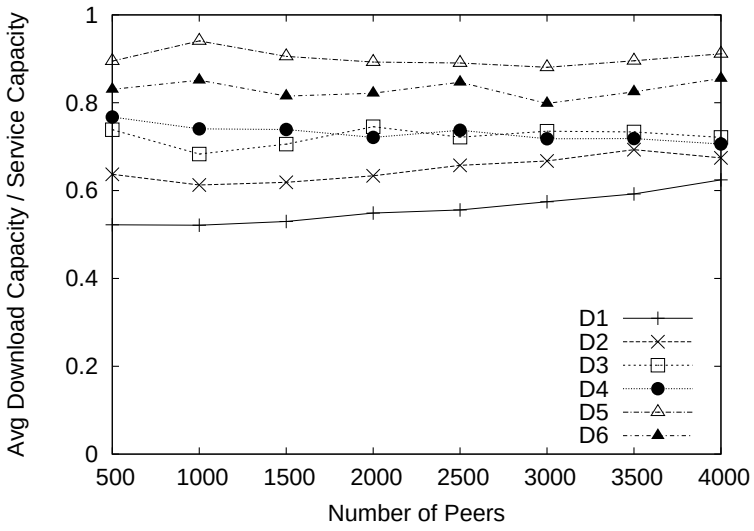


Figure 5.8: Average download capacity compared to the optimal service capacity (without *HeapTop*).

Figure 5.8 compares the effective download capacity obtained with CrossFlux to the optimal service capacity, i.e., the ratio of the aggregate upload bandwidth of the network to the number of nodes. As one can see, CrossFlux utilizes between 60% and 95% of the available bandwidth depending on the distribution. In Figures 5.9 and 5.10 it can be seen that *HeapTop* significantly increases the average download capacity for all distributions, as expected from the simulation results of *HeapTop*.

Path Lengths. In addition to being bandwidth-efficient, a good distribution tree should also balance well the load on all the nodes. This is generally the case for balanced trees, as all nodes have approximately the same number

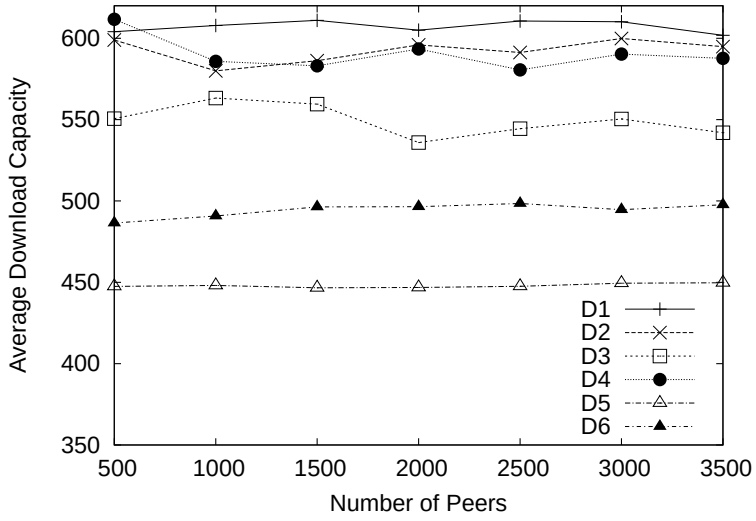


Figure 5.9: Average download capacity as a function of the number of peers (with *HeapTop*).

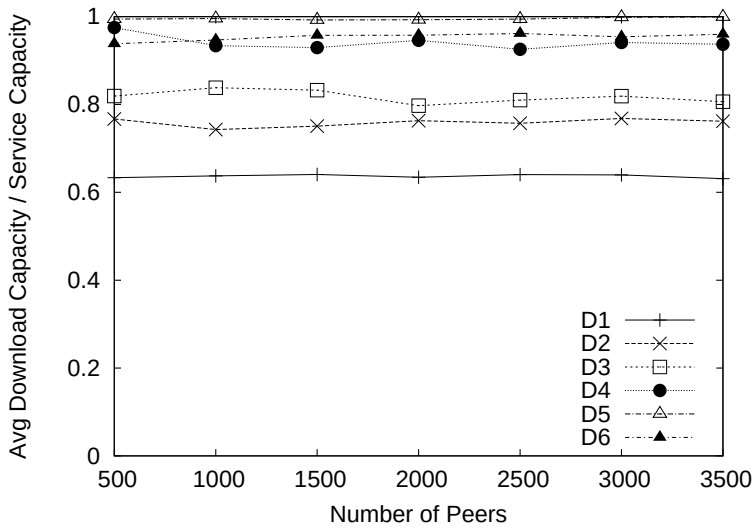


Figure 5.10: Average download capacity compared to the optimal service capacity (with *HeapTop*).

of children and all leaves the same depth. Obviously, the heterogeneity of the node capacities does not allow maintaining balanced trees. To evaluate the

quality of the distribution trees obtained with CrossFlux, the lengths of the paths from the root have been evaluated for populations of $2,000$ nodes. Distributions $D1$ and $D6$ were used, as they are the two most homogeneous distributions and are expected to produce reasonably balanced trees. The node degree of other distributions is too uneven to draw meaningful conclusions. The maximum path length over all nodes and all stripes has been computed, as well as the maximum over all nodes of the average path length, computed as the mean over all stripes (denoted by max-average). For comparison, the maximal path length of a balanced tree with a constant node degree d and n nodes is in $O(\log_d n)$.

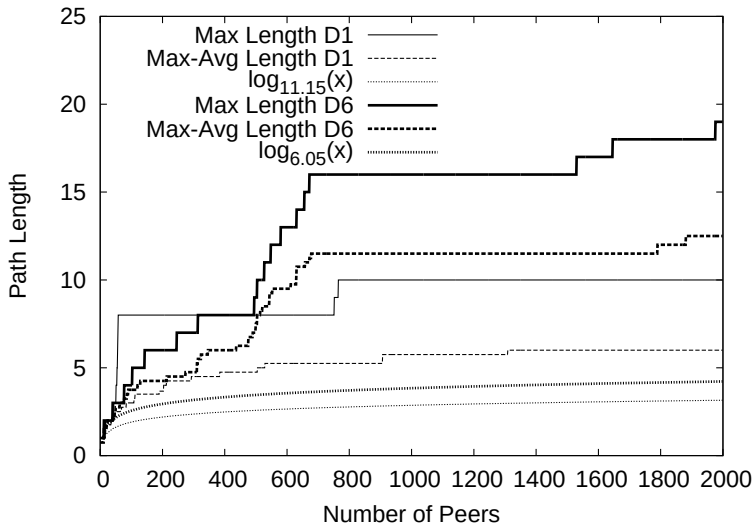


Figure 5.11: Maximal and max-average path lengths compared to the theoretical maximal path length.

Figure 5.11 shows the maximal and max-average path length for both distributions, as well as the asymptotic depth of a balanced tree with a degree equal to the average number of children of the interior nodes of the distribution trees produced by CrossFlux (11.15 and 6.05). It can be observed that paths remain reasonably short, which indicates good structure balancing. The growth of the path length follow a logarithmic curve, within a factor of $2-4$ from the theoretical optimum. The reason for this difference is that the degree of CrossFlux trees varies significantly from node to node

depending on their capacity, which makes the comparison unfair.

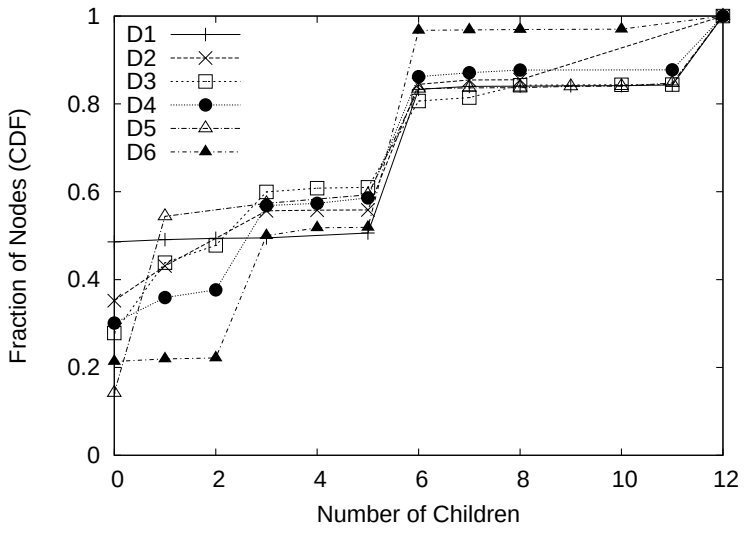


Figure 5.12: Cumulative distribution function of the number of served children.

To further evaluate how effective the join procedure is at producing a balanced tree, the number of stripes served by each node has been compared. Figure 5.12 shows the cumulative distribution function of the number of children that are served by a node for the different distributions of Table 5.1. As one can see, the load in CrossFlux is quite balanced. Independently of the distribution, only a small portion of the nodes (at most 20%) serve more than 6 children, although in the setup the fast nodes have the capacity to serve 12 children. This indicates that the nodes with less capacity also have to contribute.

Node Stress. Node stress is defined as the number of management packets a node receives in a time unit. The main overhead is due to (1) the join procedure during which join requests are issued and forwarded and (2) the reporting mechanism used by children to send their healthiness value to their parents. In CrossFlux, random walks are used instead of broadcasts during the join to reduce the number of messages. Figure 5.13 shows the average node stress for distribution $D4$ for different population sizes. At simulation

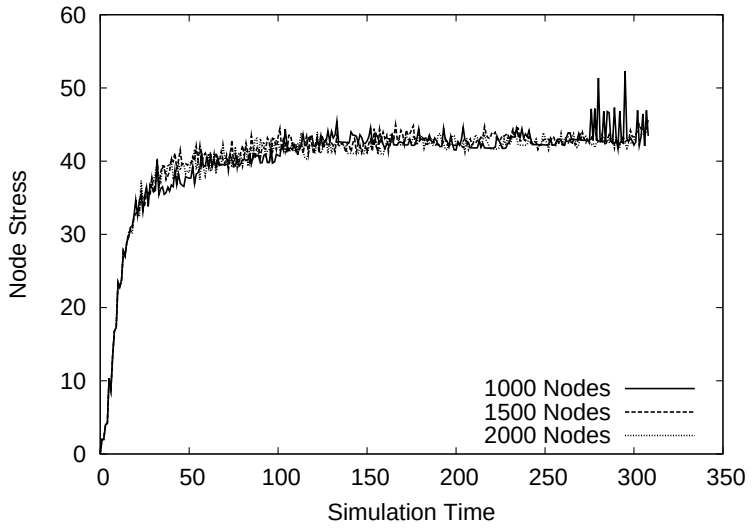


Figure 5.13: Node stress for different population sizes and distribution $D4$.

time 280 a fraction (10% of the population size) failed. As one can see, the average number of management packets increases during the join procedure and quickly stabilizes afterwards to the amount of healthiness reporting messages. During failure recovery the average node stress contains peaks but remains within 25% of its previous value. The average node stress is the same for all three population sizes, thus highlighting the low overhead of the random walks.

5.3.2 Modelnet Simulations

Experimental Setup. Modelnet is a network simulator that emulates a virtual network on top of a set of machines (typically a cluster). The software to be evaluated is deployed on multiple virtual hosts residing on each machine. The traffic generated by these virtual hosts is routed through the simulator, which mimics the behavior of the modeled links (delay, throughput, loss) and forwards it to the destination.

Modelnet [55] was used to evaluate CrossFlux on a small testbed. In Modelnet, each end-to-end link in the topology can be assigned different values for bandwidth, latency, and loss rate. For this purpose, the Inet

generator [28] was used to generate a random transit-stub topology of 4,000 nodes with 50 CrossFlux clients spread across 19 stubs. The bandwidth of each link was chosen randomly in the range from 512 Kbit/s to 1024 Kbit/s. The number of stripes that a node could serve was determined according to its connection speed. A single streaming source has been set up to serve an endless stream, which was split into chunks of 40 Kbit and distributed using 8 stripes. The streaming rate was fixed to 320 Kbit/s, thus each peer should receive at least 8 chunks per second.

Dynamic Adaptation of the Overlay. As the structure of the distribution trees obtained with CrossFlux depends on many parameters that cannot be easily controlled (including random factors), we have studied the behavior of *HeapTop* using simulations that faithfully reproduce the operations of the algorithm and evaluate its efficiency.

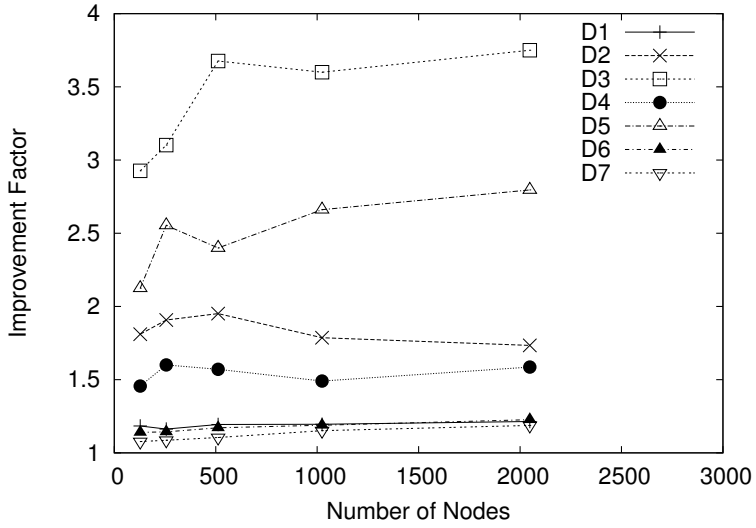


Figure 5.14: Average improvement factor with two parallel trees for different population sizes and various class distributions.

We simulated *HeapTop* by running it on the inner nodes of each stripe, as happens in CrossFlux. In other words, a leaf in the initial tree is never promoted to inner node. Figure 5.14 shows the improvement factor for different population sizes and various class distributions.

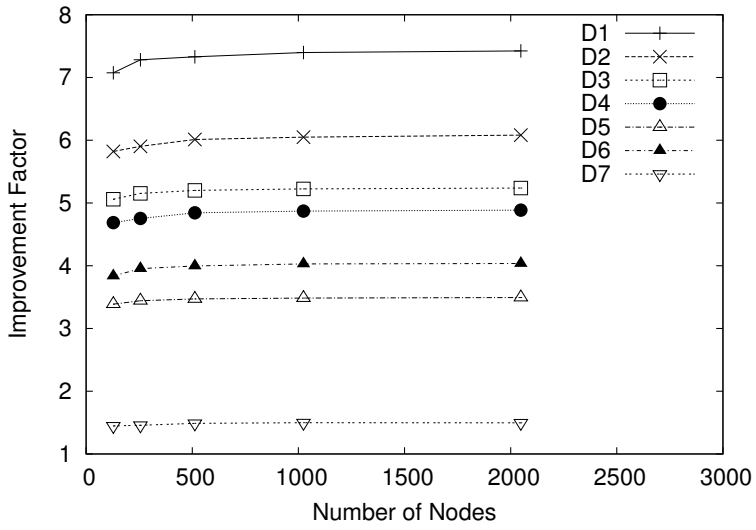


Figure 5.15: Best case improvement factor for two parallel trees for different population sizes and various class distributions.

One can observe that the gain is significant (up to almost 400%). Figure 5.15 shows the best improvement factor observed during the simulations (up to 750%) and gives a measure of the potential benefits of *HeapTop* for CrossFlux.

In the implementation of CrossFlux, a buffer was used where received chunks are stored until they have been read for playback. The average size of this buffer gives an estimate of the download rate of the peer: if the buffer becomes empty, then the peer does not receive the content at a sufficient rate.

Figure 5.16 shows the cumulative distribution function of the fraction of nodes with a given percentage of free receive buffer space. As one can see, there is much less free buffer when *HeapTop* is used than without. With *HeapTop* activated, 90% of the nodes can fill their buffer up to 80% of the available space. When *HeapTop* is disabled, only 50% can fill their buffer up to 80%. This indicates that fast nodes are effectively moved toward the root and chunks are flowing faster from the root to the leaves.

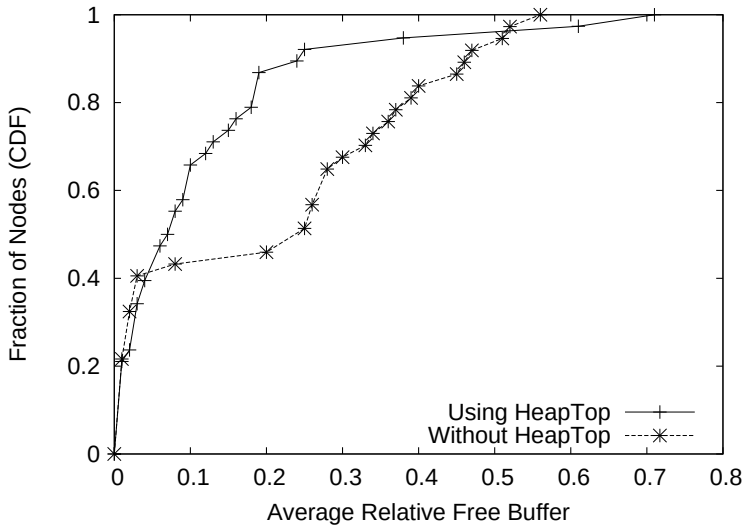


Figure 5.16: Cumulative distribution function of the free receive buffer space.

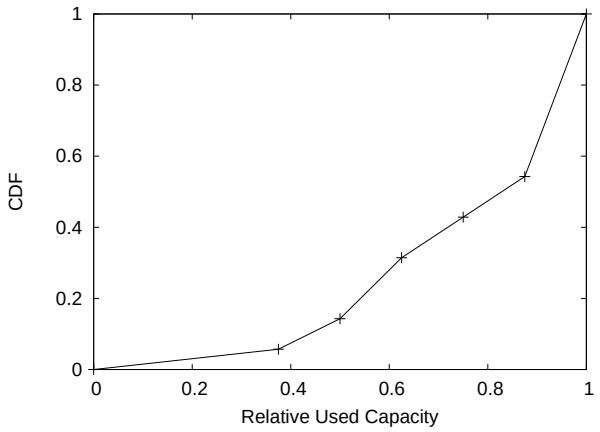


Figure 5.17: Cumulative Distribution Function of the Used Capacity.

5.3.3 Load Balancing

To evaluate the load balancing and fairness properties of the join procedure, we compared the number of stripes served by each node. To that end, we added all 50 nodes sequentially, with one new host joining every 5 seconds. Figure 5.17 shows the cumulative distribution function of the average used capacities after all nodes have joined. As one can see, the trees in CROSS-

FLUX are well balanced. About 50% of all nodes have no spare capacity and very few are almost idle, indicating that the capacity of the system is well used.

Failure Recovery. A key requirement of P2P media streaming is fast recovery from failures. CROSSFLUX deals with that problem using backup links. In this experiment, we compared the recovery time when switching over to backup links and when rejoining from the source. We terminated a node and observed the traffic at one of its children. The failure of the node was immediately discovered by the children upon socket disconnection, so that there is no noticeable delay in fault detection.

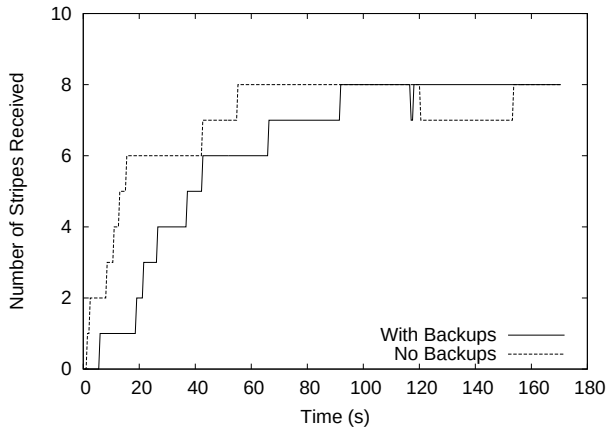


Figure 5.18: Comparison of Recovery Time with Backup Links and without.

Figure 5.18 shows the number of stripes received as a function of time for a node that must recover from its parent's failure. We observe that, when using backup links, the interruption is almost unnoticeable while it takes approximately 30 seconds to rejoin from the source. Note that, in both cases, the missing chunks were buffered and delivered after recovery.

5.3.4 PlanetLab

CROSSFLUX was deployed on 136 randomly chosen nodes of the PlanetLab test-bed [39]. The source was started on one of the nodes and the others

subsequently executed the join procedure. In PlanetLab the resources of each machine are shared between many concurrent users and nodes may fail due to temporary overload or maintenance at random. In Figure 5.19 the download rate over time for a sample node is shown. One can observe that the node does indeed receive content at a constant speed. The node was affected by a parent failure at time 320. There are small spikes during the recovery procedure, but the overall reception rate has not been affected by the failure.

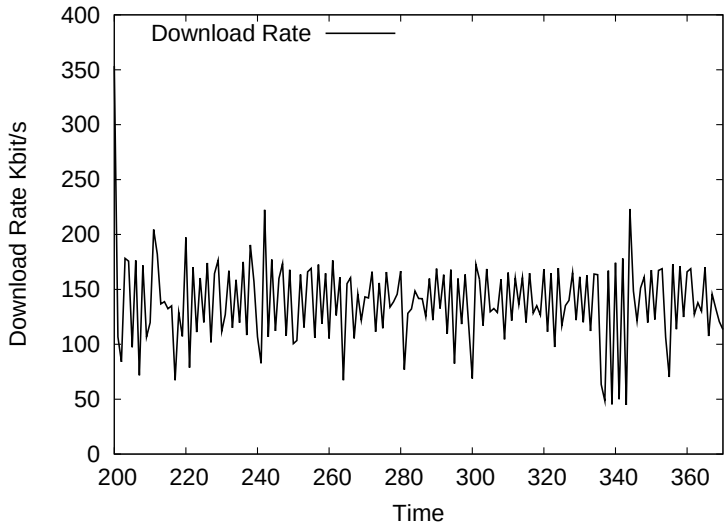


Figure 5.19: Download rate for a PlanetLab node with a parent failure at time 320.

5.4 Summary

In this chapter, the design of CrossFlux, a peer-to-peer architecture specifically designed to satisfy the stringent requirements of media streaming, has been presented. For distributing a stream from a single source to a large population of clients, the content is split into multiple stripes that are distributed over different trees. Instead of a classical tit-for-tat strategy based on the upstream and downstream throughputs, altruistic peers that serve others are rewarded by additional robustness, as each connection can be used as

a backup link in the reverse direction. This structural tit-for-tat is more adapted to media streaming, because streaming content must be delivered at a uniform rate.

CrossFlux dynamically modifies the structure of the trees to adapt to bandwidth fluctuations and to optimize the efficiency of content distribution. Stable nodes with high bandwidth capacities are moved upward the trees toward the root where they are most useful. The evaluation shows that CrossFlux produces balanced and efficient distribution trees and can quickly recover from failures.

The tree structure used in CrossFlux provides little flexibility for peers to choose a parent peer at tree-construction time. After having joined the tree, a peer can adapt its position according to its upload bandwidth. The adaption is only based on comparisons with its direct neighbors. A more flexible structure based on mesh-based P2P architectures is analyzed in the next chapter. It will be shown that the analytical results and the algorithms obtained for trees can also largely be applied to mesh-based architectures.

Chapter 6

Tree-based Analysis of Mesh Overlays for Peer-to-Peer Streaming

6.1 Introduction

Additional to tree-based architectures like those presented in Chapter 3 mesh-based architectures which are less restrictive on connection construction algorithms exist as well.

The tree-based approach explicitly places peers in a single tree or multiple multicast trees, where they receive the stream from their parent(s) and forward it to their children. In the mesh-based approach, the P2P overlay is unstructured, formed by peers connecting to neighbors, which may be randomly selected. The media stream is typically split into small data blocks that are exchanged between neighboring peers, resulting in their propagation throughout the overlay. The main advantage of mesh overlays compared to tree-based overlays is their much higher robustness to peer churn. In tree-

Parts of this chapter have been published in: Bartosz Biskupski, Marc Schiely, Pascal Felber, René Meier. Tree-based Analysis of Mesh Overlays for Peer-to-Peer Streaming. *Proceedings of the 8th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'08)*, June 2008.

based approaches, a peer can receive data only from its specified parent and when that parent fails or leaves the network, its whole sub-tree loses that data until the tree is reconstructed. In mesh-based streaming systems, data chunks can be obtained from any neighbor that holds it and thus when one neighbor fails, other neighbors may still provide the data. For that reason, many researches focus on mesh overlays for P2P streaming. However, one problem posed by mesh overlays is that they do not rely on any predefined network structure and thereby are more difficult to study than tree-based overlays.

In this chapter, we show that when data chunks are streamed over mesh overlays, tree-based diffusion patterns dynamically emerge in the overlay. These tree-based patterns of diffusion can be studied in the same manner as tree-based overlay structures in Chapter 3 and adaptive algorithms as the one presented in Chapter 4 can be applied.

The contribution of this chapter is that we identify and analyze properties of the emerging tree structures in mesh overlays and, in order to evaluate their performance, we compare them to optimal diffusion trees in both homogeneous and heterogeneous environments. This provides insights into how mesh overlays can be adapted to reduce buffering delay in mesh-based streaming systems to a theoretical minimum. Based on this analysis we developed an algorithm that reduces diffusion tree heights in a mesh overlay and thus, also reduces buffering delay.

The chapter is organized as follows: Section 6.2 shows how diffusion trees emerge in mesh overlays and analyses these diffusion trees. Finally, the adaptation algorithm is presented and evaluated in Section 6.3 before the chapter is summarized in Section 6.4.

6.2 Mesh-based P2P Streaming

The mesh-based approach to data streaming originates from research on gossip and epidemic protocols, where nodes periodically exchange information among each other, which results in the eventual dissemination of all information to all nodes. The BitTorrent [16] file-sharing system popularized this approach for the dissemination of large volumes of data from a transmitter to

all receivers. BitTorrent creates an unstructured overlay mesh to distribute a data file. A file is divided into chunks, which are exchanged by nodes in a pull-based fashion until nodes can reconstruct the original file.

In contrast to file-sharing systems, the transmitter in live P2P streaming protocols does not have access to the entire data as it is generated "live", and thus, it cannot split the whole data into chunks for distribution throughout the network. In order to leverage mesh-based delivery, streaming protocols require a delay between the stream creation time at the transmitter and the receiver playback time. The data stream produced within this delay is split into small chunks and distributed throughout the network similar to the way chunks of an entire file are distributed in mesh-based file-sharing protocols. Nodes maintain sliding windows that reflect this delay and capture which chunks have already been received and which are still missing. The buffers move forward with the speed of the original video transmission rate, which is discovered by all nodes from the video stream. The beginning of the buffer points at the chunk currently being played at the receiving node and the end of the buffer reflects the chunk currently generated at the transmitting node. Chunks that do not arrive in time (outside the sliding window) are lost and cause video playback degradation.

A mesh overlay is created in a random fashion by joining nodes connecting with selected nodes. The selection of neighbors can be based on different strategies, e.g., random or bandwidth-based. Neighboring nodes maintain local knowledge about data chunks they possess by informing each other whenever they receive a new chunk. The missing chunks are requested from neighbors immediately or periodically, following a chunk selection algorithm. Different strategies such as most-recent-chunk-first, rarest-chunk-first or random can be used to schedule the chunk requests.

6.2.1 Mesh Overlay Properties

In previous research on mesh overlay adaptation [4, 5], it was identified that completely random mesh overlays limit the network throughput by underutilizing the available upload bandwidth at peers. Limited network throughput in turn reduces possible video streaming rates and the corresponding video

quality. We showed properties of mesh overlays that, when satisfied, optimize the network throughput. This requires that each peer maintains two sets of neighbors — (1) children, which are the neighbors to which data is uploaded and (2) parents, which are the neighbors from which data is downloaded. The network throughput is optimized in such a directed mesh overlay when:

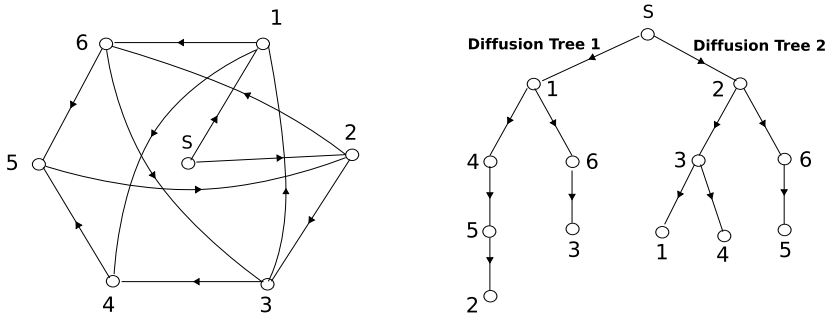
- Each peer has a constant (configurable) number of parents
- Each peer has a number of children proportional to its upload bandwidth

We showed in [4] that a mesh overlay satisfying these two conditions optimizes the upload bandwidth utilization and enables all peers to download at the maximum possible global video streaming rate. We also proposed algorithms for adapting the mesh overlay to satisfy these conditions. In this chapter, we conduct our analysis on directed mesh overlays that satisfy these two conditions and thus we can provide a fair comparison to multiple-tree-based overlays that also optimize the network throughput. This analysis is novel in that we show how diffusion trees emerge in these adapted directed mesh overlays; we analyze properties of diffusion trees and compare them to those of multiple-tree-based overlays; and finally, propose an algorithm that improves these properties.

6.2.2 Tree-based View of Mesh Overlays

Mesh overlays are very dynamic and thus are difficult to analyze. In contrast, trees are well understood and it is easier to derive properties of trees. Meshes can be seen as a structure of multiple trees if we assume that bandwidth of all peers remain constant over time and that the chunk selection algorithm is deterministic. We assume that peers request missing chunks from parents immediately when they are notified of them, following a most-recent-chunk-first strategy, i.e., when a decision is made between two chunks, a chunk with a more recent timestamp is requested. This chunk request strategy is based on an observation that most recent generated chunks are also the rarest in the overlay and thus need to be given priority for distribution.

We assume that the stream rate is set to the maximum rate supported by the overlay such that all peers can receive it, i.e., equal to $\frac{\sum_i^N upload_i}{N-1}$, where N is the total number of peers including the source node (the source uploads, but does not download data). We also assume that the mesh overlay satisfies conditions discussed in Section 6.2.1 and that a peer's upload bandwidth is shared equally by all its connections. Under such assumptions, upload of all peers is saturated and the upload rate of each link is the same, equal to $\frac{\sum_i^N upload_i}{(N-1)K}$, where K is a globally configurable number of parents of each peer. From this follows that each chunk is transferred over a link in time $\frac{s(N-1)K}{\sum_i^N upload_i}$, where s is the size of a chunk. The source node generates a new chunk every $\frac{s(N-1)}{\sum_i^N upload_i}$ time units, so by the time a single chunk is transferred to a child, K new chunks are generated. Since it is desired that the source node sends different chunks to different children (to distribute chunks equally in the overlay), we use a round-robin strategy to push chunks from the source node to its direct children in which the i^{th} child receives chunks with sequence numbers $t_0 + jK + (i \bmod K)$, for some initial t_0 and $j = 0, 1, 2, 3, ...$ Peers, which are not direct children of the source node, request the most recently generated missing chunks, so they always request a missing chunk that travelled the least number of hops (and time). Effectively, K diffusion trees emerge, where each tree propagates every K^{th} chunk. This process of diffusion trees emerging in a mesh overlay, which has properties outlined in Section 6.2.1, is illustrated in Figure 6.1 for $K = 2$.



(a) Directed mesh overlay where each peer has 2 parents

(b) Corresponding diffusion trees

Figure 6.1: Mesh overlay and its two diffusion trees.

6.2.3 Analysis

In this section we show how optimal multiple trees are constructed in both homogeneous and heterogeneous environments and analyze their heights in order to compare them, in the next subsection, to diffusion trees emerging in mesh overlays.

Height of optimal trees in a homogeneous environment. First, we analyze a homogeneous environment, where all peers have the same upload capacity. Optimal K distribution trees can be created by placing each peer as an inner node in exactly one tree and as a leaf node in the other $K-1$ trees. Thus, each peer has K parents, one in each optimal distribution tree. In a homogeneous environment, this means that the out-degree d of each peer is equal to K. Since a peer has children in only one tree, K and d are the number of children of each inner node in each tree. Thus, the height of each of K optimal distribution trees in a homogeneous environment with N nodes is equal to the height $H(d,N)$ of an evenly balanced tree with N nodes and out-degree d, which is calculated using a relation

$$\sum_{i=0}^{H(d,N)} d^i = N$$

based on the fact that there are d^i peers at tree level i. Solving this geometric sequence gives an equation for the height of a balanced homogeneous tree:

$$H(d,N) = log_d\left((d-1)\,N+1\right) - 1 \tag{6.1}$$

Therefore, the height of each of K optimal trees in a homogeneous environment is given by $H(K,N)$. In this chapter we also use an equation for the number of leaf nodes $L(d,N)$ in a balanced homogeneous tree with N nodes and out-degree d, given by

$$L(d,N) = d^{H(d,N)} = \frac{(d-1)N+1}{d} \tag{6.2}$$

Height of optimal trees in a heterogeneous environment. We study the construction of optimal trees in a heterogeneous environment by using

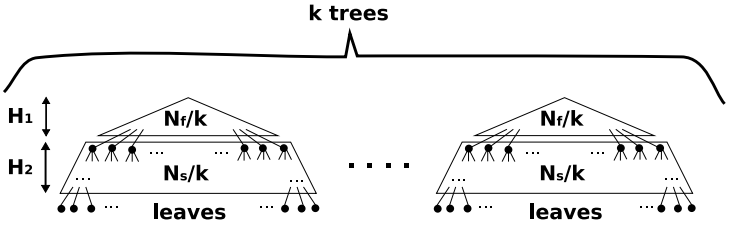


Figure 6.2: Optimal construction of K trees consisting of fast and slow nodes.

two types of peers — N_s slow peers and N_f fast peers, where a fast peer has upload bandwidth f times higher than a slow peer. In such a scenario, the optimal placement of peers that minimizes the height of each of the K trees is presented in Figure 6.2. Similar to homogeneous environments, each peer is an inner node in exactly one tree and a leaf node in $K-1$ trees. Additionally, fast nodes are placed at the top of the trees in order to reduce the height of the trees. Slow nodes have out-degree d, while fast nodes can upload f times faster, so their out-degree is fd. The out-degree of slow and fast nodes is derived from the fact that the total number of outgoing links of all peers must be equal to the total number of incoming links in the P2P overlay, while taking into account that the source node has out-going links, but does not have any incoming links. From this we have $N_s d + N_f f d = K(N_s + N_f - 1)$, which gives

$$d = \frac{K\left(N_s + N_f - 1\right)}{fN_f + N_s} \tag{6.3}$$

The height H_{het} of each heterogeneous tree constructed as in Figure 6.2 is calculated as $H_{het} = H_1 + H_2 + 1 + 1$, which is the sum of the height H_1 of the upper part of the tree composed of inner fast nodes only, the height H_2 of the lower part of the tree composed of slow inner nodes only, plus one level between the two parts of the tree and one level for the peers that are leaves in the tree (and which are inner nodes in other trees). The height H_1 is calculated using Eq. 6.1 as the height of a homogeneous tree of N_f/K fast nodes with out-degree fd:

$$H_1 = H(fd, \frac{N_f}{K}) = log_{fd}\left(\left(fd - 1\right)\frac{N_f}{K} + 1\right) - 1$$

The height H_2 is calculated as the height of a homogeneous tree of $\frac{N_s/K}{L_1 fd}$ slow

nodes with out-degree d

$$H_2 = H(d, \frac{N_s/K}{L_1 fd}) = log_d \left((d-1) \frac{N_s/K}{L_1 fd} + 1 \right) - 1$$

where $L_1 = L(fd, \frac{N_f}{K})$ is the number of leaves in the upper part, i.e., H_1. From these equations we derive a formula for the optimal height H_{het} of each optimal heterogeneous diffusion tree

$$H_{het} = log_{fd} \left((fd-1) \frac{N_f}{K} + 1 \right) + log_d \left((d-1) \frac{N_s}{(fd-1) N_f + K} + 1 \right) \tag{6.4}$$

where d is the out-degree of a slow node given by Eq. 6.3.

6.2.4 Evaluation

We compare the optimal tree heights, calculated in Equation 6.4, to the average height of diffusion trees that emerge in mesh overlays and are calculated by our custom-built simulator of mesh overlays. The simulator relies on the assumptions outlined in Sections 6.2.1 and 6.2.2. We used 50,000 nodes and studied both a homogeneous environment and environments with different levels of heterogeneity. In experiments involving heterogeneity, 10% of all nodes are fast nodes with upload bandwidth 2 and 8 times higher than the remaining slow nodes. The overall upload bandwidth in all overlays is the same. The results are presented in Figure 6.3. The results show that the average height of diffusion trees in homogeneous mesh overlays is around 2 levels above the optimal height, for all K. The reason for that is that in the optimal tree each peer is an inner node in exactly one diffusion tree, whereas in the trees emerging in mesh overlays a peer is located randomly and can be an inner node in several trees. The results show that when the level of heterogeneity increases, the gap between the height of diffusion trees in the mesh overlay and optimum trees significantly increases. For the case with 10% of peers being 8 times faster than the remaining slow peers, the average height of a diffusion tree in the mesh overlay for $K = 2$ is 3 times higher than the optimum and drops to 2 times over the optimum for $K = 16$. Increased heterogeneity results in higher importance of the location of fast and

slow peers in the tree. Worse performance for small K, in turn, is caused by higher variation in the height of diffusion trees - some leaves are much lower or higher than the others. This tree imbalance can be observed in Figure 6.4 that shows the cumulative distribution function (CDF) of the depth of leaf nodes in diffusion trees that emerge in a mesh overlay for both homogeneous and heterogeneous environments. The highest diffusion tree imbalance is for small K.

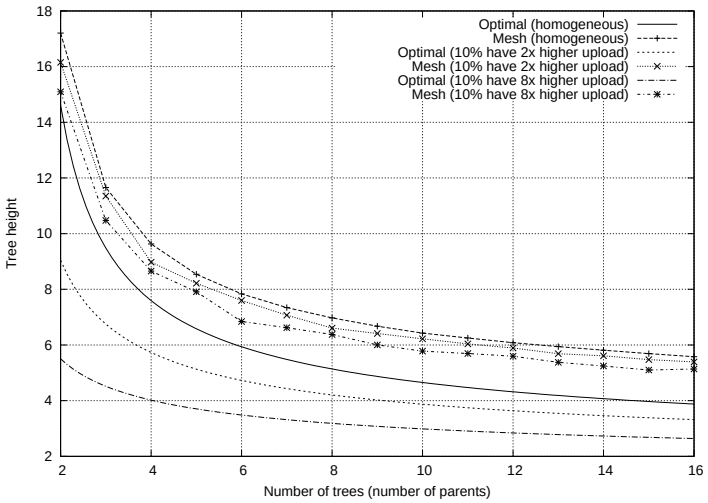


Figure 6.3: Average tree height for different number of parents and different heterogeneity levels.

Chunk propagation delay. In order to measure the impact of the tree height on the buffering delay, we analyze the time required to propagate a chunk through the diffusion trees in mesh overlays. Since in a mesh overlay, a peer can be placed anywhere in each diffusion tree, its buffering delay needs to accommodate the maximum difference between chunk arrivals in each distribution tree, which is equal to the chunk propagation delay. The propagation delay can be calculated as

$$delay = \frac{HsK(N-1)}{\sum_i^N upload_i}$$

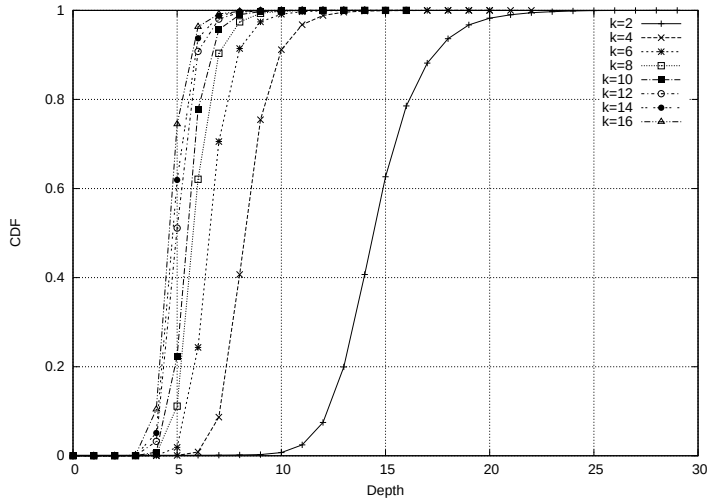


Figure 6.4: CDF of the height of diffusion trees in mesh overlays in a heterogeneous (10% peers have 4x upload) environment.

where H is the height of the tree, s the size of a chunk and the remaining part of the formula derives from the equation for the bandwidth of a link (see Section 6.2.2). It can be observed that this delay represents a trade-off between the height of a tree and the number K of distribution trees. Larger K produce shorter trees, however, it takes longer for a node to upload a chunk to all its children (since a node has more children). Smaller chunk sizes allow for their faster propagation, but more control messages are required to notify/request chunks. Propagation delay as a function of the number of diffusion trees (peer parents) is shown in Figure 6.5 (for an average upload bandwidth of 1,000kbps and a chunk size of 4KB). The results show that a small number of diffusion trees result in shorter buffering delays. However, small number of diffusion trees also means that the number of parents of each peer is small and this reduces robustness to peer failures.

6.3 Mesh Adaptation Algorithm

In the previous sections we showed that the heights of diffusion trees in mesh overlays are much higher than the optimal height. In this section we present an algorithm that adapts the location of high-bandwidth peers dynamically.

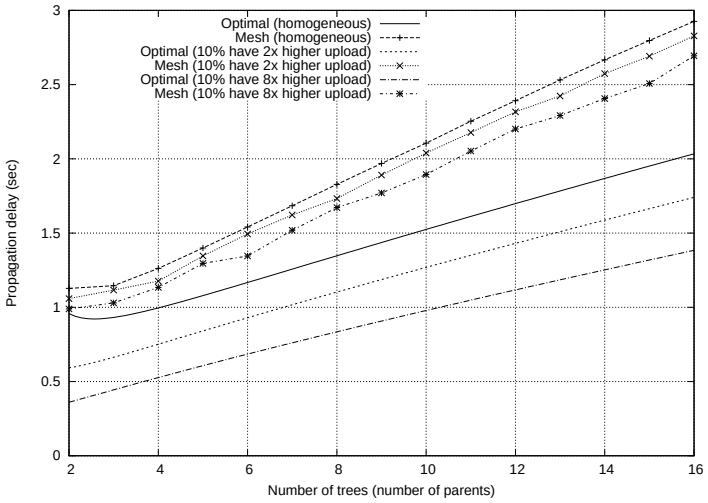


Figure 6.5: Propagation delay for varying number of trees for mesh overlays and the optimal case.

To shorten tree lengths it is advantageous to place high-bandwidth nodes near the source and low-bandwidth peers near the leaves.

6.3.1 Algorithm

We assume that peers have accurate information about their bandwidth, either through user input or through passive measurement techniques, such as [52]. Furthermore, the assumption is made that techniques are deployed that prevent peers from cheating about their bandwidth. To do this, peers may for example team up to compare effective bandwidth of neighbors with their indicated bandwidth and drop links to cheaters if the difference is too high. Alternatively, a reputation system like [35] could be implemented.

Each chunk being distributed from the source s to a peer p contains a hop count of the path it travelled. Peers can use this hop count as an estimate of their distance to the source. As explained in previous sections, the goal of each peer is to climb up, respectively to its upload bandwidth, in one diffusion tree and to become a leaf node in all other diffusion trees. In order to achieve this, each peer periodically executes Algorithm 3, which improves a peer's position in one diffusion tree. Since each parent of a peer is responsible for delivering only one tree, the algorithm aims at improving the peer's position

by replacing its current best parent (nearest to the source) with one of its grandparents that is closer to the source, subject to the conditions discussed below, effectively moving higher in one tree. Specifically, a peer p tries to find its parent *parent* and a grandparent *grandparent* (a parent of *parent*) that satisfies the following conditions:

1. distance(*grandparent*) < distance(bestparent(p))
2. upload(p) > upload(*parent*) OR bestparent(*parent*) $\neq$ *grandparent*

The first condition requires that *grandparent* is closer to the source than the current best parent. The second condition requires that the upload bandwidth of peer p is greater than the upload bandwidth of *parent* (child of *grandparent*) or *grandparent* is not the best parent of *parent* (*parent* does not climb up in that tree) and thus, *parent* can give up that *grandparent*. If these two conditions are satisfied, then peer p climbs up one level by: replacing *parent* as a child of *grandparent*, becoming a new parent of *parent* and losing one child, which becomes a child of *parent* (Figure 6.6 shows the exchange protocol). This way, the number of children and parents of all peers involved (p, *parent* and *grandparent*) remain unchanged and thus, the properties of the overlay required for achieving the optimal network throughput, described in Section 6.2.1, remain satisfied. The presented adaptation algorithm effectively results in each peer climbing up in one tree as long as its parent in this tree has lower upload bandwidth and climbing down in other trees (by giving up its position in these other trees to its children that climb up in these trees). The algorithm does not affect the network throughput as it does not change the number of children or parents of any peer.

6.3.2 Evaluation

In this section, we show the results of our evaluation of the adaptation algorithm presented in Section 6.3.1. The algorithm was implemented in our custom-built simulator and executed on 50,000 nodes with different ratios of upload bandwidth of fast and slow nodes. First, an initial mesh was created and tree heights calculated. Then, Algorithm 3 was executed to adapt the positions of all peers until no more adaptations were possible.

Algorithm 3 Adapting position of peer p in the mesh overlay

```
for all parent ← parent(p) do
  for all grandparent ← parent(parent) do
    if parent ≠ source then
      if distance(grandparent) < distance(bestparent(p)) then
        if upload(p) > upload(parent) or bestparent(parent) ≠
        grandparent then
          exchangePosition(p, parent, grandparent)
        end if
      end if
    end if
  end for
end for
```

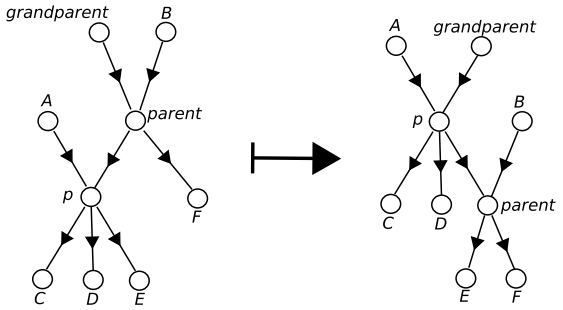


Figure 6.6: Peers p and *parent* exchange their positions respectively to *grandparent*.

In all experiments 10% of all peers had i ($i = \{2, 8\}$) times higher upload bandwidth than the remaining peers. The number of trees K varied from 2 to 16. As can be seen in Figure 6.7, there is a significant benefit of placing high-bandwidth nodes near the source. The average tree heights decrease by about 35% for two trees ($K = 2$). The same improvement is in the buffering delay, which is proportional to the tree height. Figure 6.8 shows the cumulative distribution function (CDF) of the depth of leaf nodes in diffusion trees in adapted mesh overlays. This figure, when compared to the analogous Figure 6.4, shows that diffusion trees in the adapted mesh overlays are significantly more balanced. However, despite of much improvement, some imbalance in the diffusion tree heights remains and, for that reason, the height of diffusion trees (and the corresponding buffering delay) is suboptimal. To achieve

optimal diffusion trees, a more system-wide adaptation is required, which is a focus of our future work.

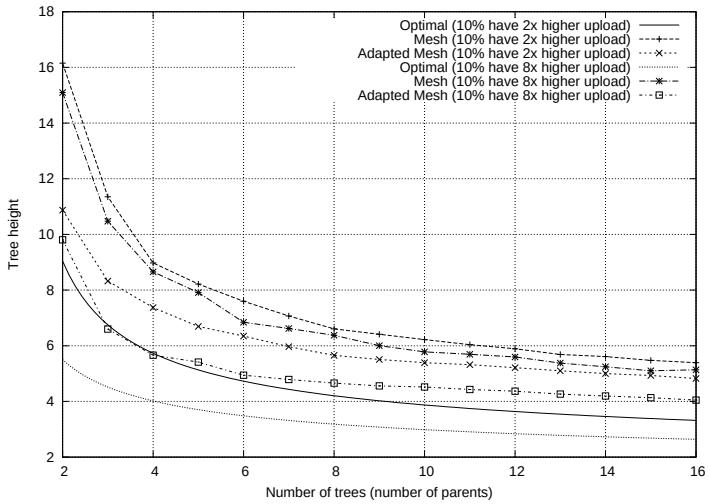


Figure 6.7: Average tree heights for different proportions of upload bandwidth and 50,000 peers.

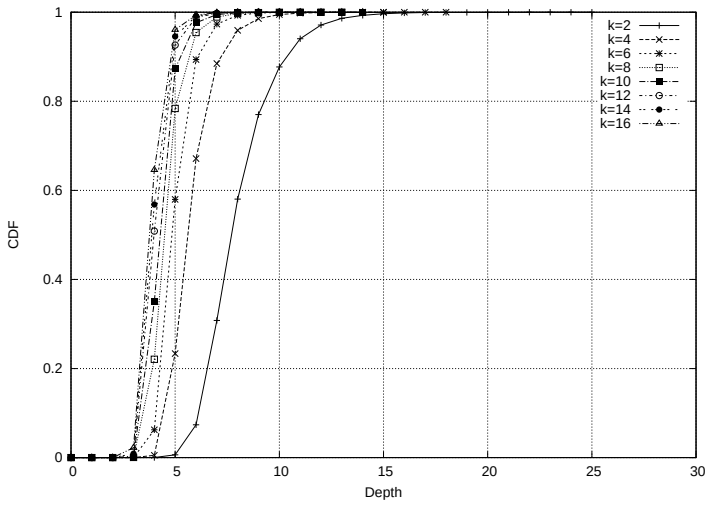


Figure 6.8: CDF of the height of diffusion trees in adapted mesh overlays in a heterogeneous (10% peers have 4x upload) environment.

6.4 Summary

In this chapter we have studied data diffusion in mesh overlays. We have shown that data chunks follow dynamically formed diffusion trees and we have analyzed the properties of these trees. The proposed structured view of meshes allows us to apply knowledge gained in the previous chapters about trees directly to mesh-based streaming approaches. Our results show that diffusion trees in mesh overlays are unbalanced with suboptimal height and thereby, buffering delay in mesh overlays is suboptimal. With the increasing heterogeneity in an overlay, the diffusion trees become even more suboptimal due to imperfect placement of fast peers in the diffusion trees. This implies that a mesh adaptation algorithm that places fast nodes closer to the source in exactly one diffusion tree shortens the height and improves the balance of diffusion trees, thereby significantly reducing the data buffering delay. We have presented such a mesh adaptation algorithm and we have shown that it improves tree heights.

Chapter 7

Conclusions

7.1 Summary

The problem of distributing continuous streaming content to a large peer population has been studied in the context of this thesis. The various challenges that come with such architectures have been discussed and solutions have been proposed.

We have developed novel architectures that are not relying on any hierarchical structure or different peer roles. These architectures take into account heterogeneous upload bandwidths and provide uniform download bandwidths to all participating peers. We have proposed dynamic adaptation algorithms that allow participating peers to benefit from higher reliability and shorter latency. A new tit-for-tat mechanism has been introduced to establish some fairness among peers and further enhance robustness.

Although the thesis focuses on tree-based architectures, mesh-based architectures have been discussed and it has been shown how mesh-based architectures can be analyzed with the same methods used for tree-based architectures.

The different architectures and algorithms have been supported by models and have been analyzed and evaluated in detail. The algorithms have been designed to be practical enough to be implemented. A prototype has been developed as a proof of concept and to evaluate the different architectures and algorithms.

While the main research goal presented in Chapter 1.3 has been achieved in the developed models, analysis, algorithms, and CrossFlux implementation, many problems remain open. The following section lists some perspectives for future research.

7.2 Perspectives of Future Work

Peer-to-peer media streaming is a research topic that, despite being widely studied, still offers a rich set of problems of various nature (e.g., networking, economics, social) that would deserve further study. In the direct continuation of this thesis, one can mention a couple of obvious directions for future work.

First, the architectures presented in this thesis are tree-based. In Chapter 6 it has been shown how mesh-based architectures can be analyzed with the same methods used in tree-based systems. Based on these results, one could design new mesh-based architectures and algorithms that would share the desirable properties of tree-based architectures, but provide more robust and dynamic behavior.

One could integrate more sophisticated fairness mechanisms in a mesh approach, such as maintaining a "balance of trade" for each pair of peers. The balance of trade between two peers, maintained independently by each peer, would be increased when sending chunks and decreased upon reception. one could accept for both direct exchanges and chunks transmitted via other peers. If the absolute value of the balance reaches a threshold (one peer contributing much more than the other), then the connection could be closed and the peers forced to connect to different neighbors. Free-riders could be detected and added in a blacklist.

There exist many other research directions and possible refinements worth pursuing. Yet, ultimately the success of a streaming solution will likely be driven by factors that are not just technical, but rather economical and social like emerging business models and copyright issues.

Bibliography

[1] E. Adar and B. A. Huberman. *Free Riding on Gnutella*, volume 5, page 2000. Oct 2000.

[2] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proceedings of the ACM SIGCOMM 2002 conference*, pages 205–220, Aug 2002.

[3] E. W. Biersack, P. Rodriguez, and P. Felber. Performance analysis of peer-to-peer networks for file distribution. In *Proceedings of 5th International Workshop on Quality of future Internet Services (QofIS)*, pages 1–10, Sep 2004.

[4] B. Biskupski, R. Cunningham, J. Dowling, and R. Meier. High-bandwidth mesh-based overlay multicast in heterogeneous environments. In *Proceedings of the 2nd International Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications (AAA-IDEA'06)*, pages 4–11. ACM Press, 2006.

[5] B. Biskupski, R. Cunningham, and R. Meier. Improving throughput and node proximity of p2p live video streaming through overlay adaptation. In *Proceedings of the 9th IEEE International Symposium on Multimedia (ISM 2007)*, pages 245–252. IEEE Computer Society, 2007.

[6] B. Biskupski, M. Schiely, P. Felber, and R. Meier. Tree-based analysis of mesh overlays for peer-to-peer streaming. In *Proceedings of the 8th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS)*, pages 126–139, Jun 2008.

[7] D. Carra, R. Lo Cigno, and E. W. Biersack. Introducing heterogeneity in performance analysis of p2p networks for file distribution. Technical Report DIT-04-113, University of Trento, Dec 2004.

[8] M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth multicast in a cooperative environment. In *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*, pages 298–313, Oct 2003.

[9] M. Castro, P. Druschel, A.-M. Kermarrec, and A.I.T. Rowstron. Scribe: a large-scale and decentralized application-level Multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8):1489–1499, Oct 2003.

[10] J. Chakareski, S. Han, and B. Girod. Layered coding vs. multiple descriptions for video streaming over multiple paths. In *Proceedings of the 11th ACM international conference on Multimedia (MULTIMEDIA'03)*, pages 422–431, New York, NY, USA, 2003. ACM.

[11] Y. Chawathe. Scattercast: An adaptable broadcast distribution framework. *Multimedia Systems*, 9(1):104–118, 2003.

[12] Y. Chu, S. Rao, and H. Zhang. A case for end system multicast. In *Proceedings of ACM Sigmetrics*, pages 1–12, Jun 2000.

[13] Y.-H. Chu, J. Chuang, and H. Zhang. A case for taxation in peer-to-peer streaming broadcast. In *Proceedings of PINS*, pages 205–212, 2004.

[14] Y.-H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 1–12, 2000.

[15] Y.-H. Chu and H. Zhang. Considering altruism in peer-to-peer internet streaming broadcast. In *Proceedings of IEEE NOSSDAV*, pages 10–15, 2004.

[16] B. Cohen. Incentives build robustness in BitTorrent. In *Proceedings of 1st Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, Jun 2003.

[17] B. Cohen. Incentives to build robustness in BitTorrent. Technical report, http://www.bittorrent.com/bittorrentecon.pdf, May 2003.

[18] A. da Silva, E. Leonardi, M. Mellia, and M. Meo. A bandwidth-aware scheduling strategy for p2p-tv systems. In *Proceedings of International Conference on Peer-to-Peer Computing*, pages 297–288. IEEE Computer Society, 2008.

[19] P. Felber and E. W. Biersack. *O. Babaoglu, M. Jelasity, A. Montresor, C. Fetzer, S. Leonardi, A. van Moorsel, M. van Steen (Eds.): Self-Star Properties in Complex Information Systems*, chapter Cooperative Content Distribution: Scalability through Self-Organization, pages 343–357. Springer-Verlag, 2005.

[20] Gnutella. rfc-gnutella.sourceforge.net, 2009.

[21] P. Gummadi, S. Saroiu, and S. Gribble. A measurement study of napster and gnutella as examples of peer-to-peer file sharing systems. *Multimedia Systems Journal*, 9(2):170–184, 2003.

[22] M. Hefeeda, A. Habib, B. Boyan, D. Xu, and B. Bhargava. PROMISE: peer-to-peer media streaming using CollectCast. In *Proceedings of the 11th ACM international conference on Multimedia (MULTIMEDIA'03)*, pages 45–54, New York, NY, USA, Nov 2003.

[23] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross. Insights into PPLive: A measurement study of a large-scale p2p IPTV system. In *Proceedings of IPTV Workshop, International World Wide Web Conference*, 2006.

[24] Cisco Systems Inc. Cisco visual networking index: Forecast and 2008-2013 methodology, 2009.

[25] M. Izal, E. W. Biersack, P. A. Felber, G. Urvoy-Keller, A. Al Hamra, and L. Garces-Erice. Dissecting BitTorrent: Four months in a torrent's lifetime. In *Proceedings of the 5th Passive and Active Measurement Workshop*, pages 1–11, Apr 2004.

[26] J. Jannotti, D. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole. Overcast: Reliable multicasting with an overlay network. In *Proceedings of the 4th Symposium on Operating System Design and Implementation (OSDI)*, pages 197–212, Oct 2000.

[27] X. Jiang, Y. Dong, D. Xu, and B. Bhargava. Gnustream: A P2P media streaming system prototype. In *Proceedings of International Conference on Multimedia and Expo (ICME)*, volume 2, pages 325–328, Jul 2003.

[28] C. Jin, Q. Chen, and S. Jamin. Inet: Internet topology generator. Technical Report CSE-TR443-00, University of Michigan, 2000.

[29] R. Kumar, Y. Liu, and K. Ross. Stochastic fluid theory for p2p streaming systems. In *Proceedings of the 26th IEEE International Conference on Computer Communication (INFOCOM)*, pages 919–927, May 2007.

[30] J. Li. Peerstreaming: A practical receiver-driven peer-to-peer media streaming system. Technical Report MSR-TR-2004-101, Microsoft Research, Sep 2004.

[31] W. S. Lin, H. V. Zhao, and K. J. R. Liu. A game theoretic framework for incentive-based peer-to-peer live-streaming social networks. In *Proceedings of ICASSP*, pages 48–56, 2008.

[32] Z. Liu, Y. Shen, S. Panwar, K. W. Ross, and Y. Wang. Using layered video to provide incentives in p2p streaming. In *Proceedings of SIGCOMM P2P-TV Workshop*, pages 311–316, 2007.

[33] N. Magharei and R. Rejaie. PRIME: Peer-to-peer receiver-driven mesh-based streaming. In *Proceedings of the 26th IEEE International Conference on Computer Communication (INFOCOM)*, pages 1415–1423, May 2007.

[34] N. Magharei, R. Rejaie, and Y. Guo. Mesh or multiple-tree: A comparative study of live p2p streaming approaches. In *Proceedings of the 26th IEEE International Conference on Computer Communication (INFOCOM)*, pages 1424–1432, May 2007.

[35] A. Nandi, T.-W. Ngan, A. Singh, P. Druschel, and D. S. Wallach. Scrivener: Providing incentives in cooperative content distribution systems. In *Proceedings of Middleware*, pages 270–291, 2005.

[36] V. N. Padmanabhan, H. J. Wang, and P. A. Chou. Resilient peer-to-peer streaming. In *Proceedings of the IEEE International Conference on Network Protocols (ICNP)*, page 16, Nov 2003.

[37] V. S. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr. Chainsaw: Eliminating trees from overlay multicast. In *Proceedings of the 4th International Workshop on P2P Systems (IPTPS)*, pages 127–140, 2005.

[38] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. Almi: An application level multicast infrastructure. In *Proceedings of the 3rd conference on USENIX Symposium on Internet Technologies and Systems*, page 5, Mar 2001.

[39] L. Peterson, D. Culler, T. Anderson, and T. Roscoe. A blueprint for introducing disruptive technology into the Internet. In *SIGCOMM Computer Communications Review*, volume 33, pages 59–64. ACM, Oct 2003.

[40] F. Pianese, D. Perino, J. Keller, and E. Biersack. PULSE: an adaptive, incentive-based, unstructured p2p live streaming system. *IEEE Transactions on Multimedia, Special Issue on Content Storage and Delivery in Peer-to-Peer Networks*, 9(6):1645–1660, 2007.

[41] F. Picconi and L. Massoulié. Is there a future for mesh-based live video streaming? In *Proceedings of the 8th International Conference on Peer-to-Peer Computing*, pages 289–298. IEEE Computer Society, 2008.

[42] F. Picconi and L. Massoulié. Isp friend or foe? making p2p live streaming isp-aware. In *Proceedings of the 29th IEEE International Conference on Distributed Computing Systems (ICDCS'09)*, pages 413–422, Washington, DC, USA, 2009. IEEE Computer Society.

[43] PPLive. www.pplive.com, 2009.

[44] S. Rao, V. Padmanabhan, S. Seshan, and H. Zhang. The impact of heterogeneous bandwidth constraints on DHT-based multicast protocols. In *Proceedings of the 4th International Workshop on P2P Systems (IPTPS)*, pages 115–126, Feb 2005.

[45] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *SIGCOMM Computer Communication Review*, 27(2):24–36, 1997.

[46] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking*, pages 156–170, 2002.

[47] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking*, 2002.

[48] M. Schiely and P. Felber. Peer-to-peer distribution architectures providing uniform download rates. In *Proceedings of the International Symposium on Distributed Objects and Applications (DOA)*, pages 1083–1096, Oct 2005.

[49] M. Schiely and P. Felber. *CrossFlux: An Architecture for Peer-to-Peer Media Streaming*, volume 8 of *Emerging Communication: Studies in New Technologies and Practices in Communication*, pages 342–358. R. Baldoni and G. Cortese and F. Davide and A. Melpignano (Eds.), 2006.

[50] M. Schiely and P. Felber. Tit-for-tat revisited: Trading bandwidth for reliability in p2p media streaming. *Multiagent and Grid Systems*, 5(2):197–215, 2009.

[51] M. Schiely, L. Renfer, and P. Felber. Self-organization in cooperative content distribution networks. In *Proceedings of the IEEE International Symposium on Network Computing and Applications (NCA)*, pages 109–116, Jul 2005.

[52] J. Strauss, D. Katabi, and F. Kaashoek. A measurement study of available bandwidth estimation tools. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement (IMC'03)*, pages 39–44, New York, NY, USA, 2003.

[53] Akamai Technologies. www.akamai.com, 2009.

[54] D. Tran, K. Hua, and T. Do. Zigzag: An efficient peer-to-peer scheme for media streaming. In *Proceedings of the IEEE International Conference on Computer Communication (INFOCOM)*, pages 1283–1292, 2003.

[55] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 271–284, Dec 2002.

[56] V. Venkatraman, K. Yoshida, and P. Francis. Chunkyspread: Heterogeneous unstructured end system multicast. In *Proceedings of the 14th IEEE International Conference on Network Protocols*, pages 2–11, Nov 2006.

[57] F. Wang, J. Liu, and Y. Xiong. Stable peers: Existence, importance, and application in peer-to-peer live video streaming. In *Proceedings of the IEEE International Conference on Computer Communication (INFOCOM)*, pages 1364–1372, 2008.

[58] W. Wang, D. A. Helder, S. Jamin, and L. Zhang. Overlay optimizations for end-host multicast. In *Proceedings of the International Workshop on Networked Group Communications (NGC)*, pages 154–161, Oct 2002.

[59] X. Yang and G. de Veciana. Service capacity of peer-to-peer networks. In *Proceedings of the IEEE International Conference on Computer Communication (INFOCOM)*, pages 1–11, Mar 2004.

[60] Zattoo. www.zattoo.com, 2009.

[61] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum. Coolstreaming/DONet: A data-driven overlay network for peer-to-peer live media streaming. In *Proceedings of the IEEE International Conference on Computer Communication (INFOCOM)*, pages 2102–2111, Mar 2005.

Appendix A

Prototype Implementation

To evaluate the algorithms and architectures presented in this thesis, a proof-of-concept prototype has been implemented. It is entirely written in Java and was used for the evaluation of the algorithms in Chapter 5 on different testbeds like ModelNet and PlanetLab, but also as input to our own simulator and as a prototype for demonstration use.

A.1 System Overview

We use a single source on a desktop machine to encode a single video and stream it repeatedly to its children. The video is encoded by the HTTP streaming feature of the VLC framework,[1] which serves HTTP requests and splits the stream into TCP packets to be sent over the network. The CrossFlux client on the source initiates an HTTP GET request to VLC and receives the TCP packets. The packets are then stored in a buffer in the source and are scheduled for sending to the children.

The CrossFlux clients running at the source's children request the packages from the source and also store them in their buffer for playing and for forwarding to their respective children. The VLC instance on the children starts an HTTP streaming session to the CrossFlux client and waits for the TCP packets. As soon as the buffer has enough packets, they are sent to VLC for playback.

[1] Available from http://www.videolan.org.

A.2 Buffer Management

A critical component in the prototype implementation is the buffer and its management. There is a trade-off between memory usage and robustness: the more blocks are stored in the buffer, the robuster the system gets, but the higher the memory usage is. If too few blocks are kept in buffer, peers are not able to sustain bandwidth fluctuations and peer exchanges.

Therefore we use different buffer sizes for different network conditions. Packets are deleted if they are older than the current playback position plus a specified time window for serving children that are behind the current peer playback time.

For each child a separate buffer map is used to mark which chunks have been successfully received. Children can have slightly different playback positions in the stream due to peer exchanges or short network outages. If the difference becomes too high than a threshold, the child is purged and has to initiate a new join request starting from the source.

A.3 Network Layer

The network layer has been designed such that it can easily be replaced by a different implementation. With this design we can use the same prototype for different settings like simulation frameworks, locally emulated networks or global distributed systems such as Planetlab.

The basic implementation uses Java sockets to connect to other peers. For each child two connections are established: a control connection and a data connection. A peer first establishes a control connection to a candidate child to initiate a join procedure. It exchanges protocol packets with the child until the parent-child relation is established and data streaming can start. Data packets are then sent over the data connection on a different port.

Each data packet is acknowledged by the receiver. The sender marks the packet as received and deletes it according to the buffer management described earlier. A packet which is never acknowledged will still be deleted after a timeout to avoid buffer overflows.

A.4 HeapTop

The peer exchange described in *HeapTop* has been implemented with an announcement protocol: if a peer wants to exchange position with its parent it announces its intent to the parent. If the parent acknowledges the request, the connections are closed and new connections to exchanged children and parents are opened.

Our implementation assumes that peers do not refuse connection requests. This assumption should be relaxed in a real world implementation.

The exchange needs to be very fast in order to not fill up buffers with chunks that are received from the source. If the buffer reaches a threshold during such an position exchange, a small number of chunks is skipped and the children continue streaming with a few missed video frames.

A.5 Failure Recovery

Failures in connections are detected by means of indirect heartbeats. If a parent does not send a chunk within a timeout period, the parent is assumed to have failed and the backup links are used to search for a replacement peer. Children failures are detected by timeouts on chunk acknowledge packets. If a child does not send a acknowledgement for a number of packets, then it is assumed to have failed and it is purged from the list of children.

The detection of failures directly depends on the chosen timeout parameters. We tested different values based on the network setup. In a real world implementation these values should be adapted to the network conditions and the expected failure rate.

Appendix B

List of Publications

[1] B. Biskupski, M. Schiely, P. Felber, and R. Meier. Tree-based analysis of mesh overlays forpeer-to-peer streaming. In *Proceedings of the 8th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS)*, pages 126–139, Jun 2008.

[2] M. Schiely and P. Felber. Peer-to-peer distribution architectures providing uniform download rates. In *Proceedings of the International Symposium on Distributed Objects and Applications (DOA)*, pages 1083–1096, Oct 2005.

[3] M. Schiely and P. Felber. *CrossFlux: An Architecture for Peer-to-Peer Media Streaming*, volume 8 of *Emerging Communication: Studies in New Technologies and Practices in Communication*, pages 342–358. R. Baldoni and G. Cortese and F. Davide and A. Melpignano (Eds.), 2006.

[4] M. Schiely and P. Felber. Tit-for-tat revisited: Trading bandwidth for reliability in p2p media streaming. *Multiagent and Grid Systems*, 5(2):197–215, 2009.

[5] M. Schiely, L. Renfer, and P. Felber. Self-organization in cooperative content distribution networks. In *Proceedings of the IEEE International Symposium on Network Computing and Applications (NCA)*, pages 109–116, Jul 2005.