The Internet has become a popular platform for video distribution. Live video streams, movies and video clips are being served to millions of users. Most of these are distributed using the traditional client-server model. As in a theatre, a single source (server) distributes the video images to all the viewers (clients). The size of the audience is necessarily limited: at some point, all seats are simply booked.

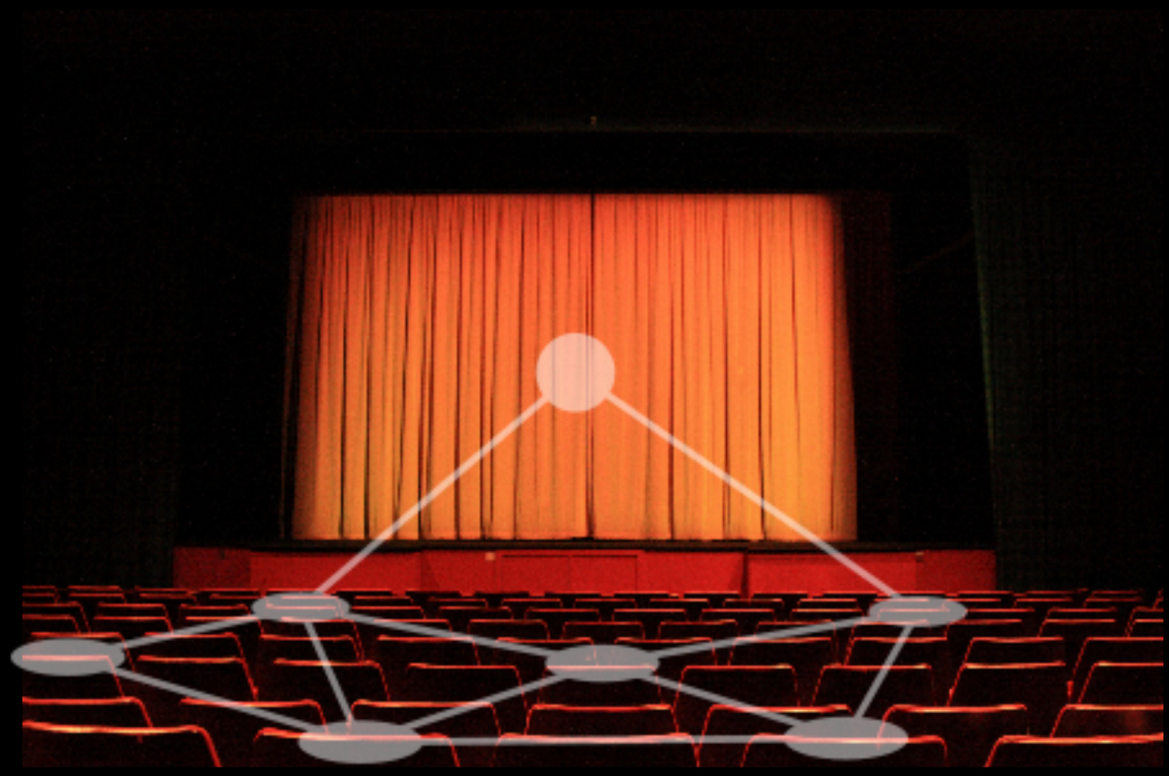
Peer-to-peer technology offers a solution to this problem. Instead of forcing users (peers) to download the video stream from a single source, it allows the viewers to provide each other with the video stream. Unfortunately, users do not have an inherent incentive to help with the distribution of the video stream. Those who shoulder less than their fair share of the costs are called free-riders, and if too many of them are present, well-behaving users will be unable to watch the video stream at its intended quality.

In this thesis, we present the design and analysis of peer-to-peer video streaming algorithms that punish free-riders. We thus provide the users with an incentive to help distributing the video stream, Our work brings peer-to-peer systems a step closer towards scaling up to millions of users, at which point one no longer needs an expensive client-server theatre to distribute video streams to a large audience.

Free-riding Resilient Video Streaming in Peer-to-Peer Networks

Jacob Jan David Mol

Free-riding Resilient Video Streaming in Peer-to-Peer Networks



Jacob Jan David Mol

TU Delft Delft University of Technology

Free-riding Resilient Video Streaming in Peer-to-Peer Networks

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof.ir. K.C.A.M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op dinsdag 26 januari 2010 om 12:30 uur

door **Jacob Jan David MOL**

informatica ingenieur
geboren te Delft

Dit proefschrift is goedgekeurd door de promotor:
Prof.dr.ir. H.J. Sips

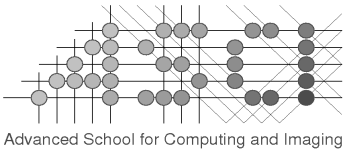
Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof.dr.ir. H.J. Sips	Technische Universiteit Delft, promotor
Dr.ir. D.H.J. Epema	Technische Universiteit Delft, copromotor
Prof.dr.ir. R.L. Lagendijk	Technische Universiteit Delft
Prof.dr.ir. M.R. van Steen	Vrije Universiteit Amsterdam
Prof.dr.ir. P.H.N. de With	Technische Universiteit Eindhoven
Prof.dr. S. Haridi	Royal Institute of Technology, Zweden
Prof. D.S. Reeves, PhD	North Carolina State University, VS
Prof.dr.ir. K.G. Langendoen	Technische Universiteit Delft (reservelid)

Published and distributed by: Jan David Mol
E-mail: jjdmol@gmail.com

Cover photo by Lesley Middlemass. ⓒ ⓘ ⓢ
Printed in The Netherlands by Wöhrmann Print Service.

This work was performed in the context of the Freeband I-Share project. Freeband Communication is supported by the Dutch Ministry of Economic Affairs through the BSIK program (BSIK 03025).



This work was carried out at the ASCI graduate school. ASCI dissertation series number **190**.

Preface

The thesis that you see in front of you would not be here if not for the many people who helped and supported me during my PhD track. After all, it is through interaction with others that we can shape our thoughts and derive solutions to the problems that we face. But also support outside of work is vital in order to sustain a project which takes several years to complete. Over the years, the list of people who deserve a big "thank you" has grown, as such things inevitably do.

I start with my supervisor Dick Epema and my promotor Henk Sips. Not only did they teach me how scientific papers are written, their guidance and critical thinking were essential in shaping this work. They, and my other co-authors, Johan Pouwelse, Arno Bakker, Michel Meulpolder, and Yue Lu, aided me in the research and writing. But support and critical thinking was also provided by many other colleagues, such as Paweł Garbacki, Alexandru and Ana Iosup, and Gertjan Halkes. I would like to thank the Parallel and Distributed Systems Group at Delft University of Technology for providing me with an environment for obtaining a PhD. I thank Freeband Communication for the funding of my research, as well as for providing a research context in the form of the I-Share research project, which focuses on sharing technologies in virtual communities.

For their strong support from home, I like to thank my wife Christie and my daughter Selene. Both of them provide the love that I need to function, to allow me to focus on the complex research questions, and to help deal with the inevitable stress. Similarly, my parents Marga and Nanne, as well as my sister Karin, have supported me throughout my PhD track.

As you all know, it is impossible to provide a complete list. I thank you for your support, or at the very least, for reading this preface.

Jan David Mol

Contents

Chapter 1

Introduction

A MOVIE is actually an illusion created by showing a series of pictures in rapid succession. The trick was already known in second-century China [71], but remained a curiosity up to the end of the 19^{th} century. Around 1888, both the motion picture camera and projector were invented and used to create the first film [82]. For the first time, it was possible to record scenes in an automated fashion and to show them to an audience. Wherever the projectors and films went, groups of people could watch the movie simultaneously on a large screen. The audience was substantially increased in size with the rise of television broadcasting after its invention in 1928 [18]. Television allowed viewers to watch from different locations, such as the comfort of their own homes. Both live events and recorded films could be sent to eventually millions of television sets simultaneously. A third boost in the popularity of moving pictures came at the end of the 20^{th} century with the invention of the Internet [86] and of the World Wide Web [14]. The moving pictures migrated to this platform, and with the rise of broadband Internet connections [74], end users became able to receive video of acceptable quality on their home computers.

A common setup for Internet video streaming is the *client-server (CS) architecture* [36, 43], in which the end users (clients) obtain the video stream from a relatively small number of servers [62]. Even though such techniques place a large burden on the servers in terms of bandwidth costs, such a setup is still popular due to its simplicity and predictable performance. Large companies, such as Google, can afford to deploy server parks in order to serve millions of users every day. Nevertheless, the cost of the bandwidth required to serve for example a television channel to millions

of viewers simultaneously is prohibitive for even the largest of companies.

An alternative setup for Internet video streaming is offered by the *peer-to-peer (P2P) architecture*, in which the end users (peers) act as servers as well as clients. A possibly low-capacity server (injector) is used to inject the stream into a P2P network by sending it to several peers. The peers subsequently forward the stream among each other. Since most peers are actually served by other peers, the load on the injector remains low. Every peer that joins the network both adds capacity and uses it. The P2P architecture is thus a scalable way of serving a video stream, as long as enough capacity is provided and can be put to effective use. However, in practice, the capacity that can be delivered through a P2P network is limited, and the quality of the video that can be streamed over a P2P network is similar to the quality of regular TV [7, 52, 100]. High-definition television (HDTV) is far beyond the reach of current P2P networks. The capacity of a P2P network is limited for three reasons.

First, the capacity that can be provided by a P2P network is limited by the total upload bandwidth of the peers. Since every peer desires to obtain the video stream, the number of uploaders and downloaders is the same. The video bit rate that can be streamed over a P2P network to all peers is thus limited by the average upload bandwidth of the peers. However, the peers in a typical P2P network do not have sufficient upload capacity to provide each other with an HDTV stream. The Internet connections of end users are typically asymmetric. Both ADSL and cable broadband connections provide several times more download capacity than upload capacity. Even though the download capacity of many broadband connections is enough to receive an HDTV stream, the upload capacity is only a fraction of what is needed to provide enough capacity to serve such a stream over a P2P network. The speeds of the Internet connections will likely continue to improve, but cannot be influenced by the P2P video distribution algorithm.

Secondly, the upload capacity that exists at the peers is not necessarily made available. The peers have no inherent incentive to provide their upload capacities to the P2P network in order to serve other peers. Peers that do not provide their upload capacity are called *free-riders* [6, 49, 88, 101], because they try to obtain the video stream for free. The upload capacity of free-riders thus remains unused, which shifts the cost of serving the video stream to the other (altruistic) peers. The altruistic peers, faced with an increased cost, thus will have an incentive to become free-riders as well. The altruistic peers that remain do not necessarily have enough capacity to provide all of the peers in the network with the video stream.

Thirdly, the upload capacity that is available in a P2P network can not always be put to effective use. A P2P network is not fully connected. Instead, every peer in a P2P network maintains a small set of connections to other peers. A peer with upload capacity to spare is thus not necessarily connected to another peer that needs to be served. Although the P2P network can be augmented with additional connections, the P2P network is limited by the connections that can be formed as well. A large fraction of the Internet connections are behind a filtering device, such as a firewall or a Network Address Translator (NAT) [7, 100], either as a security measure or as part of the connection setup. The filtering devices block incoming connections to the peers behind them. As a result, two peers behind a filtering device cannot connect to each other, which limits the connectivity within the P2P network. If the achievable connectivity is too low, supply and demand cannot be matched in the P2P network. The upload capacity of some peers will thus remain unused, effectively forcing them to free-ride, while others are unable to obtain the video stream.

It is the subject of this thesis to investigate and counter the effects of free-riding within P2P video streaming networks. We will present, simulate, and deploy several P2P video streaming algorithms which incorporate incentives for peers to provide their upload bandwidth to the P2P network. Separate algorithms will be presented for streaming live video and prerecorded video, and the performance of most of these algorithms will be assessed by means of mathematical analysis. Furthermore, we study the effects of limited connectivity on the performance that can be obtained in P2P networks by providing analytical bounds supported by simulations, as well as by measuring deployed P2P networks.

In the remainder of this chapter, we will first provide the context of our research. Then, we will provide a more detailed introduction to the transportation of video streams over networks, and to the nature of P2P video streaming networks. Then, we will discuss the key challenges faced in this thesis, followed by an outline covering the remainder of this work.

1.1 Research Context

The research of which the results are presented in this thesis was performed in the context of the I-Share project [3], which is funded by the Dutch government through the Freeband Communications programme [2]. The I-Share project investigates the sharing of resources in virtual communities, which are (dynamic) groups of nodes

that are willing to collaborate for the better of the whole. Such communities can be formed over any network, ranging from an ad-hoc network of hand-held devices to global Internet communities. Such communities typically consist of end users with limited resources, which have a joint interest in sharing a certain type of resource. For example, a group of end users may form a virtual community based on friendship or a common interest.

Within a community, content and bandwidth are shared among the users. Protocols are needed to maintain the virtual communities, and to locate the content the user wants to obtain. If the amount of content shared within the virtual community is large enough, the community becomes a social network in which users can be linked by their taste. A P2P client called Tribler [80] has been developed within the I-Share project to exploit these links by building a social P2P network based on taste. Users within the Tribler network converge based on the content they like, which allows them to locate similar content and to exchange resources with people with similar interests. One of the aims of Tribler is to enable non-professional content producers, such as ordinary individuals, a local football club, a church, or even a small television station, to stream a live or a prerecorded video stream to a set of viewers. Such small organisations typically have very limited resources, which are insufficient to deliver a video stream to more than a few peers directly. By incorporating P2P streaming algorithms into Tribler, the distribution costs for the video stream are lowered substantially. Professional content producers can benefit from such a cost reduction as well.

In this thesis, we focus on a single end user who is interested in sharing a specific video stream with a group of peers that are interested in viewing the same stream. We aim to distribute the burden of providing the video across the end users by using P2P algorithms to distribute the video data. The algorithms we present in this thesis are designed to be included into Tribler. For that reason, our algorithms are designed to be deployed in the Internet, by taking the characteristics of peers in the Internet into account.

1.2 Video Streaming

Video streaming is the transportation of a video from its source to the end user. The raw video has to be compressed and packetised before it is transported over the network. The packets are collected and decompressed by the receivers, in order to re-

construct the video stream. The constraints on this process depend on the network properties such as available bandwidth, but also on the type of streaming that is used. For instance, streaming a live video event introduces stricter real-time constraints than streaming a prerecorded video. In this section, we will discuss the modes of streaming we distinguish, followed by the key aspects of video processing that are relevant for this thesis.

1.2.1 Streaming Modes

We distinguish three modes of video streaming based on the moments at which the video is generated, downloaded, and viewed.

Live Video Streaming, in which the video stream is being generated at the same time as it is being downloaded and viewed by the users. All of the users will aim to watch the latest generated content. The end-user experience is thus comparable to a live TV broadcast or a webcam feed. The user needs a download speed at least equal to the playback speed if data loss is to be avoided.

Video-on-Demand, in which the video stream is generated beforehand but the users will view the stream while they are downloading it. An on-line video store, for example, would be using this model to provide movies on-demand to its customers. The users typically start watching the video stream from the beginning. At any moment, all of the users will be watching different parts of the video. Video clips available on websites such as YouTube [5] are examples of video-on-demand streaming. Similar to live video streaming, the user needs a download speed at least equal to the playback speed to avoid data loss.

Off-line Downloading, in which the generation, downloading, and viewing of the video stream are completely decoupled. As with video-on-demand, the user can download the video stream only after it has been generated. However, with off-line downloading, the user has to wait for the video to be completely downloaded before he can view it. The experience is thus similar to that of a VCR. The advantage of off-line downloading is the lack of real-time constraints. The speed at which the video is downloaded has no impact on the viewing experience, which allows the user to download a video of any quality regardless of the bandwidth he has available.

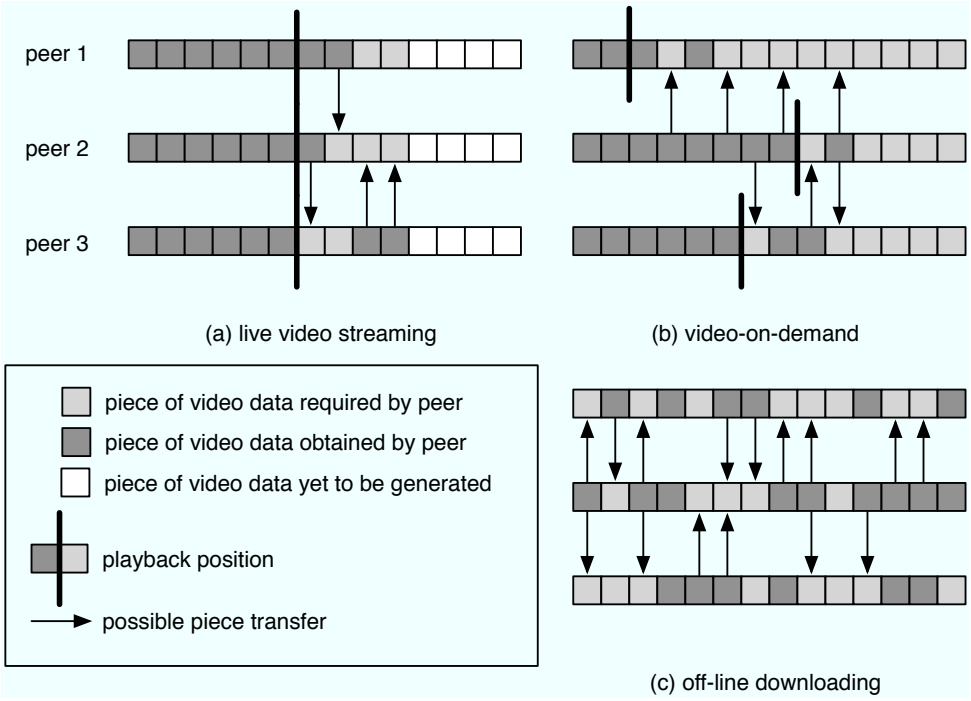


Figure 1.1: The three modes of video streaming.

Even though the differences in end-user experience are minor between these modes, the implementation differences are significant. In live video streaming, only the latest generated data is available, and wanted by all peers. In contrast, both video-on-demand and off-line downloading allow any peer to offer as well as request any piece of data. Figure 1.1 illustrates the differences between the three modes. In each subfigure, three peers exchange data for a specific mode. The blocks represent pieces of video data such as a series of bytes or a group of frames. Dark gray blocks have been obtained by the corresponding peer, light gray blocks have not. Figure 1.1(a) represents the case of live video streaming, with all the peers having the same playback position (represented by a black line). All of the peers require the pieces from the playback position up to the latest piece generated by the live video feed. Their interests are thus symmetrical. Most of the remainder of the live feed has yet to be generated, which is represented by white blocks. Figure 1.1(b) depicts video-on-demand, in which the playback position differs between peers, making the interests

Format	Resolution	Frame rate	Raw data rate
Webcam	640x480	15 Hz	0.1 Gbps
SDTV (NTSC)	720x480	30 Hz	0.2 Gbps
SDTV (PAL)	720x576	25 Hz	0.2 Gbps
HDTV 720p25	1280x720	25 Hz	0.5 Gbps
HDTV 1080p25	1920x1080	25 Hz	1.2 Gbps
HDTV 1080p50	1920x1080	50 Hz	2.3 Gbps

Table 1.1: Common TV video formats and their properties.

of the peers asymmetric. For example, a peer far ahead in the video (peer 2) has no need for the data that has been obtained by a peer that has just started playback (peer 1). On the other hand, all of the video is available from the start until the end. A peer can thus download pieces that lay substantially ahead of its playback position. Finally, Figure 1.1(c) shows off-line downloading. The peers have no playback position while downloading, as playback commences only after the download has been completed. As a result, the peers can obtain the pieces in any order, and are interested in receiving all of them. The interests of the peers are once again symmetrical. In both video-on-demand and off-line downloading, a peer does not immediately depart when it has completed the download. Instead, it becomes a *seeder* and offers all of the pieces to the rest of the peers. The upload capacity offered by the seeders thus remains available, but the seeders do not require any download capacity. The average upload capacity available to the peers that are still downloading the video stream (the *leechers*) thus increases whenever a peer becomes a seeder. In live video streaming, seeding is not possible as new pieces are continuously generated, and the peers are only interested in the latest pieces. Off-line downloading is, from an algorithmic point of view, actually a file-sharing problem rather than a streaming problem. The end user merely happens to play the video after it is done downloading it. As such, we will not consider off-line downloading in this thesis.

1.2.2 Video Processing

A video stream is transported from a camera to the screen (and the accompanying audio signal to the speakers) of an end user. First, the video is digitised and recorded by the video camera, by taking pictures (called *frames*) at a constant rate, and recording the audio in parallel. Each frame consists of millions of pixels, resulting in a raw data

stream with a bitrate which is typically too large to propagate to an end user. Table 1.1 lists common TV formats as well as their raw data rates. Even traditional TV quality (SDTV) requires a data rate several times the available bandwidth of a typical end user. For that reason, the video stream is compressed using a video compression algorithm (called *codec*). Depending on the quality that has to be delivered to the end user, the resolution and frame rate can be reduced as well. These forms of compression reduce the quality of the video stream. A trade-off between quality and bitrate is thus required, and depends, among other things, on the properties of the network used to deliver the video to the end user.

The compressed stream is divided into packets, which are transported over the network to the client, which is hosted on the machine of the end user. The client orders the packets if needed, and decompresses the video stream. When data is lost for any reason, the decompression typically continues at the next available packet. Data loss results in temporary distortions when decompressing the video.

1.3 P2P Video Streaming Networks

In the client-server (CS) architecture, a set of clients connect to a server to obtain a certain service. A computer in such a network is thus either a consumer (client) or a producer (server). In contrast, a peer-to-peer (P2P) network is a set of *peers* (computers) that use their collective upload capacity to provide the service to each other. Each peer obtains the service by forming ad-hoc connections to other peers (its *neighbours*), and is a potential consumer as well as a potential producer. In our case, the service consists of a video stream, which is propagated over the P2P network. The video stream is injected into the network by a specific peer called the *injector*.

A P2P architecture is more complex than the CS architecture, but offers a scalable network at a substanially reduced cost. The available upload capacity in a P2P network is proportional to the number of clients. If the clients provide enough bandwidth, the cost for the injector for providing a video stream becomes constant with respect to the number of clients. In a CS architecture, the available upload capacity is fixed, as all of it has to be provided by the server. The CS architecture can thus only serve a fixed number of clients, and the cost of providing a video stream increases linearly with the number of clients.

Even though the decrease in cost when switching from a CS to a P2P architecture can be significant, there are disadvantages to the P2P architecture as well. The peers

receive the video stream from other peers, which implies that the quality of service depends on the behaviour of the peers. The peers behave less predictably than a server in a CS architecture, leading to several obstacles that need to be taken into consideration:

Free-riding The amount of bandwidth made available by the peers to serve others directly affects the quality of the service that is provided. Peers that cannot or do not provide sufficient upload bandwidth will hurt the P2P network if they consume more resources than they provide. In such cases, it becomes harder for other peers to obtain the video stream at full speed, resulting in loss of quality. If a peer is unwilling to provide upload bandwidth, it is called a *free-rider* [6, 49, 88, 101]. The peers in the Internet are not inherently cooperative, so it is of paramount importance that the peers are given an incentive to provide their upload capacity to others. Low-capacity peers, which have a slow Internet connection, are unable to provide much upload bandwidth regardless of their willingness to share, and can therefore not always be distinguished from free-riders.

Limited Connectivity, caused by software or hardware that blocks incoming connections to certain peers. A firewall running on the machine of the end user or at the gateway of the corporate network can prevent incoming connections from reaching the peers for security reasons. A Network Address Translator (NAT) is also a de-facto firewall, since it can prevent peers on one side of the NAT from being reachable. A NAT is often deployed as part of an ADSL setup and is used by most peers in the Internet [7, 100]. The presence of firewalls limits the connectivity within a P2P network, and therefore limits the flow of video data.

Churn is the rapid arrival and departure of peers [88, 93]. Under churn, the structure of the P2P network is in constant flux as the arriving peers have to be incorporated into the network and the departing peers have to be removed. The connections between the peers appear and disappear as a result. At every change in connections, a peer has to reevaluate from whom it will receive the video stream and to whom it has to send it.

Malicious Behaviour takes place when a peer seeks to actively disrupt the service. Active service disruptions become possible if one peer can easily disrupt the

service provided to others, for instance by injecting false content or by confusing the peers by abusing the protocol used by the P2P network [61, 64, 91]. Other notable attacks include the Sybil attack [34], in which a peer forges many identities (fake peers) to falsely represent a large group of peers to support him, and the Eclipse attack [19, 92], in which one or more peers conspire to dominate the set of neighbours of an honest peer, effectively eclipsing the honest peers from each other's view.

These obstacles are a result of the non-cooperative nature of the environments in which P2P networks are deployed, such as the Internet. A P2P network will thus have to be designed with these obstacles in mind, if it is to be deployed in environments such as the Internet. Not all of the obstacles are necessarily present if the environment can be controlled, and the peers can be forced to behave in a cooperative manner. Examples of such environments include corporate networks and networks of set-top boxes interacting within a controlled distribution network.

1.4 Video Streaming Approaches

The Internet uses the IP protocol to establish connections between pairs of hosts. An extension called IP Multicast has been proposed [29], which supports streaming at the network layer, but this extension has not been widely deployed [32]. The reasons for this include its inherent complexity, its lack of application-level control, and its lack of security [24]. Live video streaming therefore has to take place at the application level, using regular connections between hosts. Performing multicast using regular connections is called application-level multicasting (ALM). The first ALM algorithms focused on video-on-demand [41, 53, 90], for which IP Multicast is not sufficient regardless of its deployment [10, 90]. These algorithms arrange the peers in either a chain [90] or a tree [41, 53], with each non-leaf peer forwarding the stream. The tree structure was also used by the first P2P live video streaming algorithms [13, 16, 21, 24, 30, 95], which started to appear when the deployment issues of IP Multicast became apparent [24].

Tree-based P2P streaming algorithms construct a tree over the peers, with the inner nodes forwarding the stream to one or more peers. The leaves of the tree do not have to forward any data. When a peer arrives, it is incorporated into the tree. When a peer departs, the tree has to be repaired. The peers in the subtrees below a

departed peer will have their reception interrupted until the tree is repaired. The tree structure has two weaknesses, which make its deployment in the Internet difficult. First, the inner nodes in the tree have to forward the stream to several other peers. However, end user Internet connections are often asymmetric, with 4–8 times less upload bandwidth than download bandwidth. A P2P network in the Internet will likely not contain enough peers capable of uploading a video stream several times in parallel. Furthermore, the load on the peers is highly skewed as the leaves cannot put their upload capacity to use. Secondly, churn in P2P networks often causes the multicast tree to be permanently broken. As the tree grows in size, the probability of a service interruption increases for each peer. The tree structure thus does not scale well considering the volatility of the peers in P2P networks. Two separate approaches can be distinguished for trying to alleviate these weaknesses:

Multiple Trees Using a video technique called Multiple Description Coding (MDC) [44], the video stream can be split into several substreams called *descriptions*. A peer will be able to view the video stream with a quality proportional to the number of substreams it receives. MDC makes it possible to create a forest structure, in which multiple trees, one for each description, are constructed originating at the injector and spanning the peers. By making the trees interior-node disjoint, the video streaming algorithm is made less vulnerable to individual peer departures, as only one tree is interrupted at a time. Furthermore, the load on the peers can be balanced as a peer that acts as a leaf in one tree can be used as an interior node in another tree [20, 75].

Swarm Another method of distribution is to split the video stream into pieces of a fixed size. A peer requests each piece from its neighbours. Once it obtains a piece completely, it announces this to its neighbours. A set of peers exchanging pieces of the same video is called a *swarm*, a concept that became popular after the introduction of the BitTorrent file-sharing protocol [26]. The BitTorrent protocol turned out to be easy to implement and provides good performance for offline downloading. The concept of swarms has been extended to live video streaming [12, 48, 58, 76, 103] as well as to video-on-demand [28, 33, 98].

There are several key differences between these two approaches. The swarm-based approach does not require an expensive tree-repairing algorithm, making it easier to implement. On the other hand, the tree-based approach offers a lower and more

controlled latency from the injector to the peers, as data can be forwarded directly by the peers. In the swarm-based approach, each piece first has to be downloaded completely by a peer and subsequently has to be requested by a neighbour, before it can be forwarded.

1.5 Problem Statement

In Section 1.3, we discussed the obstacles that have to be overcome in order to enable video streaming over a P2P network. We believe free-riding to be the most important obstacle when designing a P2P streaming algorithm, because peers that cannot or will not contribute their uplink bandwidth have a direct impact on the quality of the video that can be streamed. The problem of free-riding leads to the following research questions, which we will address in this thesis:

Can free-riding be eliminated in tree-based live video streaming?

In live video streaming, the needs of all peers are symmetrical as they all aim to obtain the most current video data. Every peer thus needs an amount of bandwidth equal to the video bitrate. Free-riding can be eliminated if every peer can somehow be forced to contribute a similar amount of bandwidth to the network in return, which raises the question how an appropriate distribution structure can be constructed and maintained.

Can free-riding be discouraged in swarm-based video-on-demand?

In video-on-demand, the needs of the peers are asymmetric. One peer may have finished downloading the video while the other has just started. As a result, peers cannot always contribute to each other, which makes a forced contribution impossible in many cases. If free-riding cannot be avoided in video-on-demand, the question we raise is whether it is possible to at least discourage it. By encouraging peers not to free-ride, more resources become available in the P2P network, which improves the average quality of service of the peers. Another advantage of not outright banning free-riding is the fact that the bar of entry is lowered. Some peers may be willing but not able to provide much upload bandwidth due to their asymmetric Internet connections. However, care must be taken that such peers cannot hurt the quality of service for the rest of the peers if the resources that are available in the P2P network are a bottleneck.

Can free-riding be discouraged in swarm-based live video streaming?
The presence of asymmetric Internet connections has to be taken into account when deploying a swarm-based live video streaming solution as well. We raise the question whether free-riding can be allowed but discouraged in swarm-based live video streaming in a similar way as for video-on-demand. If so, a unified solution could be possible for both video-on-demand and live video streaming to discourage free-riding using a swarm-based approach, which will ease the implementation and deployment of algorithms for both streaming modes.

How does the lack of connectivity among peers influence their contribution?
Each peer that is willing to provide upload capacity to the P2P network has to find peers to serve. A lack of connectivity in the P2P network thus directly influences how well the available upload capacity in the P2P network can be put to use. The massive deployment of NATs and firewalls limits the connectivity in P2P networks significantly, which raises the question of their effect on the amount of bandwidth each peer will be able to contribute to others.

In all cases, any answer to these questions must take the nature of P2P networks into account as described in Section 1.3. In particular, the proposed P2P streaming algorithms have to be resilient to churn, as well as to malicious peers that try to trick or sabotage the system. To keep the scope of this thesis tractable, we assume that the P2P network provides an identity system to keep track of the individual identities of the peers, which prevents peers from colluding or forging fake identities to subvert the free-riding-resilient video-distribution mechanisms we put in place. Such a reduction in scope is reasonable, because the design of an identity system is orthogonal to the answers that we will derive to our research questions.

1.6 Research Contributions and Thesis Outline

The research we present in this thesis focuses on balancing the bandwidth contributions of peers in P2P streaming networks, both for live video streaming and for video-on-demand. We will consider algorithm design, modelling, analysis, simulation, emulation, as well as the measurement of deployed P2P networks. The body of the thesis covers the following subjects:

Tree-based live video streaming (Chapter 2). We will present an algorithm for live video streaming that forces each peer to contribute as much bandwidth as it consumes. The multiple-tree approach is used, where the video stream is split into several substreams using MDC. A forest is constructed over the peers with a tree for each substream. The performance of the algorithm will be assessed with mathematical analysis, simulations, and emulations. The work in this chapter is published in [67, 68].

Swarm-based video-on-demand (Chapter 3). The tree and multiple-tree approaches are an awkward fit for video-on-demand. Unlike with live video streaming, a tree spanning the nodes cannot be easily arranged since the peers require different parts of the video. In a tree, each peer has to be ahead in the video of its children to be able to provide them with the video data they require. In this chapter, we present a swarm-based algorithm for video-on-demand based on the popular BitTorrent file-sharing algorithm. The algorithm includes an incentive for peers to provide their upload capacity designed for a video-on-demand setting. The work in this chapter is published in [63, 70].

Swarm-based live video streaming (Chapter 4). The swarm-based approach is also feasible for live video streaming. In this chapter, we provide a live video streaming extension to BitTorrent, which together with the algorithm from Chapter 3 allows a single base protocol (BitTorrent) to be used for all three modes of streaming. We study the performance of a globally deployed P2P network using this extension. The work in this chapter has been accepted for publication [66].

Bounds on bandwidth contributions (Chapter 5). We analyse how connectivity problems due to NATs and firewalls place fundamental bounds on the amount of upload capacity that can be put to use by the peers. We compare our bounds to those obtained with simulations, and assess the extent of the connectivity problems in existing P2P networks. The work in this chapter is published in [69].

In Chapter 6, we will conclude this thesis with presenting the insights we have obtained in the problems stated in Section 1.5.

Chapter 2

Tree-based Live Streaming

ALTHOUGH the main purpose of many current peer-to-peer (P2P) networks is off-line file sharing, such networks may also provide suitable environments for multicasting high-quality video streams over the Internet. A multicast solution which aims for deployment on the Internet should take into account the behaviour of peers as observed in existing P2P networks, for example, *free-riding* [88] (peers unwilling to donate resources) and *churn* [88, 93] (peers arriving and departing at a high rate). While multicasting solutions exist which address these problems [22, 50, 72, 75], they either require a central authority or a distributed trust system to deal with free-riders. In this chapter, we propose the Orchard algorithm for building and maintaining multicast trees for video streams in P2P networks, which deals with both free-riding and churn without the need for a central authority or a distributed trust system.

Orchard assumes a video stream to be split up into several substreams, and builds a separate spanning tree for each of these in such a way that in the resulting *forest*, no peer is forced to upload more data than it receives. In general, in an environment which is prone to errors (e.g., due to congestion in the Internet), it is advantageous to split up video streams into multiple substreams and forward these independently. In this chapter we assume the use of Multiple Description Coding (MDC) [44] for this purpose. With MDC, a peer can continue viewing a video as long as it receives at least one substream, with the viewing quality increasing for every additional substream received.

For multicasting the substreams we use Application Level Multicasting (ALM), because multicasting at the network layer has been found to be problematic [24].

However, existing P2P networks suffer from free-riding and churn, which create problems for ALM. In conventional ALM, a single spanning tree is constructed over the nodes (peers) interested in a video stream, and the burden of forwarding the stream is on the interior nodes of the tree. A peer can avoid this burden and free-ride simply by refusing to forward any data. In addition, when a peer fails, the peers in the subtree below it stop receiving the video stream, making the single-tree approach vulnerable to churn. To solve the problem of free-riding, peers have to be given incentives to share their resources. This was recognised in BitTorrent [11], a download protocol in which peers exchange pieces of a file on a tit-for-tat basis. In Orchard, we combine MDC with the bartering spirit of BitTorrent. Orchard's primitives for building the spanning trees enforce that each peer has to forward a substream for every substream it wants to receive (with some minor exceptions). In the resulting forest, no peer has to forward more data than it receives, which protects the system against free-riders, and, by using multiple trees, against churn.

This chapter is organised as follows. In Section 2.1, we further specify the problem we address. We present the Orchard algorithm in Section 2.2. In Section 2.3, we discuss how Orchard prevents free-riding and how it behaves under several other well-known types of attack. An analysis of the expected performance is presented in Section 2.4. Our experimental results are presented in Section 2.5. We discuss some related work in Section 2.6. Finally, we discuss our approach, and draw conclusions in Sections 2.7 and 2.8, respectively.

2.1 Problem Description

We will now specify the problem of ALM of video streams split up into multiple substreams in P2P networks. First, we will describe the required pre-processing of video streams. Next, we will present the assumptions we make about the underlying P2P network, and finally, we will give our problem statement.

2.1.1 Stream Pre-processing

Before a video stream is distributed, it is split up into some number of substreams called *descriptions* with the technique of Multiple Description Coding (MDC) [44]. We will assume that these descriptions require equal bandwidths (with C descriptions, a fraction of about $1/C$ of the bandwidth of the original stream), and that they are of

equal value regarding their contribution to the viewing quality. With MDC, a peer will be able to play a video as long as it receives at least any one of the descriptions. For ease of discussion, each description will be represented by a *colour* (not to be confused with a colour in images of the video stream). Splitting the video stream adds overhead to the data stream as well as complexity to the encoder and decoder. This overhead highly depends on the video content, the video encoding, and the number of descriptions used [38]. In our experiments, the original video stream is split up into four MDC descriptions, which is a reasonable compromise. If too few descriptions are used, the loss of one description causes a big drop in quality. If too many descriptions are used, the overhead is too large.

2.1.2 The Underlying Peer-to-Peer Network

We assume the existence of a P2P network in which every peer can connect to every other peer. Every peer has sufficient incoming as well as outgoing bandwidth to receive and send all descriptions simultaneously. This implies that peers which cannot contribute as much as they consume are barred from receiving the video stream. The bandwidth bottlenecks are assumed to be at the end-user, not in the core network.

When a peer becomes interested in a video stream, it will try to contact other peers who are already receiving one or more descriptions. For this purpose, we assume that interested peers maintain a *neighbour set* of other interested peers, which are selected at random. The neighbour relation is symmetric: if a is a neighbour of b, b is also a neighbour of a. A peer is assumed to be able to keep his neighbour set populated with at least m peers. When a neighbour departs and a peer has $m-1$ neighbours left, a new neighbour is added to bring the number of neighbours back to m. In our experiments, we use $m = 20$.

Mechanisms for discovering other interested peers are outside the scope of this chapter. A possible way to do this is by using epidemic information dissemination [8, 35]. Another way is presented in [37], in which information is pushed to a random subset of peers in the underlying P2P network, and peers looking for that information query a random subset of peers. This method provides a high probability of interested peers discovering peers which receive the stream. In our experiments, for convenience, we use a central server for peer discovery (see Section 2.5.1).

2.1.3 Problem Statement

We suppose that there is a special peer s (the *source*) that has a video stream available for multicasting. We want to design an algorithm that creates a set of multicast trees rooted at s, one for every MDC description, with the peers interested in the video stream as the nodes in all these trees.

We focus on data dissemination and stay codec and content agnostic. What constitutes acceptable frame loss and stream latency highly depends on the codec as well as the content, so we will refrain ourselves from using them. Instead, the main performance metric we will use to assess our algorithm is the (average) number of descriptions received by the peers.

We want our algorithm to satisfy the following conditions:

1. *Fairness:* No peer forwards more descriptions than it receives.
2. *Decentralisation:* A peer does not need to have s as one of its neighbours.
3. *Resilience:* Peer arrivals and departures do not severely impact the number of descriptions received by other peers.

2.2 The Orchard Algorithm

In this section, we will give a detailed description of the Orchard algorithm for constructing a forest of multicast trees for distributing video streams from a single source. First, we will present the forest-building primitives of Orchard. Then we show how the forest can be repaired when peers depart. Finally, we will present some observations on the structure of the trees in Orchard.

2.2.1 Constructing the Forest

An Orchard forest for a source s is represented by a directed graph with a set of nodes (peers) P and a set of links L. Each node $a \in P$ represents a peer, and each link $e = (a,b) \in L$ represents the forwarding of a description from peer a to peer b, with $a,b \in P$. Every link and every peer is assigned a single colour. Each link in L is given the colour of the description sent over it. The subset of links of a certain colour and the peers they connect form the multicast tree for a single description. All the trees are rooted at the source s. Each peer gets the colour of the first description it receives.

As long as a peer is not part of any tree, it will be considered to have the special colour *blank*. The source always has the special colour *white*. Each peer will be allowed to forward any colour it receives, and we assume that each peer will strive to receive all colours.

When a peer p wants to join the Orchard forest, it will try to become a member of all multicast trees. In order to do so, p will query every peer q in its neighbour set to see whether q is willing to strike a *deal* with it until it has joined every tree or until its neighbour set is exhausted. The multicast forest is constructed using three types of deals:

1. *Join at the source* is a deal a peer can strike with the source. The source forwards one description to each of the first few peers that arrive in the system.

2. *Exchange descriptions* is a deal in which peers exchange descriptions in a pairwise fashion in order to obtain more descriptions.

3. *Redirection* is a deal in which a neighbour of a peer p redirects one of its outgoing links through p in order to have p obtain more descriptions. We will define two variations of this type of deal depending on whether p was still blank before the deal or not.

As we will see in the next section, the mechanism of deals ensures that peers contribute as much outgoing bandwidth as they consume incoming bandwidth, with the exception of the join-at-the-source deals: The source is willing to forward descriptions without expecting anything in return, and a few peers will be so lucky as to get a description for free from the source. The source makes sure it forwards every description at least once. A deal will exist as long as both parties keep their part of the deal, and each peer keeps track of the deals it has with other peers.

To strike these deals, peers exchange control information and send deal requests. Every peer exchanges updates with its neighbours about its colour and the colours it receives. If a peer receives a deal request, it will answer whether it will accept the deal. If a deal is accepted, information required by the requesting peer, such as what colour needs to be forwarded to whom, is sent along with the accept message.

2.2.2 Primitives to Build the Trees

In this section we will describe each of the three types of deals in more detail. To obtain its first colour, a peer has to strike either a join-at-the-source deal or a redi-

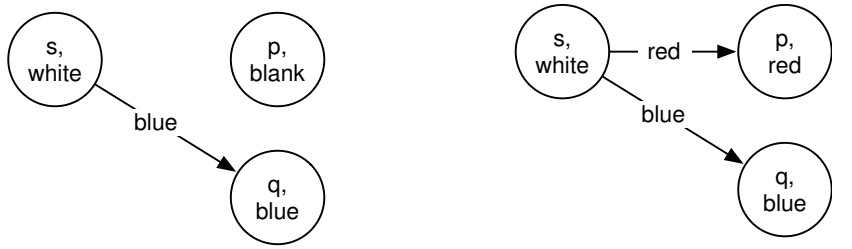


Figure 2.1: Joining at the source: before and after peer p joins at the source s.

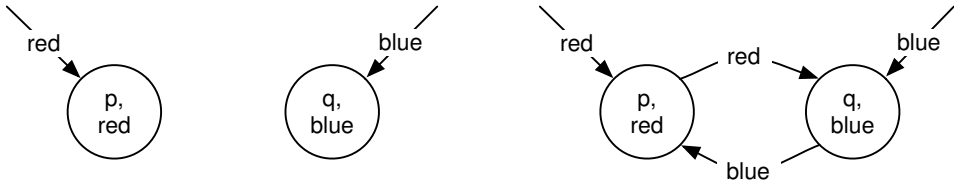


Figure 2.2: Exchanging descriptions: before and after two peers strike an exchange deal.

rection deal. For every additional colour, a peer has to strike an exchange deal or a redirection deal. To distinguish whether a peer strikes a redirection deal for its first or for additional colours, we will refer to the latter as *redirection through coloured peers*. Every time a peer looks to strike a deal, it orders its neighbours on increasing hop-count, preferring to strike a deal with a neighbour close to the source.

Join at the Source

If a peer p has the source in its neighbour set, and the source has spare outbound capacity, the source will forward one of the descriptions to p and the peer joins the corresponding tree at the source. The source will favour sending a description to p that it is not yet forwarding to any other peer. Peer p will then get the colour of that description. An example of this is shown in Figure 2.1; here the source s decides to forward the red description to p. The source will refuse to forward more than one description to the same peer.

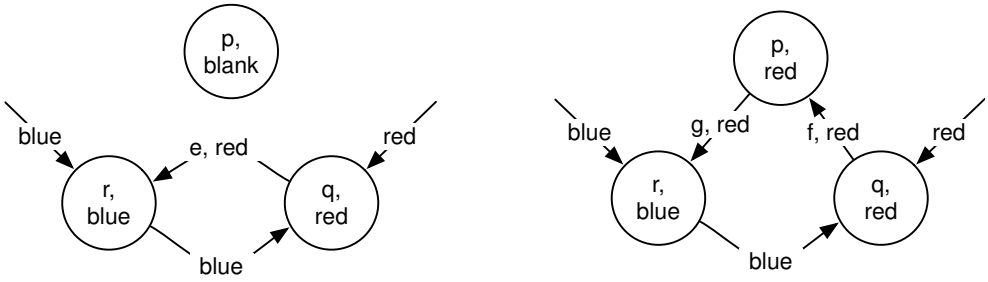


Figure 2.3: Redirection: before and after peer p joins the multicast tree below q.

Exchange Descriptions

To obtain a description it is not yet receiving, a peer p checks its neighbour set for a peer q such that p does not receive q's colour and q does not receive p's colour. When p has found such a peer q, they strike an *exchange deal*: peer p forwards its colour to q and vice versa, as shown in Figure 2.2.

Redirection

If a peer p cannot join any tree at the source and is still blank, it cannot exchange descriptions because it does not receive any. It will then ask each of the non-blank peers in its neighbour set whether it can redirect through p one of the streams it is forwarding, as is shown in Figure 2.3. Here, peer q strikes a *redirection* deal with peer p, which replaces the link e with two links $f = (q, p)$ and $g = (p, r)$ of the same colour as e. This way, q does not require additional bandwidth, and p is asked to receive the colour and also to forward it.

Repeatedly redirecting a link $e = (q, r)$ would create a chain of peers, which is undesirable — a chain of peers between q and r would result in an increased latency as well as an increased chance of failure along the chain. To avoid this, only links which were created by an exchange deal can be redirected. So in Figure 2.3, peer q cannot redirect link f, because it does not have an exchange deal with p. The same holds for p and link g.

Once a peer p has struck a redirection deal, it receives and forwards a description, and it gets the colour of that description. Even though p has selected the peer closest to the source that was willing to strike a redirection deal, it is possible that p will later encounter another peer t of the same colour that is even closer to the source. In

that case, p switches parents by breaking the redirection deal with q and striking a redirection deal with t.

Redirection through Coloured Peers

The algorithm so far makes peers depend on exchange deals to obtain additional colours. However, not all peers will be able to obtain every additional colour this way. Because a peer may not find appropriate neighbours to strike exchange deals with.

This problem could be solved by letting every peer connect to a large set (for example, 100) of neighbours, but this is undesirable from a practical point of view: if a peer has to connect to and query many neighbours in order to obtain colours, it takes longer to obtain them, especially if connecting involves DNS lookups and getting past firewalls. Also note that a large neighbour set requires more maintenance, as the chance increases that some neighbours will depart. For these reasons, Orchard has another way for peers to obtain additional colours: we also allow *redirections through coloured peers*. This results in the transition shown in Figure 2.4, which is similar to the one shown in Figure 2.3, except that p already has a colour before striking the deal with q, which is different from q's colour and which it retains after the deal.

This relaxation is provided as a backup system to allow peers to obtain all colours fast without having to query many neighbours, but makes the coloured peers compete with the blank peers for redirection deals. To prevent this, peers will break a redirection deal through a coloured peer if it can be replaced with a redirection through a blank peer. Because coloured peers know this can happen, they will give priority to striking exchange deals.

2.2.3 The Resulting Trees

We will now argue that in each tree, a peer forwards its own colour to at most $C-1$ other peers, where C is the number of colours in the system. By construction of Orchard, the out-degree of every peer is at most C. There are two ways in which a peer p can obtain its first colour. Either p joins at the source, or it strikes a redirection deal for its first colour. In the first case, p receives its colour for free. In the second case, p joins through a redirection deal, and as part of this deal has to forward its own colour to a peer of another colour (see Figure 2.3). In both cases, p receives at least one colour without forwarding anything to a peer of the same colour in return, which

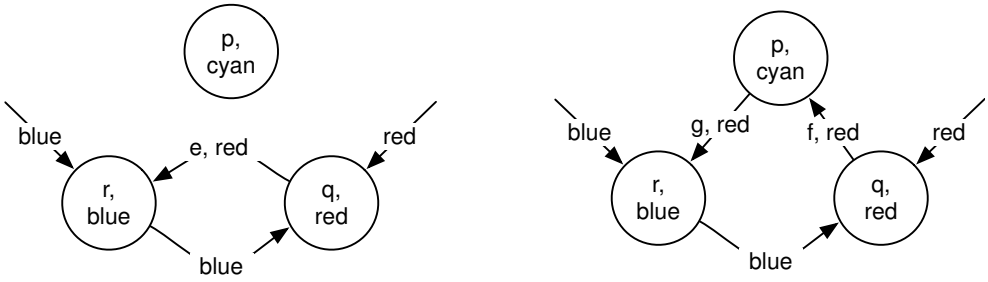


Figure 2.4: Redirection through coloured peers: before and after peer p joins the multicast tree below q.

proves our assertion. Because a peer has at most $C-1$ children of the same colour, the system needs to contain at least three colours, for otherwise all trees are linear chains.

Let us take the colour red as an example to further analyse the structure of the trees in Orchard. Every red peer gets its own colour either from the source (see Figure 2.1), or from another red peer (see Figure 2.3). The non-red peers that receive the colour red are either leaves (see Figure 2.2), or peers that forward red to precisely one leaf (see Figure 2.4). In other words, the 'core' of the red tree consists of red peers and the source, while the leaves, and in some cases peers one level above the leaves, are peers of other colours. As a consequence, if a single red peer p fails, most peers in the subtrees below p stop receiving at most one colour (red) until the forest is repaired.

Because peers prefer to join close to the source, the trees created are shallow, which is desirable because of reduced latency and packet loss, but also because on average, nodes in a shallower tree have smaller subtrees below them. To see this, take a full w-ary tree of n nodes. The height of the tree can be expressed as

$$h(n) \quad = \quad \log_w(1+(w-1)n),$$

and every node at distance d from the root is a descendent of d nodes. Let $c(n)$ be the average number of children of all nodes. Then,

$$c(n) \quad = \quad \frac{1}{n}\sum_{d=1}^{h(n)-1} dw^d.$$

Multiplying both sides with $(w-1)n$ leads to

$$\begin{aligned}
(w-1)nc(n) &= \sum_{d=1}^{h(n)-1} dw^{d+1} - \sum_{d=1}^{h(n)-1} dw^d \\
&= (h(n)-1)w^{h(n)} - \sum_{d=1}^{h(n)-1} w^d \\
&= (h(n)-1)w^{h(n)} - \left(\frac{w^{h(n)}-1}{w-1} - 1\right) \\
&= (h(n)-1)(1+(w-1)n) - \frac{1+(w-1)n-1}{w-1} + 1 \\
&= h(n) + (h(n)-1)(w-1)n - n,
\end{aligned}$$

which results in

$$c(n) = h(n) + \frac{h(n)/n - w}{w-1}, \tag{2.1}$$

which is of order $\log_w n$. So, $c(n)$ decreases when w increases.

2.2.4 Repairing Trees

When a peer p departs or fails, other peers will stop receiving the descriptions they get from p. Any peer that has a deal with p will cancel that deal. For example, in Figure 2.3 on the right, if peer p fails, the redirection deal with q is cancelled. This causes q to restore the exchange deal with r as it was before it got redirected through p.

If the deal to be cancelled is an exchange deal, the redirection deals dependent on it (there can be at most two) have to be cancelled. An example of this is shown in Figure 2.5. Here, if peer r departs, q will break its exchange deal with r, which forces q to also break its redirection deal with p. This causes p to stop receiving its colour, and makes p unable to maintain any exchange deals with other peers. Even if p is still receiving other colours, we then force p to break all its deals and become blank.

These breaks of deals propagate down the subtrees below each peer that stops receiving its own colour. As is shown in Equation 2.1, a peer has on average $\mathcal{O}(\log n)$ children below it in the tree of its colour. If a peer departs and no measures are taken, the peers of the same colour below it will stop receiving their colour, which

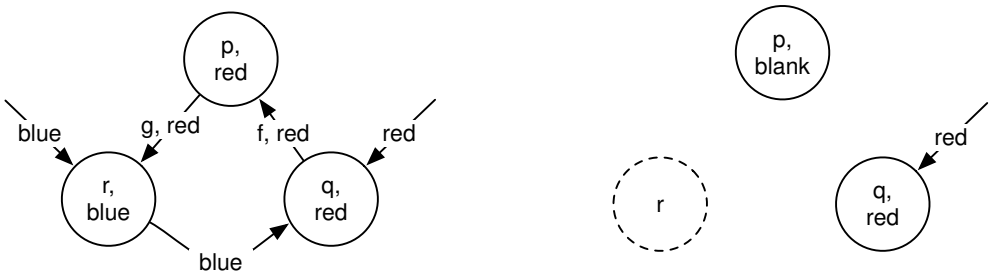


Figure 2.5: Peer departure: before and after peer *r* departs.

forces them to break most of their deals (their exchange and their redirect deals). The other peers with which these deals were struck will stop receiving only one of their colours. In order to prevent the propagation of deal breaking, we will now discuss three methods of maintaining the trees.

Backup Parents

In order for a peer to be able to quickly repair the loss of its parent of the same colour, each peer keeps track of the backup parents in its neighbour set. A peer q is a *backup parent* of p, if it is of the same colour and the path from the source to q in that colour does not contain p's parent.

When the parent of p fails, p asks each of its backup parents, in order of increasing hop count from the source, whether it can strike a redirection deal. If p happens to have the source as a backup parent, it will first try to join at the source. Peer p will join the tree at the first backup parent that allows it. If no backup parent allows p to join, p will break all its deals, become blank, and try to rejoin the system.

Because no descendant of the departed parent is allowed to be a backup parent, no cycles are created in the multicast tree. To satisfy this condition, every peer keeps its neighbours informed about its paths to the source. Since each peer only has to be able to compare paths, it is enough to send only the hashes of the identities of the peers along the paths. The use of hashes preserves their anonimity, which prevents peers from directly contacting the source or the peers close to it.

Peers Changing their Colour

Unfortunately, the backup parent strategy is not enough. If a peer of colour c near the root of a tree fails and many peers below it cannot find a backup parent, those peers are all forced to rejoin the system and obtain a different colour. The colour c then becomes more rare, and it is possible that at some point not all peers will be able to receive it. In the worst case, the description could entirely disappear from the system.

To counter this, every peer keeps track of the set of all the colours its neighbours are receiving. Once this set stabilises, a peer checks whether switching to a rarer colour is beneficial. This is the case if it will be able to receive the rare colour, and will be able to strike more exchange deals than it could before switching. Once a peer switches colours, it breaks the deals with those peers that are not willing to accept the colour change, and strikes new deals with peers that are willing to. In our experiments, the neighbour set is considered stable if they colours they receive does not change for three seconds.

Neighbour Set Maintenance

When too many peers become blank, there may be peers which have too many blank neighbours to obtain all colours. To counter this system degeneration, a peer removes a neighbour from its neighbour set if that neighbour stays blank for a certain length of time, which, in our experiments, is three seconds. Note that since we assume that each peer can keep its neighbour set populated, the neighbour set will not become depleted if peers are removed.

2.3 Attacking Orchard

In this section, we state explicitly in what way Orchard prevents free-riding. In addition, we discuss to what extend Orchard is resilient to several other types of attack.

2.3.1 Free-riding

In this chapter, we consider a peer to be a free-rider if it provides fewer resources for forwarding the video stream than it consumes. Because the source forwards data for free, there must be peers which receive data for free, and these peers can thus be said

to free-ride to some degree. In Orchard, the peers which have struck a join-at-the-source deal are those which receive data for free. However, the effect of this is limited as the source forwards every colour to a limited set of peers, and forwards at most one colour to any peer.

To obtain a colour from a peer other than the source, a peer always has to forward a colour in return, either to the same peer (exchange deal) or to a different peer (redirection deal). This forces peers to contribute if they want to obtain any colour from another peer, making it impossible for them to free-ride.

2.3.2 Other Types of Attacks

A well-known method of attacking P2P networks is by taking control of multiple identities, either by a single peer emulating them (Sybil attack [34]) or by cooperation amongst peers (collusion attack). In Orchard, the exchange and redirection deals function regardless of whether the peers cooperate. Due to the nature of these deals, all peers, except those joining at the source, will forward as much data as they receive. This upload-download balance is always maintained for any peer and thus for any subset of peers as well. Both types of attack thus have no impact on either type of deal. The join-at-the-source deal is different, because the source grants only one such deal to any peer. A peer which can fake multiple identities can thus fool the source and potentially obtain multiple colours for free. In systems where this is a problem, the source can require the peers with which it strikes a join-at-the-source deal to send a stream with random data of the same size back to it. Although this wastes resources, it removes any bandwidth advantage to join at the source.

The source is, by definition, a single point of failure in any multicast algorithm, but fortunately, in Orchard peers do not need to know the source to obtain the video stream. The probability that a new peer meets the source thus decreases when the number of receivers increases.

Many more types of attacks exist in P2P networks. Peers could enter the system purely for malicious purposes, with no interest in the video stream itself. A malicious peer could for example offer fake deals or forward fake data. For an attack to be most effective, its effects have to propagate down the trees below its victim. To do this, it has to convince a victim peer to switch parents. However, when a victim peer detects it is receiving no data or fake data, it will disconnect from the malicious peer. If the victim peer cannot repair in time, its children will also detect the absence of data, and

will look for a different parent themselves. Since peers will typically already have a backup parent to switch to, the attack can be quarantined quickly. Additionally, peers could employ a policy of making the swap of parents definitive only when verified data starts flowing.

Of course, a sufficiently powerful malicious peer can attack enough peers in the system as to render it unusable. In such cases, the help of the underlying P2P network is needed to be able to shut out such peers.

2.4 Expected Performance

In this section, we will analyse the expected performance of the Orchard algorithm. As a performance metric, we will use the average number of colours received by a peer. First, we will describe the parameters used and make some general observations. Then, we will analyse the expected number of exchange deals in the system as peers arrive and depart, as well as the expected number of redirection deals, for both variants.

2.4.1 Parameters of the Model

We define two types of events: peer arrivals and peer departures. Furthermore, we will assume an event is completely processed before the next event occurs. We will use the following parameters in our model:

- N is the number of peers in the system,
- C is the number of descriptions (colours),
- m is the size of the neighbour set a peer obtains upon arrival,
- n_i is the fraction of the peers that are of colour i,
- e_{ij} is the fraction of the peers that are of colour j and have an exchange deal for colour i ($e_{ii} = 0$ and $e_{ij} = e_{ji}$),
- $\alpha_i = \sum_{j=1}^{C} e_{ij}$ is the fraction of the peers that have an exchange deal for colour i,
- r_i is the fraction of the peers that are not of colour i but have colour i redirected through them,

- $\rho_i = n_i + \alpha_i + r_i$ is the fraction of the peers that receive colour i.

These parameters represent the situation before an event occurs, and primed versions of the parameters (n'_1, etc.) will describe the situation in the system after an event has been processed. For instance, when a peer arrives, $N' = N + 1$, and when a peer leaves, $N' = N - 1$.

As a metric, we will use $R = \sum_{i=1}^{C} \rho_i$, which is the number of colours the peers receive on average. We will assume that the system is *stable*. We define a system to be stable when (for large N) the expected values of the parameters n_i, e_{ij}, α_i and r_i (and so, also ρ_i and R) do not change under peer arrivals and departures. System stability with respect to the fractions of peers of the same colour ($E[n'_i] = n_i$) is found to occur in our experiments, which even seem to indicate that the distribution of colours over the peers is almost uniform (see Figure 2.10). For settings in which the distribution of colours nevertheless becomes non-uniform, peers could keep each other informed about the colour distribution using gossip-based protocols [55]. An arriving peer can then decide its colour based on more than its local observations, rebalancing the system.

The performance in a stable system depends on the arrival and departure patterns of the peers. We shall analyse two extremes: a system in which peers only arrive, and a system in which peers only depart. For both scenarios, we will analyse for which values of e_{ij} and r_i the system is stable.

2.4.2 Peer Arrivals and Exchange Deals

A peer p that arrives and becomes of colour i (by joining at the source or by a redirection deal) will try to obtain additional colours by striking exchange deals with its neighbours. The probability of p becoming of colour i is equal to n_i, because we analyse a stable system with respect to the colour distribution. Let us consider colour $j \neq i$. If p has a neighbour which is both of colour j and does not have an exchange deal for colour i yet, an exchange deal will be struck. Since a random neighbour has probability $n_j - e_{ij}$ of falling into that category, p will have a probability of $1 - (1 - n_j + e_{ij})^m$ of striking an exchange deal for colour j. Note that this probability depends only on n_j and e_{ij}, not on the distributions of the other colours and exchange deals.

Now, consider e'_{ij}. The number of exchange deals between colours i and j can increase if a peer arrives and becomes of colour i or j. The probability that an arriving peer becomes of colour i and is able to strike an exchange deal for colour j is equal to

$n_i(1-(1-n_j+e_{ij})^m)$. Let $f_{ij}(e_{ij})$ be the expected increase if there are $e_{ij}N$ exchange deals between peers of colours i and j already. Then, the following holds:

$$f_{ij}(e_{ij}) \quad = \quad n_i(1-(1-n_j+e_{ij})^m)+n_j(1-(1-n_i+e_{ij})^m).$$

Because $E[e'_{ij}N'] = e_{ij}N + f_{ij}(e_{ij})$ with $N' = N+1$, we have

$$E[e'_{ij}] \quad = \quad e_{ij} + \frac{f_{ij}(e_{ij}) - e_{ij}}{N+1}.$$

Note that e_{ij} is bounded by $0 \leq e_{ij} \leq \min\{n_i, n_j\}$, because there cannot be more exchange deals than peers of colour i or j. Let $\overline{e_{ij}} = \min\{n_i, n_j\}$.

In a stable system, $E[e'_{ij}] = e_{ij}$, which holds if $f_{ij}(e_{ij}) = e_{ij}$. The value of e_{ij} for which this equation holds is unique, because f_{ij} decreases from $f_{ij}(0) \geq 0$ to $f_{ij}(\overline{e_{ij}}) \leq \overline{e_{ij}}$.

This allows us to calculate $\alpha_i = \sum_{j=1}^{C} e_{ij}$, which is the fraction of all peers obtaining colour i through an exchange deal. Furthermore, we can calculate $\sum_{i=1}^{C} \alpha_i$, which is the average number of exchange deals struck by a peer. In Figure 2.6, we plot this average using dashed lines, for various values of n_1 and m. In all cases, $C = 4$ colours are assumed, with $n_2 = n_3 = n_4$ and $\sum_{i=1}^{C} n_i = 1$ (that is, there are no blank peers). Note that a peer can strike at most $C-1$ exchange deals, as the first colour is obtained by either a redirection or a join-at-the-source deal.

We draw three conclusions from Figure 2.6. First, the optimum is clearly attained when all colours are uniformly distributed ($n_1 = 0.25$). Second, having more neighbours per peer indeed increases performance, but with diminishing returns. Third, the exchange deals enable peers to receive a rather high number of colours, but there is still some room for improvement, as we will see in Section 2.4.5.

2.4.3 Peer Departures and Exchange Deals

In this section we consider the effect of peer departures on the fractions of peers with exchange deals in the system. In this section, we will assume peers are only departing, not arriving.

Every peer p maintains a neighbour set and makes sure it contains at least m peers (see Section 2.1.2). If a neighbour departs and p has only $m-1$ neighbours left, it is replaced by a new one selected at random from the peers still in the system. We will assume that eventually, with only peer departures and no arrivals, every peer has

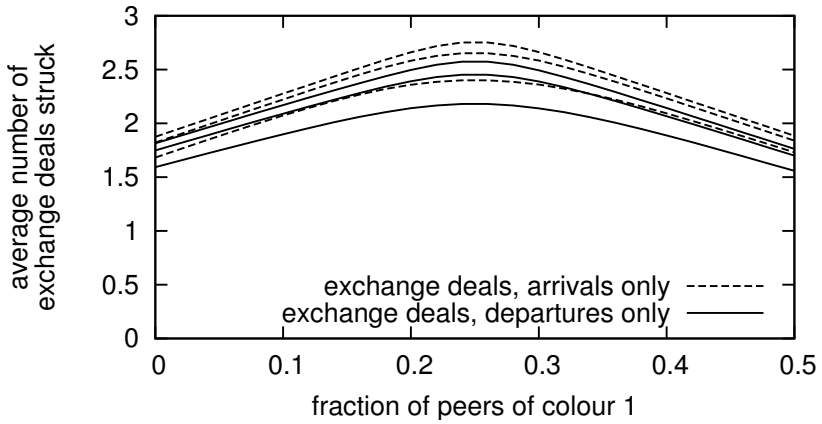


Figure 2.6: The expected number of exchange deals struck in a system with 4 colours for various numbers of neighbours m ($n_2 = n_3 = n_4$). The three graphs in either data set represent, from bottom to top, 10, 20, and 30 neighbours.

exactly m neighbours.

When a peer p departs, its children of the same colour will try to repair the tree by contacting backup parents. Those who do not succeed in obtaining a new parent this way will drop their colour and rejoin the system; this can be considered (for analysis purposes) as a peer departure followed by a peer arrival.

We will again consider the fraction e_{ij} when a random peer p departs. There are two possibilities:

1. p is of colour j and has an exchange deal for colour i, or vice-versa. This is the case with probability $e_{ij} + e_{ji} = 2e_{ij}$.

2. Otherwise.

In case 1, which has probability $2e_{ij}$, an exchange deal between colours i and j is lost. Let q be the neighbour with which p had this deal. Then, q will obtain a new neighbour to replace p, and will attempt to strike an exchange deal with any of its neighbours, which succeeds with probability $f_{ij}(e_{ij})$.

In addition to q, there are $m-1$ other peers which had p as a neighbour, and which will replace p with a new neighbour. If such a peer is of colour i or j and does not have an exchange deal yet for the other colour, it will try to strike one with its newly

acquired neighbour. For each of the $m-1$ peers, this happens with a probability of

$$\begin{aligned} v_{ij} &= (n_i - e_{ji})(n_j - e_{ij}) + (n_j - e_{ij})(n_j - e_{ji}) \\ &= 2(n_i - e_{ij})(n_j - e_{ij}). \end{aligned}$$

This is an approximation, because the neighbour sets may intersect, but for large N, the error will be small.

In case 2, which has probability $1-2e_{ij}$, there is no exchange deal between colours i and j lost when p departs, and there are m peers which had p as a neighbour. For each of these peers, there is again a probability of v_{ij} that they have no exchange deal between colour i and j but will strike one with their new neighbour. To sum up, the expectation of the new number of exchange deals is approximately

$$\begin{aligned} E[e'_{ij}N'] &\approx e_{ij}N - 2e_{ij}(1 - f_{ij}(e_{ij})) + 2e_{ij}(m-1)v_{ij} \\ &\quad + (1-2e_{ij})mv_{ij} \\ &= e_{ij}N - 2e_{ij}(1 - f_{ij}(e_{ij})) + (m-2e_{ij})v_{ij}, \end{aligned}$$

with $N' = N-1$.

The system is stable if $E[e'_{ij}] = e_{ij}$, and it can again be shown that e_{ij} is uniquely determined. This implies that $\sum_{i=1}^{C} \alpha_i$, which is the average number of exchange deals struck by a peer, is also uniquely determined. In Figure 2.6, we plot this average using solid lines for various values of n_1 and m. Again, $C = 4$ colours are assumed, with $n_2 = n_3 = n_4$ and $\sum_{i=1}^{C} n_i = 1$. The performance is slightly lower than in the previous section, where peers were only arriving.

2.4.4 Redirection

We will now analyse the probability that an arriving peer will obtain its first colour. The probability of a peer joining at the source is small if N is large, so we will focus on a peer joining the forest by striking a redirection deal.

Redirection deals take precedence over redirection deals through coloured peers, so we can safely ignore the latter in this section. For an arriving peer p to strike a redirection deal, it needs a neighbour which has an exchange deal which is not yet redirected (*free*). There are $\frac{1}{2}\sum_{i=1}^{C} \alpha_i N$ exchange deals, which provide $\sum_{i=1}^{C} \alpha_i N$ potential redirections. Of these, $\sum_{i=1}^{C} n_i N$ are redirected to give peers their first colour. This leaves $\sum_{i=1}^{C} (\alpha_i - n_i)N$ potential redirections available when p arrives. The distri-

bution of these is skewed – peers prefer to join a tree as close to the source as possible, so peers close to the source are less likely to have any free exchange deals left. Since a peer can have at most $C-1$ exchange deals, the Pigeonhole Principle ensures us that at least $\sum_{i=1}^{C}(\alpha_i - n_i)N/(C-1)$ peers have at least one free exchange deal regardless of the distribution of these deals over the peers. The probability of p having a neighbour which has a free exchange deal is at least

$$1 - \left(1 - \frac{\sum_{i=1}^{C} \alpha_i - n_i}{C-1}\right)^m.$$

As an example, when using $m = 20$ and the same conditions as before ($C = 4$, $n_2 = n_3 = n_4$, and $\sum_{i=1}^{C} n_i = 1$), the probability of p striking a redirection deal upon arrival is higher than 0.99 for $0 \leq n_1 \leq 0.55$.

2.4.5 Redirection through Coloured Peers

If a peer p cannot obtain a certain colour by striking an exchange deal, it will try to strike a redirection deal for that colour instead. However, unlike exchange deals, such redirection deals can be broken in the future even if peers do not depart. For instance, p drops an exchange deal in favour of an exchange deal it can strike when it is contacted by arriving peers. For this reason, we will look at the system at a single point in time and analyse the expected number of redirection deals for a certain colour. The following analysis will provide a lower bound on this expectation.

Let us consider colour i. Of the N peers in the system, $n_i N$ are of colour i, and $\alpha_i N$ receive colour i through an exchange deal. Let D_i be the rest of the peers, with $|D_i| = (1 - n_i - \alpha_i)N$. These peers try to strike a redirection deal with their neighbours. We will approximate the fraction of peers in D_i that is expected to succeed. To do this, we start with a system without redirection deals through coloured peers for colour i. Then, we consider the peers in D_i in a random order and let them try to strike a redirection deal with their neighbours.

For a peer in D_i to strike a redirection deal for colour i, it needs a neighbour of colour i with a free exchange deal. By using the same argument as in the previous section, but focusing on colour i, it can be shown that there are at least $(\alpha_i - n_i)N/(C-1)$ such peers if there would be no redirection deals through coloured peers for colour i.

Now we consider each peer in D_i in turn, in a random order, and analyse the

probability it will strike a redirection deal. Let y_k be the fraction of the k peers that have struck a redirection deal for colour i when k peers have been considered. Then, at least $F_i(y_k) = ((\alpha_i - n_i)N - y_k k)/(C-1)$ peers have at least one free exchange deal. We will focus on this worst case and use the results as a lower bound on the expected performance in general. We have

$$E[y_{k+1}(k+1)] \quad = \quad y_k k + 1 - \left(1 - \frac{F_i(y_k)}{N}\right)^m,$$

and by defining $G_i(y_k) = 1 - \left(1 - \frac{F_i(y_k)}{N}\right)^m$, we have

$$E[y_{k+1}] = y_k + \frac{G_i(y_k) - y_k}{k+1}.$$

Now, let γ_i be the solution to $G_i(x) = x$ with $0 \leq \gamma_i \leq 1$. Because $G_i(x)$ decreases over that interval, γ_i is uniquely defined.

We can now bound $E[y_{|D_i|}]$ as follows. Every time a redirection deal is struck, the probability for subsequent peers to strike one decreases. This leads to $E[y_{k+1}] \leq E[y_k]$. In order for that to hold, $G_i(y_k) \leq y_k$ and so $y_k \geq \gamma_i$. The same holds for $y_{|D_i|}$, so we expect to have $y_{|D_i|} \cdot |D_i| \geq \gamma_i \cdot |D_i|$ redirection deals through coloured peers for colour i. This translates to

$$\begin{aligned} r_i &= y_{|D_i|} \cdot |D_i|/N \\ &\geq \gamma_i \cdot |D_i|/N \\ &= (1 - \alpha_i - n_i)\gamma_i \end{aligned}$$

as a lower bound on r_i.

In Figure 2.7 we show the performance of a system with and without redirection deals through coloured peers, in a stable system with only peer arrivals or only peer departures. As in the previous figure, $C = 4$ colours are used, $n_2 = n_3 = n_4$, and there are no blank peers, so $\sum_{i=1}^{C} n_i = 1$. Every peer thus receives at least one colour by definition. Figure 2.7 only plots the number of additional colours obtained.

The upper two lines indicate the total expected number of deals struck by a peer to obtain additional colours (equal to $\sum_{i=1}^{C}(\alpha_i + r_i) = R - 1$), when peers are only arriving, and when peers are only departing. The lower two lines indicate the performance when redirection deals are not considered. In both cases, the advantage over having only exchange deals is clear. In the following section, we will compare these bounds

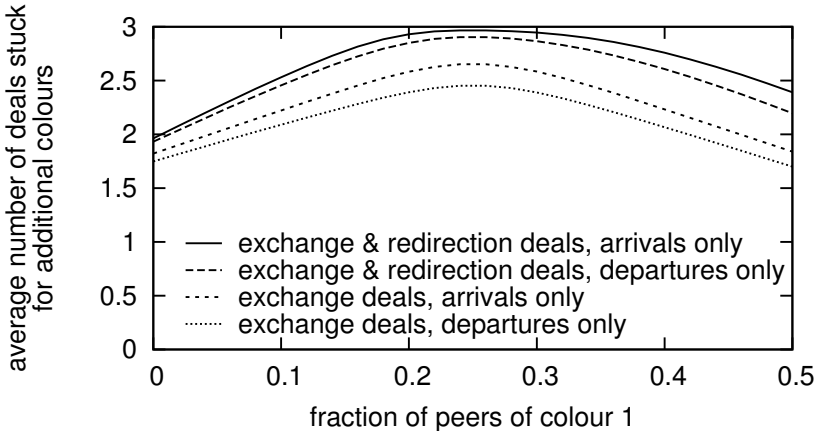


Figure 2.7: The expected number of deals struck to obtain additional colours in a system with only peer arrivals or only peer departures. Every peer maintains 20 neighbours ($C = 4$, $n_2 = n_3 = n_4$).

to measured results.

2.5 Experiments

In this section we present our experimental setup and the results of five experiments for assessing the multicast forest constructed by Orchard. We conclude this section with a discussion about possible issues when the systems contains a larger number of peers than we employed in our experiments.

2.5.1 Experimental Setup

We have tested the Orchard algorithm in two emulation environments. The first environment is the DAS2 cluster computer [1], using 20 processing nodes. The second environment is a wide-area network which we refer to as Delft-37, which consists of hosting accounts at various ISPs around the globe. These accounts range from web accounts to virtual private servers. All Delft-37 accounts have limited resources as they have to share the same machines with other accounts. The reason for creating Delft-37 rather than using PlanetLab [25] is a closer resemblance to real peers on the Internet. The nodes in PlanetLab generally have better hardware and Internet connections than the average end system on the Internet. In our experiments, we use 7

Delft-37 node	# of peers
AZ: USA (Arizona)	9
NY: USA (New York)	14
WA: USA (Washington)	11
GE: USA (Georgia)	5
UK: United Kingdom	8
D: Germany	15
SG: Singapore	4

Table 2.1: The number of peers emulated in Delft-37.

nodes of Delft-37 located in the USA, the UK, Germany and Singapore (see Table 2.1). The bandwidth between these nodes was 20–600 kbit/s, and the latency between these nodes varied between 30–400 ms.

We have implemented the Orchard algorithm in Python. The emulator is started on every host computer, and emulates multiple peers on every one of them. Every peer sets up TCP connections with each neighbour to exchange control information. We used this emulator to do five experiments, which evaluate the number of descriptions received under varying conditions. In the first experiment, peers only arrive and do not depart. The second and third experiment assess the performance of Orchard under flash crowds and churn. The fourth experiment measures the performance when multicasting a 4 Mbit/s data stream using UDP. The last experiment tests the performance of Orchard on Delft-37, a wide-area network which we will describe in the following section. All experiments except the fourth will only check the membership of peers of the multicast trees by having the emulator send one packet per second for each description instead of using a real data stream.

In our experiments, we assume four MDC descriptions. The source forwards each description at most three times. We have a central rendezvous server which keeps track of all peers interested in the video and which sends the sets of random, uniformly selected peers upon request. By keeping in touch with the rendezvous server, a peer can replace departed neighbours to make sure it always maintains a neighbour set of at least 20 peers ($m = 20$).

Peer arrivals are modelled as a Poisson process with an average of two per second. In a recent measurement of a large-scale Video-on-Demand system [102], it was shown that about half the peers disconnect within 10 minutes. Our measurements take a more pessimistic scenario into account, in order to be able to cope with zapping

number of descriptions	fraction of time		
	Arrivals only	Flash crowd (Figure 2.8)	Churn (Figure 2.11)
at least 1	0.987	0.989	0.986
at least 2	0.987	0.985	0.983
at least 3	0.986	0.975	0.971
4	0.974	0.874	0.853

Table 2.2: The cumulative distribution of the average number of descriptions received.

Type	Scenario	Colour distribution $\{n_1, n_2, n_3, n_4\}$	Avr. nr. of desc.
Measured	Arrivals only	$\{0.251, 0.251, 0.249, 0.248\}$	3.984
Analysis	Arrivals only	$n_1 = 0.25$, $n_2 = n_3 = n_4$	3.968
Measured	Flash crowd (Figure 2.8)	$\{0.254, 0.244, 0.252, 0.249\}$	3.867
Measured	Churn (Figure 2.11)	$\{0.252, 0.252, 0.249, 0.248\}$	3.846
Analysis	Departures only	$n_1 = 0.20$, $n_2 = n_3 = n_4$	3.851
Analysis	Departures only	$n_1 = 0.25$, $n_2 = n_3 = n_4$	3.905
Analysis	Departures only	$n_1 = 0.30$, $n_2 = n_3 = n_4$	3.866

Table 2.3: The average number of descriptions received and the colour distribution for the coloured peers. The colour distribution is fixed for the analyses and measured for the experiments.

behaviour which we expect to rise if P2P video multicasting becomes more popular. In those tests where peers depart, we let peers stay in the system for 120 seconds on average, using an exponential distribution.

2.5.2 Arrivals Only

In the first test we let 500 peers arrive but not depart. The performance results are shown in Table 2.2, which contains the fractions of peers which receive at least a certain number of descriptions. More than 97% of the peers was able to obtain all four descriptions. In Table 2.3, we compare the measured result (first row) against the expected performance as derived in Section 2.4 (second row). The table shows the average number of descriptions received by a peer at any moment in time, as well as the average colour distribution. On the second row, the table shows the lower bound

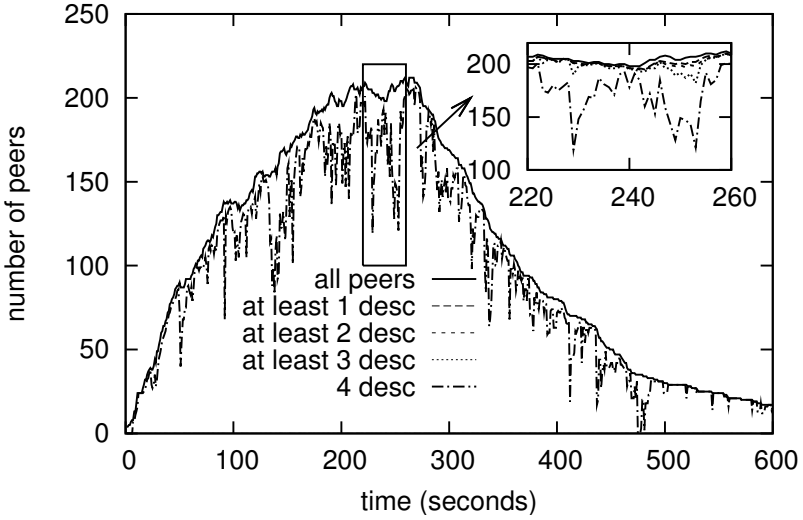


Figure 2.8: The number of descriptions received under flash crowds.

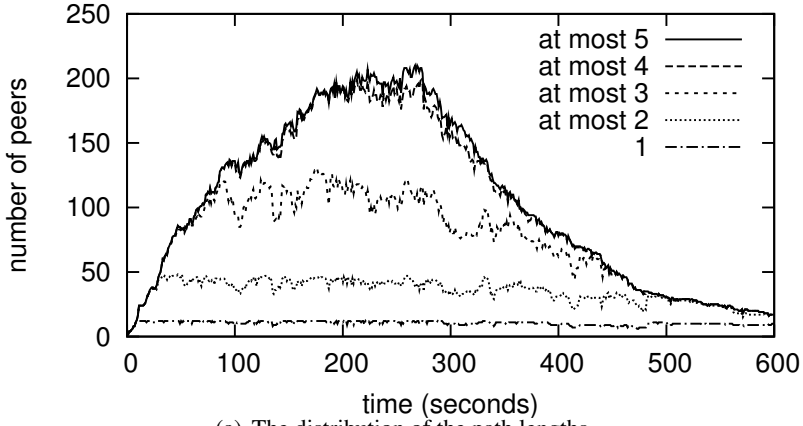
on the expected performance as derived in Section 2.4.5. Both rows are restricted to those peers that receive at least one description to make the numbers comparable to Figure 2.7. The measured results are consistent with the derived lower bound.

2.5.3 Flash Crowds

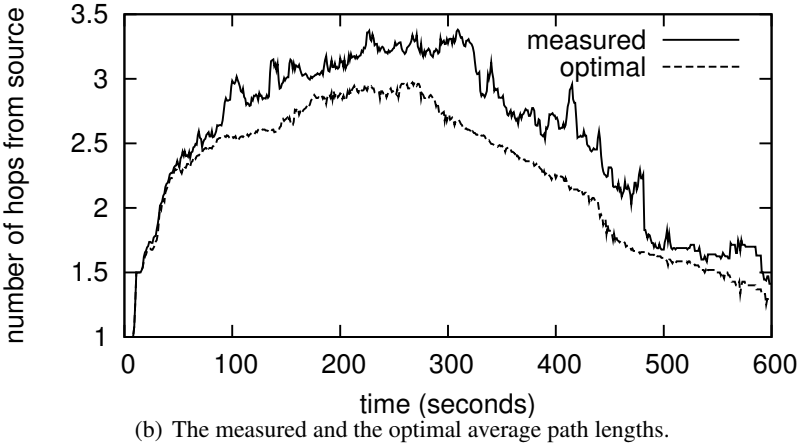
In the second test we let 500 peers arrive and depart. As the mean inter-arrival time is very short, initially there is indeed a fast build up of the number of peers. In Figure 2.8, we show the number of peers which receive a certain number of descriptions over time for a typical run. To keep the main graph readable, only the number of peers receiving all four descriptions and the total number of peers are represented. The inset shows a detailed sample which includes all curves.

All peers receive at least three out of four descriptions most of the time, but spikes can be observed in the number of peers which receive all descriptions. Peer departures, especially high up in trees, can cause many other peers to temporarily stop receiving one of their descriptions. However, the system recovers fast. In Table 2.2, we show the percentage of time a peer receives a certain number of descriptions, averaged from the moment the peer joins until it departs.

The number of hops from the source to every peer indicates the shallowness of the



(a) The distribution of the path lengths.



(b) The measured and the optimal average path lengths.

Figure 2.9: The distance from the source to the peers in the trees of their own colours.

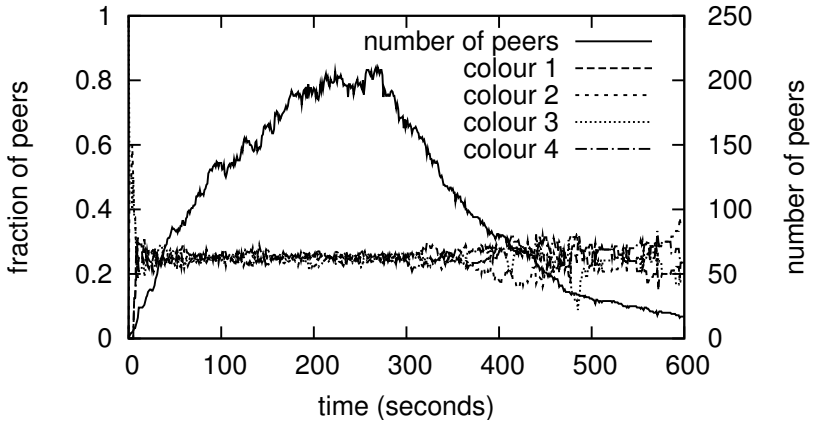


Figure 2.10: The distribution of the colours of the peers. The number of peers is plotted as well (scale on the right-hand side of the figure).

trees. The source forwards each colour to at most three peers, and every peer forwards its own colour to at most three peers as well (since there are four colours). The tree of every colour is thus ternary, and the minimal average distance from the source to each peer can be calculated accordingly (see Section 2.2.3).

The height of the trees over time in the same experiment is shown in Figure 2.9(a). Each line shows the number of peers at most a certain number of hops from the source, in the tree of their own colour. Even though peers are constantly arriving to and departing from the system, the trees do not degenerate. This is due to the fact that peers switch to parents of the same colour but closer to the source, whenever possible. We compare the average measured distance from a peer to the source against the optimal value in Figure 2.9(b). As can be seen in the figure, every peer is on average at most one hop further away from the source than in the optimal case.

In Figure 2.10, the distribution of the colours of the peers is shown. The number of peers in the system is plotted as well. The colours stay approximately evenly distributed. Of course, fluctuations are larger than when the number of peers is low.

In Table 2.3, we compare the measured results (third row) against the lower bounds on the expected performance as derived in Section 2.4.5 (rows 5–7). The table shows the average number of descriptions received by a peer at any moment in time, again restricted to those peers that receive at least one description. The colour distribution in this experiment fluctuates slightly around a uniform distribution. Be-

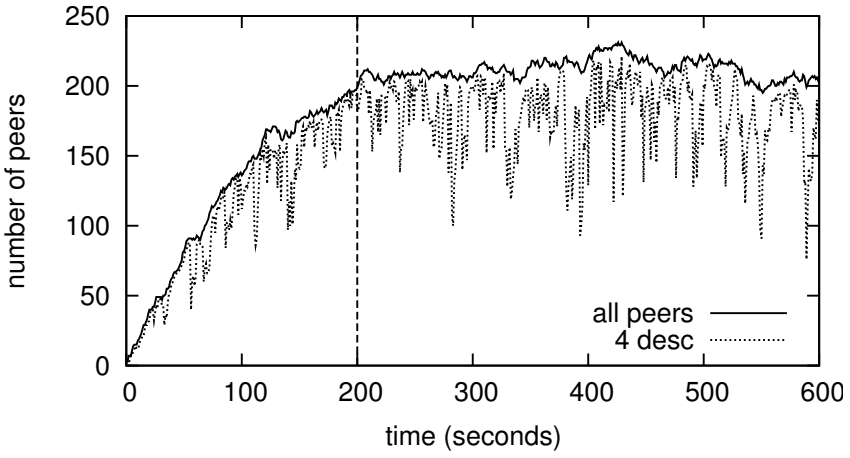


Figure 2.11: The number of descriptions received under churn. The initial 200 seconds are disregarded.

cause of this, we compare the measurements to Figure 2.7 using slight fluctuations as well, while $n_1 = 0.20, 0.25, 0.30$, $n_2 = n_3 = n_4$. The measured results are close to the derived lower bound on the expected performance in a stable situation. Also, it must be noted that in the analysis, every event is dealt with instantaneously, while in practice, events take time to process. This can cause peers to be in the process of repairing the tree for a certain amount of time. This performance loss is not taken into account in the analysis.

2.5.4 Churn

In the third experiment, we let 1200 peers arrive and depart. Initially, the departure rate is low, causing a flash crowd. Once the departure rate is (about) equal to the arrival rate of $2/s$, there is churn, which is the rapid arrival and departure of many peers in the network. The result of this experiment is shown in Figure 2.11, in which we only consider the data after 200 seconds, when the number of peers is (reasonably) stabilised. On average, the peers are still able to receive at least one description 98.6% of the time (see also Table 2.2). The continuous peer departures cause frequent big drops in the number of peers which are receiving all four colours, but the system also recovers quickly. This shows that the Orchard algorithm can handle sustained peer arrivals and departures and still deliver part of the video stream to most of the users,

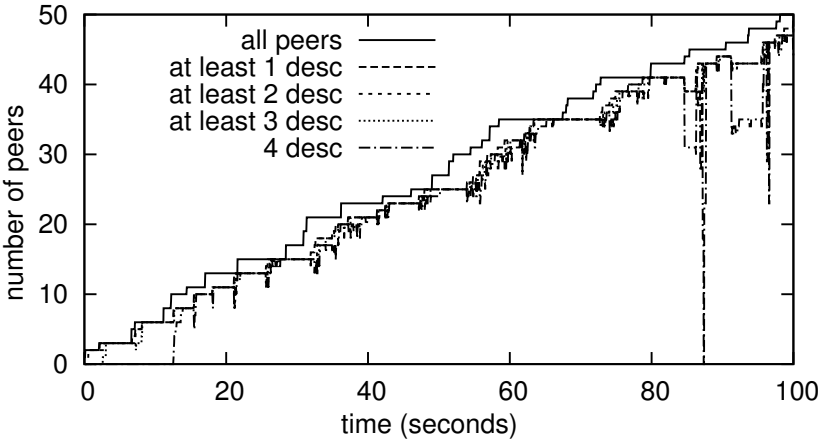


Figure 2.12: The number of descriptions received when streaming 4 Mbit/s video.

most of the time.

The measured results are again compared to the lower bound on the expected performance in Table 2.3 on row 4 and rows 5–7, respectively. Although the performance for this experiment is a bit less than in a flash crowd, it still comes close to the values derived in Section 2.4.5.

2.5.5 Real Streaming

In this experiment, we let 50 peers arrive. A 4 Mbit/s data stream (1 Mbit/s per description) was distributed using UDP. The result of this experiment is shown in Figure 2.12. The spikes are due to packet loss. When streaming these amounts of data, the peers require a bit more time to obtain their first description. The huge amount of UDP traffic at each peer slows down the ability of the emulator to initiate TCP connections. Thus, it takes a peer somewhat longer to establish connections with its neighbours, which causes additional delay between arriving in the system and obtaining the first description.

We do not do any retransmits of dropped UDP packets, nor do we define how much packet loss a peer tolerates from its parent. These parameters depend on the network (such as router configuration, network congestion, and the latencies between the peers) and local parameters (such as the codec used and the buffer size). Considering these parameters is outside the scope of this chapter.

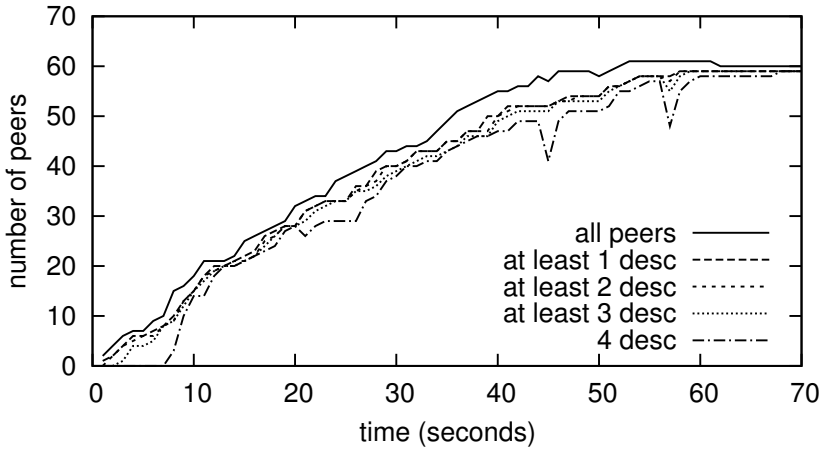


Figure 2.13: The number of descriptions received in Delft-37.

2.5.6 Delft-37

In the last experiment, we distribute the peers over the Delft-37 network. The Delft-37 nodes we used cannot all host the same number of peers due to different amounts of resources being available on the nodes. In this test, we started 65 peers, with a distribution as shown in Table 2.1. The results of this experiment are shown in Figure 2.13.

Every Delft-37 node forms the end point of a large number of TCP connections, since every peer maintains a TCP connection to all of its neighbours. Five of the peers (two in Washington, one in Georgia, two in Germany) were able to start, but were not able to set up any TCP connections. This caused four of these peers to depart, explaining the three small dips in the 'all peers' curve. We suppose this happened due to the high number of TCP connections already started on these nodes. The 60 peers that did not depart were all able to obtain all colours.

2.5.7 Scalability of Orchard

In the experiments we have presented above, there are at most a few hundred peers in the system at a single point in time. If the number of peers is significantly larger, the behaviour of Orchard could change. The analysis in Section 2.4 is independent on the number of peers and thus scales, but it does not cover all aspects of Orchard. More specifically, the trees created by Orchard become deeper if the number of peers in

the system increases, and so the average distance from a peer to the source increases. As a result, the differences in latencies with which a peer receives the colours may increase. The amount of buffering done at a peer has to be sufficient to cope with these differences. Also, the probability that a peer departs along the path from a peer to the source increases, which is likely to result in more packet loss, or even the complete loss of a colour. However, these problems are not specific to Orchard, but they will occur in any ALM streaming algorithm that employs trees.

2.6 Related Work

The idea of using MDC [44] in ALM has been proposed before [20, 81, 94]. Pouwelse et al. [81] propose the use of a bartering mechanism, but do not provide an algorithm. SplitStream [20] uses MDC to create a forest interior-node-disjoint trees and can be modified in such a way that every node forwards as much as it receives, if such a forest can be constructed. However, SplitStream was designed for cooperative environments, and lacks the mechanisms to enforce a fair bandwidth contribution. Peers respect the forwarding capacity claimed by their neighbours, and a peer can thus easily free-ride by lying about its forwarding capacity. In contrast, Orchard forces the peers to contribute, and avoids relying on the peers' own claims.

It is not necessary to use MDC to create resilience against peer failure. In Chainsaw [76], the video stream is cut into pieces, with peers forwarding to each other the most recent pieces as long as bandwidth allows. Each peer requests the pieces it is still missing from its neighbours. This makes it very resilient against peer failure, because a failing neighbour causes a peer to (re-)request the pieces from someone else.

The traditional single tree approach is the basis of many algorithms. In NICE [13] and ZigZag [95], a hierarchy of groups ensures members of a group can replace each other in case of peer failures.

None of the algorithms mentioned above provides a mechanism to enforce a fair resource contribution. All of the algorithms allow a peer to define its own maximum out-degree, and count on peers to be honest about their latency to others. This makes cheating easy [64]. Peers can simply pretend to be unable to forward data, and will still be served. Unlike those algorithms, Orchard enforces fair resource contribution and is still resilient against high-rate peer arrivals and departures.

BAR gossip [58] is a multicast solution based on an epidemic approach, in which fairness is encouraged by letting peers exchange pieces of the video stream and pro-

viding countermeasures against malicious peers. However, this algorithm relies on a central, trusted auditor to evict malicious peers from the system. The same holds for the system proposed by Haridasan et al. [51], which uses a similar approach but is able to achieve better performance.

Several algorithms have been proposed to combat free-riding in multicast systems. Some [22, 75] assign the role of central authority to the source, letting it manage the tree or enforce other rules on the other peers. However, this puts an additional load on the source, which it may not be able to handle.

Habib et al. [50] let peers keep each other informed about the behaviour of others, and punish free-riders. This creates an incentive for peers to contribute resources, but requires a central authority or the deployment of a distributed trust system to avoid peers spreading false information about others. A distributed trust system adds complexity, it needs to exist beforehand, and it needs to cover all the receiving peers. This is not always feasible or practical.

Ngan et al. [72] propose a system in which the multicast tree is periodically reshuffled. If a free-rider refuses to forward to another peer, that other peer could become the free-rider's parent after a reshuffle, and subsequently deny serving the free-rider. Such a system is very vulnerable to Sybil attacks, in which a free-rider assumes a new identity. To combat this, the authors propose the use of certified nodeIds. However, this requires a central authority or a distributed trust mechanism as well.

2.7 Discussion

The Orchard algorithm forces peers to spend as much upload bandwidth as they use download bandwidth. As a result, peers with asymmetric links (such as ADSL) are at a disadvantage. Even if such peers have enough download bandwidth to receive the video stream, they may not have enough upload bandwidth to satisfy the deals required to obtain it. For networks in which peers do not necessarily have enough upload bandwidth, the video distribution mechanism can to be extended to allow peers to repay their used bandwidth in subsequent sessions [42]. We consider the aspects of fairness across sessions to be outside the scope of this thesis however, and consider a different approach in Chapters 3 and 4, in which we will allow peers to provide upload bandwidth to those with asymmetric links, without requiring the same amount of data in return. The peers with asymmetric links will thus be able to receive a higher quality of service than would be possible when using Orchard.

Furthermore, in order to use the Orchard algorithm, the application developer has to support Multiple Description Coding, which is not (yet) a common video encoding technique. Also, no free implementation exists that we are aware of. Layered Video Coding [57], which is a similar but more common technique, introduces dependencies between the layers, which are otherwise similar to descriptions in MDC. These dependencies cause the first (base) layer to be essential in the decoding of any other layer. Would such layers be used as colours in the Orchard algorithm, the robustness of the video distribution would critically depend on the distribution of a single colour. Any peer of that colour that departs from the network would force others to interrupt their video playback because they lost the critical base layer, instead of allowing a graceful degradation in video quality as is the case in MDC when only a single description is lost.

Although our analysis and experiments assume that peers get a neighbourhood set consisting of peers selected uniformly at random, the Orchard algorithm does not require this restriction. A possible way to improve the performance of Orchard is to make the neighbourhood selection algorithm in the underlying P2P network locality aware [78, 104]: if peers strike deals with neighbours close to them in the underlying network, less network traffic is generated and thus less network congestion will occur.

2.8 Conclusions

When multicasting video streams using Application Level Multicasting, each byte received by a peer has to be uploaded by another peer. Through the Orchard algorithm, we have shown that it is possible to construct an ALM system in P2P networks which is *fair*, *decentralised* and *resilient*.

The Orchard algorithm is fair, because peers are forced to share the burden of uploading equally: no peer has to upload more data than it downloads. We achieved this by using Multiple Description Coding, and by letting the peers strike deals to exchange and redirect descriptions. Every deal ensures that a peer will not have to forward more than it receives. The Orchard algorithm is decentralised as it does not need any central component. Although the source is a special component, no peer is required to know it. The Orchard algorithm is resilient against peer arrivals and departures due to the use of multiple descriptions. This was confirmed by analysing the expected performance of Orchard as well as through emulation.

We have thus shown that it is possible to be resilient against free-riding in a peer-to-peer network for live video streaming. Our approach is resilient to a reasonable amount of churn as well as against several types of malicious behaviour. However, our approach does require a video codec with support for Multiple Description Coding. Our novel forest-construction primitives, based on equal data exchange and on the creation of forwarding triangles between peers, can aid in the design of other algorithms in which an equal exchange of resources is desired.

Chapter 3

Swarm-based Video-on-Demand

THE ORCHARD ALGORITHM presented in Chapter 2 is designed to support live video streaming, but cannot be easily extended to support video-on-demand. The peers in Orchard are able to exchange video data, because they all require the same data. When streaming a video in a video-on-demand fashion, the playback positions of the peers are independent of each other. A peer further ahead in the video has no need for the data obtained by a peer that has just started playback at the beginning of the video. Neither the single tree nor the multiple tree approach is really suitable for video-on-demand for exactly this reason. Both approaches assume that the same data can be sent to all of the peers at the same time, allowing the trees to be reconfigured by swapping any two nodes. However, in video-on-demand, using a tree structure implies that each node is further ahead in the video than its children to be able to supply them with video data. Such a restriction severely limits any tree reconfigurations should inner nodes depart before their children do.

A different approach is thus needed to provide the peers with an incentive for uploading when streaming video-on-demand. Instead of using tree structures, we turn to the swarm-based approach. The swarm-based data distribution model cuts the data into pieces of fixed size. The peers announce to each other which pieces they have obtained, and allow these pieces to be requested by others. The order in which the pieces are propagated through the network can be customised, as each peer can request any piece available at its neighbours. In contrast, the tree-based approaches typically forward the data in-order. Since pieces in the swarm-based distribution can be requested by each peer in any order, each peer can obtain the video stream from multiple neigh-

bours by distributing the requests among them. A peer can thus dynamically create multiple substreams from its neighbours, without the need for specialised techniques like Multiple Description Coding, as is used in Orchard. However, video-on-demand does not allow the exchange of pieces between peers to happen on a one-to-one basis, as peers at the same moment in time may require different parts of the data. A peer that has just joined the network is interested in downloading the full video, while other peers may be interested in only the last few pieces as they have downloaded most of the video already.

In this chapter, we present *Give-to-Get*, a P2P video-on-demand algorithm in which peers are given an incentive to upload data. The incentive is created by rewarding peers that prove to be good uploaders. A peer thus has to "give" in order to "get", a concept which bears close resemblance to the indirect reciprocity found in human societies in general [73]. We present Give-to-Get as an extension to the popular BitTorrent protocol [26] which is originally designed for non-streaming file sharing. By extending BitTorrent, we allow many existing BitTorrent implementations to be adapted to support video-on-demand, which we believe makes Give-to-Get easier to deploy. For the same reason, we have designed Give-to-Get to be video-codec agnostic: we assume that the video stream is a stream of bits to be played at a constant speed. Give-to-Get thus does not require advanced video-encoding techniques such as Multiple Description Coding (as is required for Orchard in Chapter 2) or advanced video-playback techniques like Adaptive Playback [56, 84], which allows the video speed to be varied to match the available bandwidth to the player.

This chapter is organized as follows. We further specify the problem we address in Section 3.1, followed by a description of the Give-to-Get algorithm in Section 3.2. Next, we present our experiments and their results in Section 3.3. We compare the performance of Give-to-Get to that of an analytical model in Section 3.4. In Section 3.5, we discuss related work. Finally, we draw conclusions in Section 3.6.

3.1 Problem Description

The problem we address in this chapter is the design of a P2P VoD algorithm which discourages free-riding. A free-rider is a peer which consumes more resources than it contributes to the P2P system. We will assume that a peer will not try to cheat the system in a different way, such as being a single peer emulating several peers (also called a Sybil attack [34]), or several peers colluding. In this section, we will describe

how a P2P VoD system operates in our setting in general. We assume the algorithm to be designed for P2P VoD to be video-codec agnostic, and we will consider the video to be a constant bit-rate stream with unknown boundary positions between the consecutive frames. Similarly to BitTorrent [26], we assume that the video file to be streamed is split up into chunks of equal size, and that every peer interested in watching the stream tries to obtain all chunks. Due to the similarities with BitTorrent, we will use its terminology to describe both our problem and our proposed solution.

A P2P VoD system consists of peers which are downloading the video (leechers) and of peers which have finished downloading and upload for free (seeders). The system starts with at least one seeder. We assume that a peer is able to obtain the addresses of a number of random other peers, and that connections are possible between any pair of peers. To provide all leechers with the video data in a P2P fashion, a multicast tree has to be used for every chunk of the video. Such a multicast tree can be built explicitly or emerge implicitly as the union of paths over which a certain chunk travelled to each peer. While in traditional application-level multicasting, the same multicast tree is created for all chunks and is changed only when peers arrive or depart, we allow the multicast trees of different chunks to be different based on the dynamic behavior of the peers. These multicast trees are not created ahead of time, but rather come into being while chunks are being propagated in the system.

A peer typically behaves in the following manner: It joins the system as a leecher and contacts other peers in order to download chunks of a video.After a prebuffering period, the peer starts playback. When the video has finished playing, the peer will depart. If the peer is done downloading the video before playback is finished, it will stay as a seeder until it departs. We assume that peers can arrive at any time, but that they will start playing the video from the beginning and at a constant speed. Similar to other P2P VoD algorithms like BiToS, we do not consider seeking or fast-forwarding. Give-to-Get can be extended to support these operations, but such extensions are outside the scope of this thesis. We chose to focus on the core algorithm for basic asynchronous playback. Fast-forwarding introduces its own class of problems. For example, a peer that wants to fast-forward most likely has to play the video at a faster rate than it can be downloaded, or has to receive the video from others at a lower bitrate. Support for seeking could be provided at a level above Give-to-Get, by cutting the video stream into pieces of say one minute in size each, and creating a separate Give-to-Get swarm for each piece [12]. In such a system, peers can seek by switching to the swarm containing the target timestamp. Note that rewinding can be

easily supported in the player itself without help of Give-to-Get.

3.2 Give-to-Get

In this section, we will explain Give-to-Get (G2G). First, we will describe how a peer maintains information about other peers in the system in its neighbour set. Then, the way the video pieces are forwarded from peer to peer is discussed. Next, we show in which order video pieces are transferred between peers. Fourth, we will discuss the differences with the related BitTorrent [26] and BiToS [98] protocols. Finally, we will discuss our performance metrics.

3.2.1 Neighbour Management

The system we consider consists of peers which are interested in receiving the video stream (leechers) and peers which have obtained the complete video stream and are willing to share it for free (seeders). We assume a peer is able to obtain addresses of other peers uniformly at random. Mechanisms to implement this could be centralised, with a server keeping track of who is in the network, or decentralised, for example, by using epidemic protocols or DHT rings. We view this peer discovery problem as orthogonal to our work, and so beyond the scope of this chapter. From the moment a peer joins the system, it will obtain and maintain a set of 10 neighbours in its neighbour set. When a peer is unable to contact 10 neighbours, it will periodically try to discover new neighbours. Although maintaining more neighbours typically boosts the performance in most P2P networks (including Give-to-Get), we keep the number of neighbours small to prevent the P2P network in our simulation from becoming (nearly) fully connected. The performance measured in simulations of a fully connected network cannot be extrapolated towards bigger networks.

Once a peer becomes a seeder, it will disconnect from other seeders to avoid maintaining useless connections.

3.2.2 Chunk Distribution

The video data is split up into *chunks* of equal size. As G2G is codec agnostic, the chunk boundaries do not necessarily coincide with frame boundaries. Peers obtain the chunks by requesting them from their neighbours. A peer keeps its neighbours

Algorithm 1 Unchoking algorithm.

```
choke(all neighbours)
N ⇐ all interested neighbours
sort N on forwarding rank
for i = 1 to min(|N|, 3 + n) do
    unchoke(N[i])
    b ⇐ Σ_{k=1}^{i} (our upload speed to N[k])
    if i ≥ 3 and b > UPLINK * 0.9 then
        break
```

informed about the chunks it has, and decides which of its neighbours is allowed to make requests. A neighbour which is allowed to make requests is *unchoked*. When a chunk is requested by a neighbour, the peer appends it to the send queue for the corresponding connection. Chunks are uploaded using subchunks of 1 Kbyte to avoid delays in the delivery of control messages, which are sent with a higher priority. We find a subchunk size of 1 Kbyte to be a practical trade-off between having small delays in the interaction between peers (when using small subchunks), and having a low overhead (when using big and thus fewer subchunks).

Every δ seconds, a peer decides which neighbours are unchoked based on information gathered over the last δ seconds. The neighbours which have shown the best performance will be unchoked, as well as a randomly chosen neighbour (optimistic unchoking). G2G employs a novel unchoking algorithm, described in pseudocode in Algorithm 1. A peer p ranks its neighbours according to decreasing *forwarding ranks*, which is a value representing how well a neighbour is forwarding chunks. The calculation of the forwarding rank is explained below. Peer p unchokes the three highest-ranked neighbours. Since peers are judged by the amount of data they forward, it is beneficial to make efficient use of the available upload bandwidth. To help saturate the uplink, subsequently more neighbours are unchoked until the uplink bandwidth necessary to serve the unchoked peers reaches 90% of p's uplink. At most n neighbours are unchoked this way to avoid serving too many neighbours at once, which would decrease the performance of the individual connections. The optimal value for n likely depends on the available bandwidth and latency of p's network connections. Note that a value of $n = 1$ discourages a fast dissemination of information by allowing peers to forward to only one other peer, essentially turning the data distribution topology into a chain. In our experiments, we use $n = 2$; larger values of n did not

yield a noteworthy performance improvement in our simulations.

To search for better children, p round-robins over the rest of the neighbours and optimistically unchokes a different one of them every 2δ seconds. If the optimistically unchoked neighbour proves to be a good forwarder and ends up at the top, it will be automatically kept unchoked. New connections are inserted uniformly at random in the set of neighbours. The duration of 2δ seconds turns out to be enough for a neighbour to prove its good behaviour. By having a peer upload chunks to only the best forwarders, its neighbours are encouraged to forward the data as much as possible. Peers are not obliged to forward data, but may not be able to receive video data once other peers start to compete for it. This results in a system where free-riders are tolerated only if there is sufficient bandwidth left to serve them.

A peer p ranks its neighbours based on the number of chunks they have forwarded during the last δ seconds. Our ranking procedure consists of two steps. First, the neighbours are sorted according to the decreasing numbers of chunks they have forwarded to other peers, counting only the chunks they originally received from p. If two neighbours have an equal score in the first step, they are sorted in the second step according to the decreasing total number of chunks they have forwarded to other peers. Either step alone does not suffice as a ranking mechanism. If neighbours are ranked solely based on the total number of chunks they upload, good uploaders will be unchoked by all their neighbours, which causes only the best uploaders to receive data and the other peers to starve. On the other hand, if neighbours are ranked solely based on the number of chunks they receive from p and forward to others, peers which are optimistically unchoked by p have a hard time becoming one of the top ranked forwarders. An optimistically unchoked peer q would have to receive chunks from p and hope for q's neighbours to request exactly those chunks often enough. The probability that q replaces the other top forwarders ranked by p is too low.

Peer p has to know which chunks were forwarded by its children to others. To obtain this information, it cannot ask its children directly, as they could make false claims. Instead, p asks its grandchildren for the behaviour of its children. The children of p keep p updated about the peers they are forwarding to. Peer p contacts these grandchildren, and asks them which chunks they received from p's children. This allows p to determine both the forwarding rates of its children as well as the numbers of chunks they forwarded which were originally provided by p. Because peer p ranks its children based on the (amount of) data they forward, the children of p have an incentive to forward as much as possible to obtain a high rank. Figure 3.1 shows an

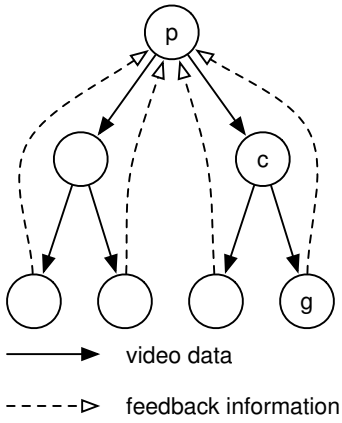


Figure 3.1: The feedback connections for an individual peer.

example of the flow of feedback information. Peer p has unchoked two other peers, amongst which peer c. Peer c has peer g unchoked. Information about the amount of video data uploaded by c to g is communicated over the dashed arrow back to p. Peer p can subsequently rank child c based on this information. Note that a node c has no incentive to lie about the identities of its children, because only its actual children will provide feedback about c's behaviour.

For the above approach to work, it is critical that p and c do not collude. Although we consider protection against collusion attacks to be outside the scope of this thesis, we do note that it is possible to protect the network against collusion. For example, peers could be forbidden to connect to peers outside of the set they are given by the peer discovery algorithm. By assigning peers random sets of neighbours, the chance of two colluding peers being allowed to report about each other's behaviour is greatly reduced.

3.2.3 Chunk Picking

A peer obtains chunks by issuing a request for each chunk to other peers. A peer thus has to decide in which order it will request the chunks it wants to download; this is called *chunk picking*. When a peer p is allowed by one of its neighbours to make requests to it, it will always do so if the neighbour has a chunk p wants. We associate with every chunk and every peer a *deadline*, which is the latest point in time the chunk has to be present at the peer for playback. As long as p has not yet started playback,

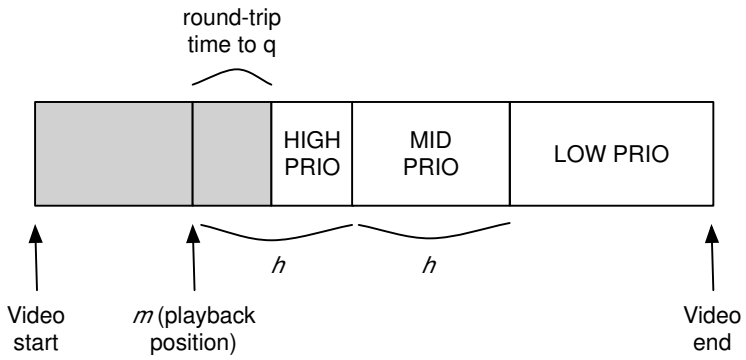


Figure 3.2: The high-, mid- and low-priority sets in relation to the playback position. The chunks in the grey areas, if requested, will not arrive before their deadline.

the deadline of every chunk at p is infinite. Peer p wants chunk i from a neighbour q if the following conditions are met: a) q has chunk i, b) p does not have chunk i and has not previously requested it, and c) it is likely that chunk i arrives at p before its deadline. Peer p will never request pieces it does not want. Because peers keep their neighbours informed about the chunks they have, the first two rules are easy to check. To estimate whether a chunk will arrive on time, p keeps track of the response time of requests. This response time is influenced by the link delay between p and q as well as the amount of traffic from p and q to other peers. Peers can submit multiple requests in parallel in order to fully utilise the links with their neighbours.

When deciding the order in which chunks are picked, two things have to be kept in mind. First, it is necessary to provide the chunks in-order to the video player. Secondly, to achieve a good upload rate, it is necessary to obtain enough chunks which are wanted by other peers. The former favours downloading chunks in-order, the latter favours downloading rare chunks first. To balance between these two requirements, G2G employs a hybrid solution by prioritizing the chunks that have yet to be played back. Let m be the playback position of peer p, or 0 if p has not yet started playback. Peer p will request chunk i on the first match in the following list of sets of chunks (see Figure 3.2):

- *High priority*: $m \leq i < m+h$. If p has already started playback, it will pick the lowest such i, otherwise, it will pick i rarest first.

- *Mid priority*: $m+h \leq i < m+(\mu+1)h$. Peer p will choose such an i rarest first.

- *Low priority*: $m + (\mu + 1)h \leq i$. Peer p will choose such an i rarest first.

In these definitions, h and μ are parameters which dictate the amount of clustering of chunk requests in the part of the video yet to be played back. A peer picks rarest-first based on the availability of chunks at its neighbours. Among chunks of equal rarity, i is chosen uniformly at random.During playback, the chunks with a tight deadline are downloaded first and in-order (the high priority set). The mid-priority set makes it easier for the peer to complete the high-priority set in the future, as the (beginning of the) current mid-priority set will be the high-priority set later on. This lowers the probability of having to do in-order downloading later on. The low-priority set will download the rest of the chunks using rarest-first both to collect chunks which will be forwarded often because they are wanted by many and to increase the availability of the rarest chunks. Also, the low priority set allows a peer to collect as much of the video as fast as possible.

3.2.4 Differences between Give-to-Get, BitTorrent and BiToS

In our experiments, we will compare the performance of G2G to that of BiToS [98]. BiToS is a P2P VoD algorithm which, like G2G, is inspired by BitTorrent. For BiToS, we will use the optimal settings as derived in the chapter where BiToS is introduced [98]. The major differences between G2G, BiToS and BitTorrent lie in the chunk-picking policy, the choking policy and the prebuffering policy. In BiToS, two priority sets are used: the high-priority set and the remaining-pieces set. The high-priority set is defined to be 8% of the video length. Peers request pieces from the high-priority set 80% of the time, and from the remaining-pieces set 20% of the time. The rarest-first policy is used in both cases, with a bias towards earlier chunks if they are equally rare. In contrast, G2G uses three priority sets. In-order downloading is used in the high-priority set once playback has started, and in the mid- and low-priority sets, the rarest-first policy chooses at random between pieces of equal rarity. BitTorrent is not designed for VoD and thus does not define priority sets based on the playback position.

The choking policy determines which neighbours are allowed to request pieces, which defines the flow of chunks through the P2P network. BiToS, like BitTorrent, is based on tit-for-tat, while G2G is not. In tit-for-tat, a peer a will allow a peer b to make requests for chunks if b proved to be one of the top uploaders to a. In contrast, a peer a in G2G will allow b to make requests if b proves to be one of the

top forwarders to others (this set of others can include a). Tit-for-tat works well in an off-line download setting (such as BitTorrent) where peers have enough chunks they can exchange. However, it is less suitable for VoD because peers in VoD bias their interests on downloading the chunks close after their playback position, and are not interested in chunks before their playback position. Two peers subsequently either have overlapping interests if their playback positions are close, or the peer with the earliest playback position is interested in the other's chunks but not vice-versa. One-sided interests are the bane of tit-for-tat systems. In BiToS, peers download pieces outside their high-priority set 20% of the time, relaxing the one-sided interests problem somewhat, as all peers have a mutual interest in obtaining the end of the video this way. The choking policy in G2G consists of unchoking neighbours which have proven to be good forwarders. Subsequently, the requirement to exchange data between peers, and thus to have interests in each other's chunks, is not present in G2G.

3.2.5 Performance Metrics

For a peer to view a video clip in a VoD fashion, two conditions must be met to provide a good quality of service. First, the start-up delay must be small, and secondly, the chunk loss must be low to provide good playback quality. If either of these conditions is not met, it is likely the user was better off downloading the whole clip before viewing it. In G2G, we say a chunk is lost if a peer cannot request it in time from one of its neighbours, or if the chunk was requested but did not arrive in time. The concept of buffering the beginning of the video clip before starting playback is common to most streaming video players. We will assume that once prebuffering is finished and playback is started, the video will not be paused or slowed down. In general, the amount of prebuffering is a trade-off between having a short waiting period and having low chunk loss during playback.

We define the prebuffering time as follows. First, a peer waits until it has the first h chunks (the initial high-priority set) available to prevent immediate chunk loss. Then, it waits until the expected remaining download time is less than the duration of the video. The expected remaining download time is extrapolated from the download speed so far, with a 20% safety margin. This margin allows for short or small drops in download rate later on, and will also create an increasing buffer when the download rate does not drop. Drops in download rate can occur when a parent of a peer p adopts more children or replaces p with a different child due to p's rank or to optimistic

unchoking. In the former case, the uplink of p's parent has to be shared by more peers, and in the latter case, p stops receiving anything from that particular parent. When and how often this will occur depends on the behaviour of p and its neighbours, and is hard to predict. The safety margin in the prebuffering was added to protect against such behaviour. It should be noted that other VoD algorithms which use BitTorrent-like unchoking mechanisms (such as BiToS [98]) are likely to suffer from the same problem. In order to keep the prebuffering time reasonable, the average upload speed of a peer in the system should thus be at least the video bitrate plus a 20% margin. If there is less upload capacity in the system, peers both get a hard time obtaining all chunks and are forced to prebuffer longer to ensure the download will be finished before the playback is.

Once a peer has started playback, it requires chunks at a constant rate. The chunk loss is the fraction of chunks that have not arrived before their deadlines, averaged over all the playing peers. When reporting the chunk loss, a 5-second sliding window average will be used to improve the readibility of the figures. Neither BiToS nor BitTorrent define a prebuffering policy, so to be able to make a fair comparison between BiToS and G2G in our experiments, we will use the same prebuffering policy for BiToS as for G2G. BiToS uses a larger high-priority set (8% of the video length, or 24 seconds for the 5-minute video we will use in our experiments), making it unfair to let peers wait until their full high-priority set is downloaded before playback is started. Instead, like in G2G, we wait until the first $h = 10$ seconds are downloaded.

3.3 Experiments

In this section we will present our experimental setup as well as the results of two experiments. In the first experiment, we measure the default behaviour with well-behaving peers and in the second experiment we let part of the system consist of free-riders. In both cases, we will compare the performance of G2G and BiToS.

3.3.1 Experimental Setup

Our experiments were performed using a discrete-event simulator, emulating a network of 500 peers which are all interested in receiving the video stream. The network is assumed to have no bottlenecks except at the peers. Packets are sent using TCP with a 1500 byte MTU and their delivery is delayed in case of congestion in the up-

link of the sender or the downlink of the receiver. Each simulation starts with one initial seeder, and the rest of the peers arrive according to a Poisson process. Unless stated otherwise, peers arrive at a rate of 1.0/s, and depart when playback is finished. When a peer is done downloading the video stream, it will therefore become a seeder until the playback is finished and the peer departs.

We will simulate a group of peers with asymmetric bandwidth capacities, which is typical for end-users on the Internet. Every peer has an uplink capacity chosen uniformly at random between 0.5 and 1.0 Mbit/s. The downlink capacity of a peer is always four times its uplink capacity. The round-trip times between peers vary between 100 ms and 300 ms[1]. A peer reconsiders the behaviour of its neighbours every $\delta = 10$ seconds, which is a balance between keeping the overhead low and allowing neighbour behaviour changes (including free-riders) to be detected. The high-priority set size h is defined to be the equivalent of 10 seconds of video. The mid-priority set size is $\mu = 4$ times the high-priority set size.

A 5-minute video of 0.5 Mbit/s is cut up into 16 Kbyte chunks (i.e., 4 chunks per second on average) and is being distributed from the initial seeder with a 2 Mbit/s uplink. We will use the prebuffering time and the chunk loss as the metrics to assess the performance of G2G. The actual frame loss depends on the video codec and the dependency between encoded frames. Because G2G is codec-agnostic, it does not know the frame boundaries in the video stream and thus, we cannot use frame loss as a metric. In all experiments, we will compare the performance of G2G and a BiToS system. We will present the results of a representative sample run to show typical behaviour. We will use the same arrival patterns, neighbour sets and peer capabilities when comparing the performance of G2G and BiToS.

3.3.2 Default Behaviour

In the first experiment, peers depart only when their playback is finished, and there are no free-riders. We do three runs, letting peers arrive at an average rate of 0.2/s, 1.0/s, and 10.0/s, respectively.to that of BiToS when using the same arrival pattern and network configuration. In Figure 3.3, the number of playing peers in the system is shown as well as the average percentage of chunk loss. In the top and middle graphs, peers start departing before all of them have arrived, resulting in a more or less stable

[1]These figures are realistic for broadband usage within the Netherlands. The results are nevertheless representative, because the overhead of G2G is low compared to the video bandwidth.

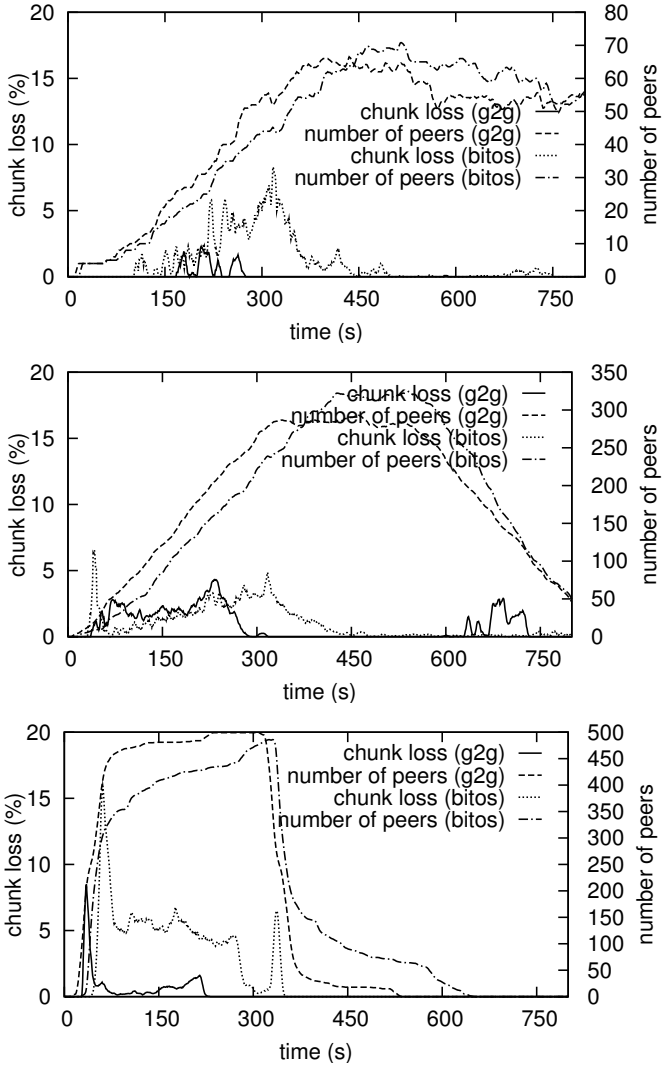


Figure 3.3: The average chunk loss and the number of playing peers for peers arriving at 0.2/s (top), 1.0/s (middle), and 10.0/s (bottom).

number of peers for a certain period that ends when all peers have arrived. In the bottom graph, all peers have arrived within approximately 50 seconds, after which the number of peers is stable until all of them are done playing. As long as the initial seeder is the only seeder in the system, peers experience some chunk loss. Because all peers are downloading the video, there is much competition for bandwidth. Once some peers are done downloading the video, they can seeder it to others and after a short period of time, no peer experiences any chunk loss at all. In effect, the seeders form a content distribution network aided by the peers which continue to forward chunks to each other. The performance of Give-to-Get exceeds that of BiToS for both low and high arrival rates (the top and the bottom graphs), as the average chunk loss is significantly lower for Give-to-Get. For an arrival rate of 1.0/s (the middle graph), the amount of chunk loss in Give-to-Get is similar to that in BiToS. The performance of Give-to-Get thus equals or exceeds that of BiToS in these simulations.

Figure 3.4 shows the distribution of the chunk loss for each arrival rate across the peers, sorted decreasingly. At all three rates, the chunk loss is concentrated on a small number of peers, but much more so for G2G than for BiToS. The graphs for G2G are mostly below those of BiToS, implying that when considering chunk loss, most peers are better off using G2G. A more sophisticated playback policy, such as allowing the video stream to pause for rebuffering, could potentially alleviate the heavy losses that occur for some peers using either algorithm.

Figure 3.5 shows the cumulative distribution of the required prebuffering time, which increases with the arrival rate. At higher arrival rates, it takes an increasing amount of time before the initial pieces are spread across the P2P network. A peer has to wait longer before its neighbours have obtained any pieces, and thus the average prebuffering time increases. The required prebuffering time is longer in BiToS, which can be explained by the fact that the high-priority set is large (24 seconds) and is downloaded with the rarest-first policy, so it takes longer to obtain the initial 10 seconds of the video.

3.3.3 Free-riders

In the second experiment, we add free-riders to the system by having 20% of the peers not upload anything to others. Figure 3.6 shows the average chunk loss separately for the well-behaving peers and the free-riders. The well-behaving peers are affected by the presence of the free-riders, and experience a higher chunk loss than in the

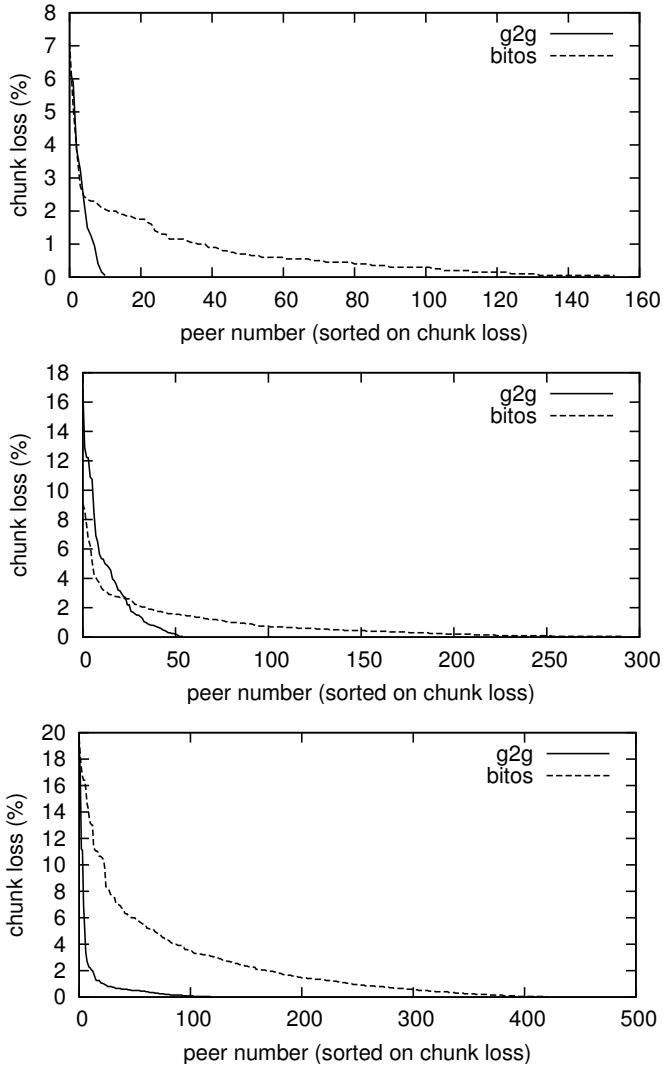


Figure 3.4: The distribution of the chunk loss over the peers for peers arriving at 0.2/s (top), 1.0/s (middle), and 10.0/s (bottom).

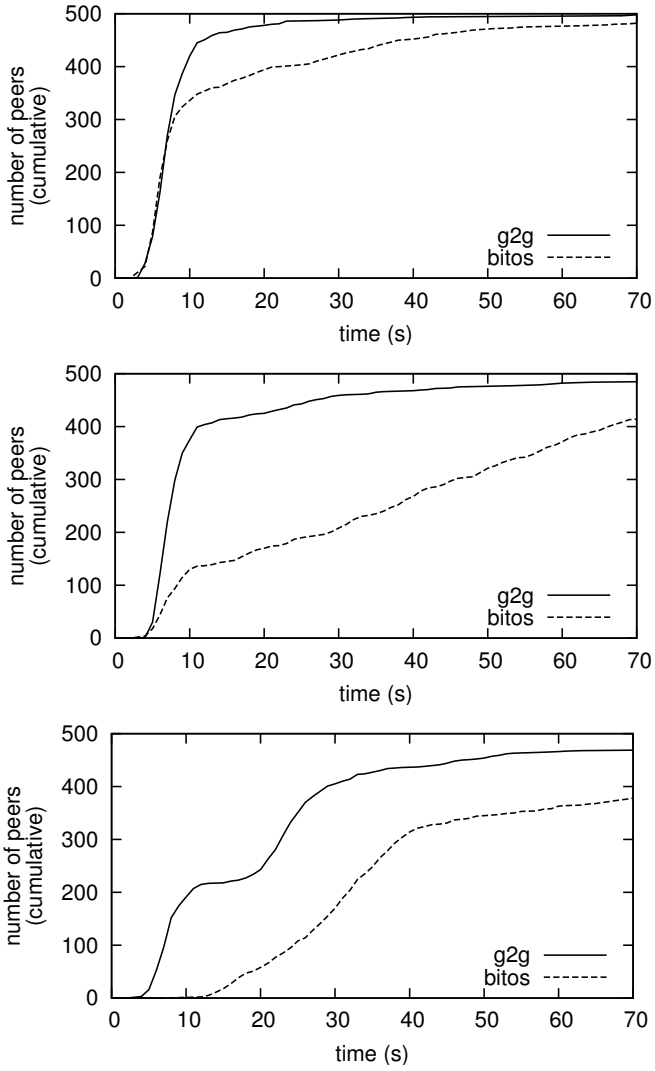


Figure 3.5: The cumulative distribution of the prebuffering time for peers arriving at 0.2/s (top), 1.0/s (middle), and 10.0/s (bottom).

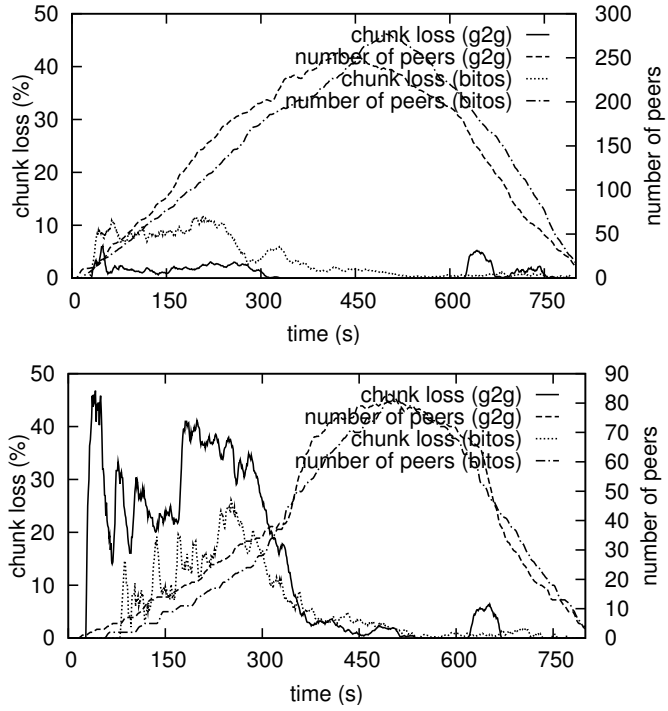


Figure 3.6: The average chunk loss and the number of playing peers for well-behaving peers (top) and free-riders (bottom).

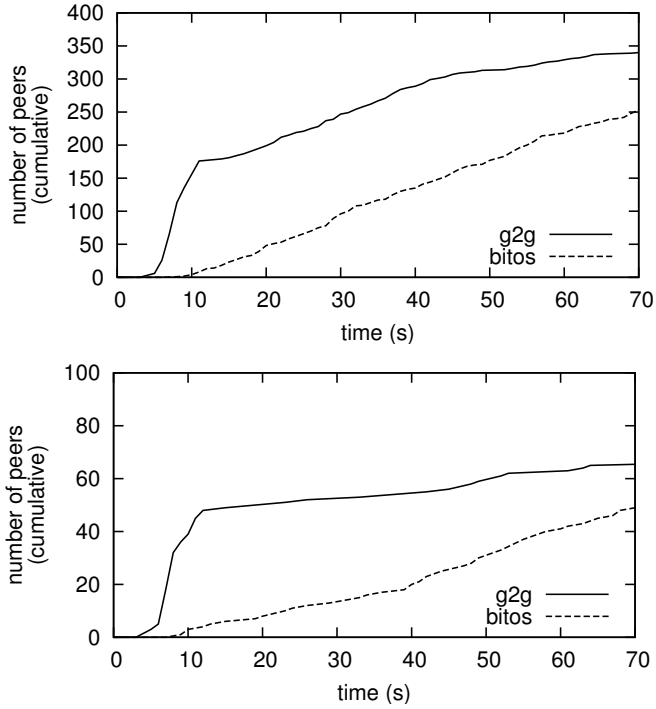


Figure 3.7: The prebuffering time for well-behaving peers (top) and free-riders (bottom).

previous experiment. A slight performance degradation is to be expected, as well-behaving peers occasionally upload to free-riders as part of the optimistic unchoking process. Without any means to detect free-riders before serving them data, losing a bit of performance due to the presence of free-riders is unavoidable. When using G2G, the well-behaving peers lose significantly fewer chunks when compared to BiToS, and free-riders suffer higher amounts of chunk loss.

The required prebuffering times for both groups are shown in Figure 3.7. Both groups require more prebuffering time than in the previous experiment: the well-behaving peers require 33 seconds when using G2G, compared to 14 seconds when free-riders are not present. Because the free-riders have to wait for bandwidth to become available, they either start early and suffer a high chunk loss, or they have a long prebuffering time. In the shown run, the free-riders required 89 seconds of prebuffering on average when using G2G.

3.4 Analysis

The performance of Give-to-Get can be analysed in a generic way by using a fluid model to describe its behaviour. In order to keep the model tractable, the Give-to-Get algorithm needs to be simplified. In this section, we will present a fluid model of a simplified version of the Give-to-Get algorithm, and compare its predictions to simulation results of the complete Give-to-Get algorithm.

The fluid model was designed by Yue Lu and others of the Network Architecture and Services group at Delft University of Technology, in conjunction with the author, who provided simulation results of a simplified version of Give-to-Get to validate the model [63]. Simplified policies make it easier to compare the simulation results to the model, but they make it harder to judge whether the model is a good predictor for the performance of Give-to-Get. We will extend the analysis in [63] by comparing the predictions of the fluid model to simulations of the complete Give-to-Get algorithm. We will first summarise the model, of which a full description can be found in [63]. Then, we will present the set up of both our model parameters and our simulations, and finally, we will present a comparison of the analytical and simulation results under two sets of conditions.

3.4.1 Model Description

The model designed by Lu et al. [63] is similar to the fluid model for file sharing by Qiu et al. [83]. We assume a system in which there exists one initial seeder, and in which peers arrive according to a Poisson process with rate λ. The initial seeder is always online. The other peers do not depart before they have become a seeder. The seeders will depart with a rate of γ, which is determined by the seeding policy. We will consider two seeding policies, which define the amount of time a seeder will seed (which we will call the *seeding time*). Either seeders will remain seeding for a fixed amount of time, or until playback is finished.

We let $x(t)$ and $y(t)$ be the number of leechers and seeders at time t, respectively. The rate at which leechers turn into seeders is determined by the total download speed of the peers. The download speed is either bounded by the available upload bandwidth or the available download bandwidth in the system, whichever is less. If D and U are the download and upload bandwidth of a peer, and S is the upload bandwidth of the initial seeder, then the maximal achievable total download bandwidth in the system is

$s(t) := \min\{x(t)D, (x(t)+y(t))U+S\}$. With the video length, in seconds, denoted by L, and the video bit rate by v, the behaviour of the P2P system can then be modelled as

$$\frac{dx(t)}{dt} = \lambda(t) - \frac{s(t)}{Lv}, \tag{3.1}$$

$$\frac{dy(t)}{dt} = \frac{s(t)}{Lv} - \gamma(t)y(t). \tag{3.2}$$

The model allows the ratio between leechers and seeders to be derived both in the start-up phase and in the steady state, which is characterised by $dx(t)/dt = 0$ and $dy(t)/dt = 0$. The two seeding policies influence the model as follows. If the seeders depart after finishing playback, every peer remains in the system for $B+L$ seconds, which is the duration of the video plus the required prebuffering time B. The sooner a peer obtains the full video, the longer that peer will be a seeder. Let $d(t)$ be the average download speed for the peers at time t, which is equal to

$$d(t) = \frac{s(t)}{x(t)}.$$

Denoting by $T(t)$ the time it takes for a peer to download the video, we have

$$T(t)d(t) \approx Lv,$$

and so the download time can be approximated by

$$T(t) \approx \frac{Lv}{d(t)} = \frac{x(t)Lv}{s(t)}.$$

A peer will thus seed for $B+L-T(t)$ seconds, which leads to a departure rate of

$$\gamma(t) \approx \frac{1}{B+L-T(t)}.$$

The departure rate γ of the seeders thus depends on $x(t)$ and $y(t)$. As a result, Equation 3.2 is non-linear, which complicates the analysis.

If peers remain seeding for a fixed amount of time, we approximate $\gamma(t)$ with the reciprocal of the seeding time. An actual client can show such behaviour by leaving the network after having seeded for a fixed amount of time, even though playback may

Parameter	Symbol	Value
video bitrate	v	0.5 Mbit/s (roughly TV quality)
video length	L	5 minutes (e.g., a short video clip)
upload bandwidth	U	0.9 Mbit/s
upload bandwidth of initial seeder	S	4 Mbit/s
download bandwidth	D	10 Mbit/s
arrival rate	λ	1
prebuffer time	B	10 seconds

Table 3.1: The values of the parameters used for the model validation.

still continue. When $\gamma(t)$ is constant, Equations 3.1 and 3.2 are linear and therefore easy to analyse. If the value of γ is lowered, the seeding time increases, resulting in a higher download speed for the peers. However, the download speed of a peer is limited by its download bandwidth D. It can be shown [63] that the downlinks of the peers will become saturated if

$$\gamma \leq \frac{\lambda U}{\lambda L v - S}. \tag{3.3}$$

Note that the download bandwidth D does not appear in Equation 3.3. Rather, the value of D influences the speed at which a leecher becomes a seeder, and thus the ratio of seeders to leechers. Systems in which the peers have a high download bandwidth D will get a high seeder-to-leecher ratio, which in turn allows the downlinks of the leechers to be saturated. The performance of the system thus depends on the download bandwidth of the peers when Equation 3.3 holds. In systems in which Equation 3.3 does not hold, the performance primarily depends on the upload bandwidth of the peers and the departure rate of the seeders.

For both seeding policies, Lu et al. [63] provide the analysis required to calculate or approximate $x(t)$, $y(t)$, and the download bandwidth of the leechers, averaged for any t, both before and in the steady state.

3.4.2 Model and Simulation Setup

We validate the model for Give-to-Get by using simulations. We limit our comparison by fixing several parameters, as shown in Table 3.1.

Each simulation starts with one initial seeder, and the peers arrive according to

a Poisson process. The simulation results are averaged over 20 runs. We found the average bandwidth utilization of a peer (upload rate/upload capacity) to be equal to 80% of the upload capacity U on average, while the bandwidth utilization rate of the original source is nearly 100%. Hence, in order to be consistent with the settings in our fluid model, we set the original source's upload capacity to $S = 4$ Mbit/s, and a normal peer's upload capacity to $U/0.8 = 1.1$ Mbit/s. We measured the average download speed over all the peers, using a 10 second sliding window. The average download speed will often be lower than expected due to the continuous arrival of peers. A newly arrived peer has a download speed of 0 and requires some time before it has reached its maximum download speed. Since such peers are included in the average, the average download speed over all peers is lowered. Also, the model requires a fixed prebuffering time B which we set to 10 seconds, while Give-to-Get uses its dynamic prebuffering policy. The other parameters are equal to those in the fluid model.

3.4.3 Results for a Non-linear System

In the first seeding policy, peers depart only when the playback is finished. The time each peer seeds, $\gamma(t)$, is not constant, and in fact depends on the number of leechers $x(t)$ and seeders $y(t)$ in the system.

Figure 3.8 illustrates the resulting performance. In the top figure, the number of leechers and seeders are shown over time. The number of leechers initially increases as the peers arrive in the system, but subsequently decreases once leechers become seeders. The total number of peers (leechers and seeders) increases until the departure rate is equal to the arrival rate, which is the case in the eventual steady state. The analytical results and simulation results match well, although the very first seeders appear a bit earlier in the simulations ($t \approx 100s$) when compared to the fluid model ($t \approx 175s$). The reason for this is that the fluid model ignores the upload bandwidth of the initial seeder S in the earliest stages. The peers in the simulations however will have the bandwidth of the initial seeder available to them, and will thus be able to obtain the video slightly faster. Once the steady state is reached, the number of leechers and seeders in the simulations closely match the analytical results.

The bottom figure shows the average download speed over time. Initially, the average download speed is low as peers contend for bandwidth. Once seeders start to appear, the average download speed for the leechers increases significantly. The difference in download speed between the analytical results and the simulation re-

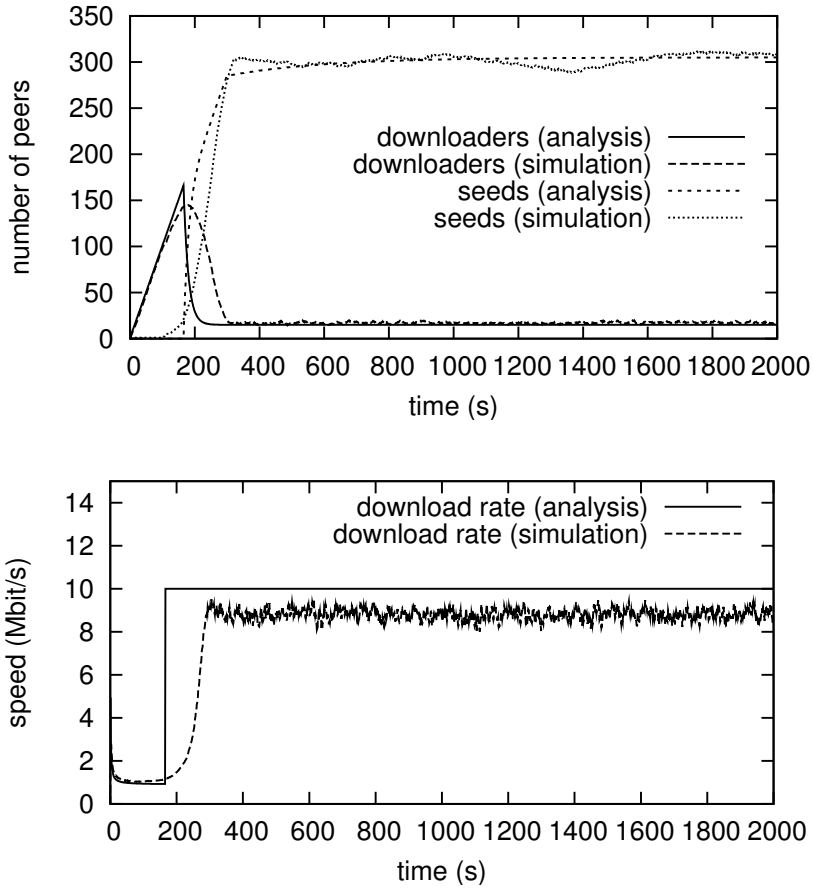


Figure 3.8: The number of leechers and seeders (top) and the average download speed per peer (bottom) as a function of time in a non-linear video system.

sults in the steady state is small, considering that the average download speed in the simulations is lowered by the arrival of new peers.

Other differences between the analytical and simulation results can be caused by the fact that in our fluid model, leechers can always exchange data. In our simulations, leechers can only exchange data if they have obtained different pieces. As a result, the pieces propagate slightly more slowly in the simulations, causing slower as well as more fluent transitions between the states. We conclude that with our settings above, this non-linear system seems to perform very well, and the fluid model is able to predict the behaviour of Give-to-Get with a decent accuracy.

3.4.4 Results for a Linearised System

The fluid model becomes linear when each peer seeds for a fixed amount of time. We use the same values for our parameters, except for the value of γ. Depending on the value of γ, the performance is either limited by the upload or the download bandwidth of the peers, as can be seen from Equation 3.3. If the value of γ is low enough, the seeds remain in the system long enough to saturate the download bandwidth of the peers. Lowering γ beyond that point cannot increase the download speed of the peers. Equation 3.3 provides us with the threshold value, which is $\gamma \approx 0.00616$ for our parameter values. For higher values of γ, the seeders depart earlier, and therefore provide less upload capacity to the system, making the system upload-bandwidth constrained. We compare our model to simulations for both $\gamma = 0.006$ (a balanced system) and $\gamma = 0.008$ (an upload-bandwidth constrained system) in order to show the performance of the system on the treshhold, and when peers seed for a shorter amount of time. We will show that the peers will be able to obtain a decent average download speed in the balanced system, making it unnecessary to let them seed for a longer period of time ($\gamma < 0.006$).

In a balanced system, each user shares the video for $1/\gamma = 1/0.006 \approx 167$ seconds after the download is finished, regardless of the download speed. Figure 3.9 shows the resulting performance. We observe that this linear case has a similar system performance as the non-linear system, because the average download speed at a peer is similar, even though the number of seeders in this case is smaller than in the steady-state. The difference between the analytical results and the simulation results are similar to those in the non-linear system. So, if the peers seed for 167 seconds or more, the average download speed will always be maximised. Therefore, the model

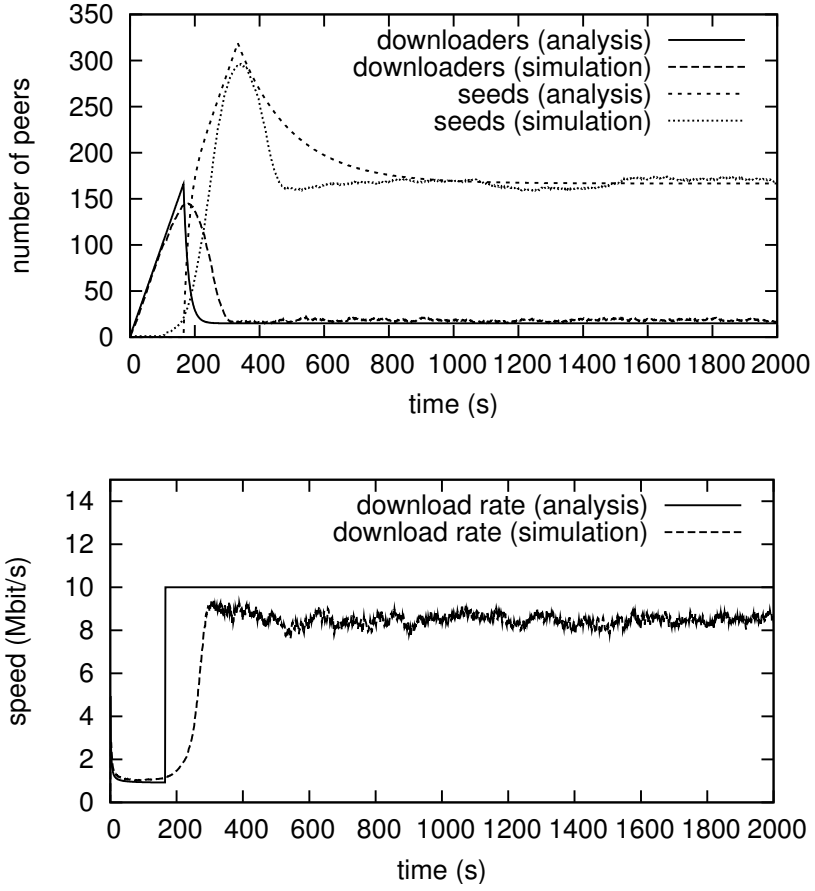


Figure 3.9: The number of downloaders and seeders (top) and the average download speed per peer (bottom) as a function of time in a balanced, linearised system ($\gamma = 0.006$).

allows a Give-to-Get developer to determine when maximum system performance is reached.

In an upload-constrained system with $\gamma = 0.008$, each user shares the video for $1/0.008 = 125$ seconds after he finishes the download. The situation of the linear system in this case is shown in Figure 3.10. Since the system is above the $\gamma = 0.006$ threshold, it is the upload capacity of the peers that limits the system performance. The seeders cannot provide enough capacity to maximise the download speed of the leechers, which leads to worse system performance. Figure 3.10 (bottom) shows the average download speeds, which originally behave in the same way as for $\gamma = 0.006$, because both configurations are the same as long as no seeder has departed yet. Once the seeders start to depart, fewer of them will be in the system when compared to $\gamma = 0.006$, as can be seen in the top figure. The download speed of the leechers rebounds slightly as the leechers necessarily stay longer in the system due to a lack of seeders, and thus provide each other with more upload capacity.

In all of the cases we tested, the trends of the simulation results and the analytical results are similar. We have thus validated the fluid model proposed in [63], and have shown that the model can be used to predict the performance of Give-to-Get. The model allows developers to calculate, for example, the length and the bit rate of the video that the system will be able to stream at acceptable quality.

3.5 Related Work

Video-on-demand is a popular service on the Internet, which is usually provided by Content Delivery Networks (CDNs) [77, 96]. A CDN is a client-server architecture, in which the server that provides the video stream is replicated at multiple sites. Websites that host video clips or live video streams, such as YouTube [5], make use of a CDN in order to be able to deliver the bandwidth needed to serve all of their users.

An early algorithm for distributed VoD is Chaining [90], in which the video data is distributed over a chain of peers. Such a solution works well for a controlled environment, but is less suitable for P2P networks. In P2Cast [47], P2VoD [33] and OBN [60], peers are grouped according to similar arrival times. The groups forward the stream within the group or between groups, and turn to the source if no eligible supplier can be found. In all three algorithms, a peer can decide how many children it will adopt, making the algorithms vulnerable to free-riding.

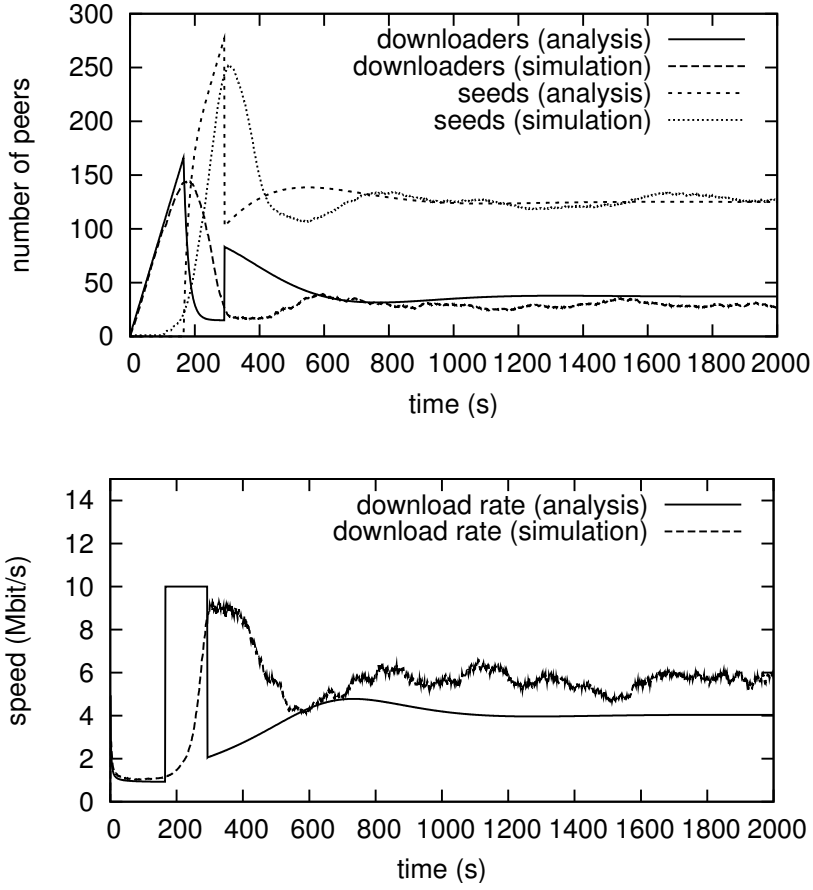


Figure 3.10: The number of downloaders and seeders (top) and the average download speed per peer (bottom) as a function of time in an upload-constrained linearised system ($\gamma = 0.008$).

Our G2G algorithm borrows the idea of bartering for chunks of a file from BitTorrent [26], which is a very popular P2P protocol for off-line downloading. In BitTorrent, the content is divided into equally sized chunks which are exchanged by peers on a tit-for-tat basis. In deployed BitTorrent networks, the observed amount of free-riding is low [11] (however, Locher et al. [61] have shown that with a specially designed client, free-riding in BitTorrent is possible without a significant performance impact for the free-riding peer). Of course, the performance of the network as a whole suffers if too many peers free-ride. The BitTorrent protocol has been adapted for VoD by both BASS [28] and BiToS [98]. In BASS [28], all peers connect to a streaming server, and use the P2P network to help each other in order to shift the burden off the source. BASS is thus a hybrid between a centralised and a distributed VoD solution. The differences between G2G and BiToS are described in Section 3.2.4. Annapureddy et al. [12] propose a comprehensive VoD protocol in which the video stream is divided into segments, which are further divided into blocks (chunks). Peers download segments sequentially from each other and spend a small amount of additional bandwidth prefetching the next segment. Topology management is introduced to connect peers that are downloading the same segment, and network coding is used within each segment to improve resilience. However, their algorithm assumes peers are honest and thus free-riding is not considered. Using indirect information as in G2G to judge other peers is not new. EigenTrust [40] defines a central authority to keep track of all trust relationships in the network. Lian et al. [59] generalise the use of indirect information to combat collusion in tit-for-tat systems.

As is common in streaming video players, G2G buffers the beginning of the video clip (the prefix) before playback is started. Sen et al. [89] propose a scheme to let proxy servers do this prefix caching for a centralised VoD service, thus smoothing the path from server to client.

3.6 Conclusions

We have presented Give-to-Get (G2G), a P2P VoD algorithm that discourages free-riding by rewarding peers which forward data to others. Peers are encouraged to compete to forward as much data as possible, since the peers which forward the most data will be provided with a better quality of service by their neighbours. Also, we have introduced a novel chunk-picking policy, in which the set of chunks required for playback is divided into three subsets: a high-, a mid-, and a low-priority set.

Such a division allows an easy-to-implement yet graceful transition between downloading pieces required on the short term (the high-priority set) and those required on the long term (the low-priority set), with a distinct chunk picking policy within each priority set. Although we have extended the BitTorrent protocol to incorporate these ideas, both our forwarding incentives and our chunk picking policies can be of use to improve the performance and free-riding resilience of other P2P VoD algorithms as well.

We evaluated the performance of Give-to-Get by conducting tests under various conditions as well as comparing its performance to that of BiToS [98]. If free-riders are present, they suffer heavy performance penalties if upload bandwidth in the system is scarce. Once enough peers have downloaded the full video, there is enough upload bandwidth to sustain free-riders without compromising the well-behaving peers.

We also compared simulations of Give-to-Get to a fluid model for P2P VoD [63]. The fluid model was shown to provide a good estimation of the performance of Give-to-Get under the conditions we considered. The model can thus be used to predict the performance of a deployed Give-to-Get system.

With the Give-to-Get algorithm, we have shown that the effects of free-riding can be significantly reduced in a peer-to-peer video-on-demand distribution system. Our approach relies on an identity system to prevent peers from colluding, but does allow peers with currently common asymmetric Internet connections to obtain video of a higer quality by encouraging all peers to upload as much data as possible. Furthermore, because we extended the existing BitTorrent protocol for which many clients already exist, Give-to-Get is easy to implement and to deploy.

Chapter 4

Swarm-based Live Streaming

THE SWARM-BASED APPROACH for video distribution is a practical one, as several swarm-based algorithms have been succesfully deployed on the Internet, with BitTorrent being the most prominent. Even though most of these algorithms focus on file sharing, they can be extended to support video streaming, as we did with BitTorrent in Chapter 3 for video-on-demand (VoD). In this chapter, we will propose extensions for BitTorrent to support live video streaming. The advantages of extending BitTorrent are three-fold. First, BitTorrent has several optimised and well-tested implementations. Extending such implementations reduces the number of bugs and increases the performance of the features offered. Second, any enhancements to the original BitTorrent protocol (for example, firewall puncturing and DHT peer discovery) carry over to the extensions with little or no extra effort. Finally, along with the Give-to-Get extensions for video-on-demand of Chapter 3, the BitTorrent protocol will support all three modes of video distribution — live video streaming, video-on-demand, and off-line downloading — within the same environment.

Although the difference between video-on-demand and live video streaming may seem minor from a user's perspective, the technological differences are actually fundamental when considering a BitTorrent-like system. First, both the original BitTorrent protocol and its VoD extensions assume all data to be available beforehand. The total length of the data as well as hashes of it are generated before the file is offered for download. Such information is not available in a live streaming session. The hashes of the data can only be calculated when the data have been generated, and the length of the stream is unknown in many scenarios (for example, when broadcasting TV

channels). Secondly, in contrast to VoD, peers in live streaming are always playing approximately the same part of the stream. All peers thus require roughly the same data, and cannot download faster than the stream is generated. Finally, when in VoD peers finish downloading the complete stream, they can share it with others (called *seeding*), which massively increases the download performance of the other peers. We have previously observed systems in which 90% of the peers are seeding. In live streaming, no peer is ever done downloading, so no seeds exist. We will provide a solution to these issues, and we will present an algorithm that allows a BitTorrent system to stream live video.

We will evaluate the feasibility of our approach by using simulations as well as a deployed system. Because it is difficult to capture the full dynamics of a deployed P2P system through simulations, we mainly use simulations to estimate optimal values for the parameters that are used in our extensions (e.g., the prebuffering time, which is the time used for the initial downloading of video data before the playback starts). We also focus on varying the percentage of peers in the system that are behind firewalls or NATs. In Chapter 5, we will show this percentage to be a key factor in the performance of P2P content distribution systems. The deployment of our extensions consisted of a public trial using our Tribler BitTorrent client [80]. We deployed an outdoor DV camera, and invited users to tune in using Tribler. Over the course of 9 days, 4555 unique users participated in our trial.

This chapter is organised as follows. First, we will briefly describe BitTorrent and discuss related work in Section 4.1. We will present our extensions to BitTorrent in Section 4.2. Section 4.3 contains the set up and results of our simulations. Section 4.4 will describe the set up and results of our public trial. Finally, we will draw conclusions in Section 4.5.

4.1 Background

In this section, we will present the background of our live video streaming extensions of BitTorrent. First, we will briefly describe the BitTorrent protocol as it forms the basis of our work; a more extensive description can be found in [27]. Then, we will discuss the differences between our approach and those of other live video streaming systems.

4.1.1 BitTorrent

The BitTorrent protocol [27] operates as follows. The injector creates a *torrent* metadata file, which is shared with the peers through, for example, an HTTP server. The file to be shared is cut into fixed-size pieces, the hashes of which are included in the torrent file. A central server called a *tracker* keeps track of the peers in the network. The peers exchange data on a tit-for-tat basis in the *swarm* of peers interested in the same file: the neighbours of a peer that provide the most data are allowed to request pieces in return (they are *unchoked*). The tit-for-tat construct thus creates an incentive for peers to upload their data to others. Once a peer has completed the download of a file, it can continue to *seed* it by using its upload capacity to serve the file to others for free.

Mature BitTorrent clients extend the basic BitTorrent protocol, for instance by including support for a global DHT ring, which allows peers to locate each other without requiring to contact the tracker. This DHT ring in particular could be used as a basis for an application-level multicast tree. However, such a multicast tree would not benefit from any tit-for-tat mechanism, and would thus be vulnerable for abuse.

4.1.2 Related Work

Many of the P2P live streaming algorithms proposed in the literature are designed to operate in a cooperative environment. Abusing such algorithms on the Internet is easy [64]. The BAR Gossip [58] and Orchard [68] algorithms do not assume full cooperation from the peers, but the P2P framework in which they operate is out of the scope of either algorithm, nor do they include measurements of a deployed system.

Deployed live streaming P2P solutions have been measured before [7, 9, 100] and reveal a decent performance. Ali et al. [9] and Agarwal et al. [7] measure the performance of commercial P2P live streaming systems, but they do not describe the P2P distribution algorithm that was used. Xie et al. [100] measure a deployed and open protocol called Coolstreaming [103]. Our approach is different in that we extend BitTorrent, which can be deployed in environments in which peers do not necessarily behave well. We will compare our results with those found by Agarwal et al. [7] and Xie et al. [100] to provide insight into our results.

4.2 Extensions for Live Streaming

Our live streaming extensions to BitTorrent use the following generic setup. The injector obtains the video from a live source, such as a DV camera, and generates a *tstream* file, which is similar to a torrent file but cannot be used by BitTorrent clients lacking our live streaming extensions. An end user (peer) which is interested in watching the video stream obtains the tstream file and joins the swarm (the set of peers) as per the BitTorrent protocol. We regard the video stream to be of infinite length, which is useful when streaming a TV channel or a webcam feed.

We identify four problems for BitTorrent when streaming live video, which are unknown video length, unknown future video content, a suitable piece-selection policy, and the lack of seeders, which we will discuss in turn.

4.2.1 Unlimited Video Length

The BitTorrent protocol assumes that the number of pieces of a video is known in advance. This number is used throughout a typical implementation to allocate arrays with an element for every piece, and therefore, it is not practical to simply increase the number of pieces to a very large value that will not be reached (for instance, 2^{32}). Instead, each peer maintains its own sliding window that rotates over a fixed number of pieces at the speed of the video stream. Each peer deletes pieces that fall outside of its sliding window, and will consider them to be deleted at its neighbours as well, thereby avoiding the need for additional messages. Within its sliding window, each peer barters for pieces according to the BitTorrent protocol.

The injector defines its sliding window as the A most recently generated pieces. The value of A is included in the tstream file and is thus known to all peers. Any other peer is not necessarily connected to the injector, and synchronises its sliding window with the pieces available at its neighbours. For any peer, let piece p be the youngest piece available at more than half of its neighbours. Then, its sliding window is $(p-A, p+A]$. Figure 4.1 illustrates the relationship between the sliding windows of the peers. We allow the playback position, which we will define in Section 4.2.3, of any peer is to be no more than A pieces behind the latest generated piece. A peer can verify this by comparing its clock to the timestamps which we will include in the pieces. The playback position of any peer thus falls within the sliding window of the injector, which allows the injector to serve any peer if all others depart. Furthermore,

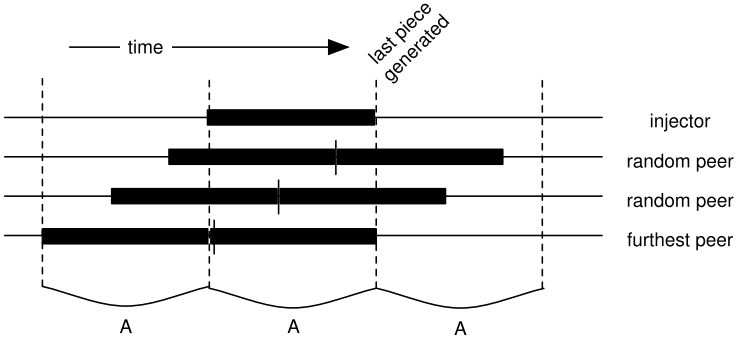


Figure 4.1: The sliding windows of valid pieces for the injector, two random peers, and the peers furthest from the injector in terms of delay. The vertical markers on each bar indicate the peer's playback position.

the sliding window of one peer cannot overlap with overlap with the sliding window of another peer due to a wrap around over the total set of pieces.

A peer thus has to be able to determine which piece is the youngest available. In order to make that possible, we let the total number of pieces be $4A$. Each neighbour has a sliding window of size at most $2A$, and will subsequently report at least $2A$ consecutive missing pieces. The last piece reported before a missing range of at least $2A$ pieces is thus the youngest piece owned by that neighbour. Neighbours that report to have no pieces at all, are ignored. We use majority voting to be able to ignore malfunctioning and malicious clients, which may report pieces that are outdated, or that they do not own.

4.2.2 Data Validation

Once a peer completes the download of any piece, it proceeds to validate the data. The ordinary BitTorrent download protocol computes a hash for each piece and includes these hashes in the torrent file. In the case of live streaming, these hashes have to be filled with dummy values or cannot be included at all, because the data are not known in advance and the hashes can therefore not be computed when the torrent file is created. The system would thus be left prone to injection attacks and data corruption.

We prevent data corruption by using asymmetric cryptography to sign the data, which has been found by Dhungel et al. [31] to be superior to several other methods

for data validation in live P2P video streams. The injector publishes its public key by putting it in the torrent file. Each piece includes an absolute sequence number, starting from zero, and a time stamp at which the piece was generated. The SHA1 hash of each piece is signed by the injector, and the signature is appended to the piece. Any peer can thus check whether a piece was generated by the injector. The sequence number allows a peer to confirm that the piece is recent, since the actual piece numbers are reused by the rotating sliding window. The time stamp allows a peer to determine the propagation delay from the injector. We use 64-bit integers to represent both the sequence numbers and the timestamps, so a wrap around will not occur within a single session.

In our case, the signature adds 64 bytes of overhead to each piece, and the sequence number and timestamp add 16 bytes in total. If a peer downloads a piece and detects an invalid signature or finds it outdated, it will delete the piece and disconnect from the neighbour that provided it. A piece is outdated if was generated A pieces or longer ago.

The validity of a piece can only be verified once it has been downloaded completely, and only verified pieces are offered to others and to the video player. As a result, small pieces reduce the delay of each hop traversed by the piece. On the other hand, if pieces are too small, the overhead of dealing with an increased number of pieces increases as well.

4.2.3 Live Playback

Before a peer starts downloading any video data, it has to decide at which piece number it will start watching the stream, which we call the *hook-in point*. We assume each peer desires to play pieces as close as possible to the latest piece it is able to obtain. However, if the latest pieces are available at only a small number of neighbours, downloading the stream at the video bitrate may not be sustainable. We therefore let a peer start downloading at B pieces before the latest piece that is available at at least half of its neighbours. The *prebuffering phase* starts when the peer joins the network and ends when it has downloaded 90% of these B pieces. We do not require a peer to obtain 100% of these pieces, to avoid waiting for pieces that are not available at its neighbours or that take too long to download. The *prebuffering time* is defined as the length of the prebuffering phase.

The pieces requested from a neighbour are picked on a rarest-first basis, as in Bit-

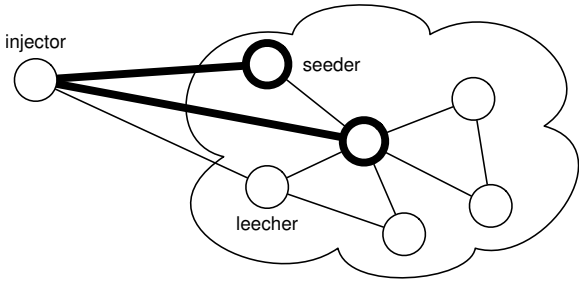


Figure 4.2: The injector, seeders and leechers in a live streaming setting. The bold nodes are seeders. The bold edges are connections which are guaranteed to be unchoked.

Torrent. This policy encourages peers to download different pieces and subsequently exchange them. If peers would request their pieces in-order, as is common in video streaming algorithms, peers would rarely be in the position to exchange data, which would conflict with the tit-for-tat incentives in BitTorrent.

A buffer underrun occurs when a piece i is to be played but has not been downloaded. Since pieces are downloaded out of order, the peer can nevertheless have pieces after i available for playback. If a peer has more than $B/2$ pieces available after i, the peer will drop missing piece i. Otherwise, it will stall playback in order to allow data to be accumulated in the buffer. Once more than $B/2$ pieces are available, playback is resumed, which could result in dropping piece i after all if it still has not been downloaded. We will use a value of B equivalent to 10 seconds of video in our trial, a value which we derive through simulation in Section 4.3.3.

We employ this trade off since neither dropping nor stalling is a strategy that can be used in all situations. For instance, if a certain piece is lost because it was never available for download, it should be dropped, and playback can just ignore the missing piece. On the other hand, a peer's playback position can suddenly become unsustainable due to network dynamics. A different set of neighbours may only be able to provide the peer with older data than it needs. In that case, a peer should stall playback in order to synchronise its playback position with its new neighbours.

4.2.4 Seeders

In BitTorrent swarms, the presence of seeders significantly improves the download performance of the other peers (the leechers). However, such peers do not exist in a

live streaming environment as no peer ever finishes downloading the video stream.

We redefine a *seeder* in a live streaming environment to be a peer which is always *unchoked* by the injector and is guaranteed enough bandwidth in order to obtain the full video stream.The injector has a list of peer identifiers (for example, IP addresses and port numbers) representing trusted peers which are allowed to act as seeders if they connect to the injector. The relationship between the injector, seeders and leechers is shown in Figure 4.2. The seeders and leechers use the exact same protocol, but the seeders (bold nodes) are guaranteed to be unchoked by the injector (bold edges). The identity of the seeders is not known to the other peers to prevent malicious behaviour targeted at the seeders.

The injector and seeders behave like a small Content Delivery Network (CDN). Even though the seeders diminish the distributed nature of the BitTorrent algorithm, they can be required if the peers cannot provide each other with enough bandwidth to receive the video stream in real-time. The seeders boost the download performance of other peers as in regular BitTorrent swarms. Also, the seeders increase the availability of the individual pieces, which reduces the probability of an individual piece not being pushed into the rest of the swarm. We will measure the effect of increasing the number of seeders through simulation in Section 4.3.5.

The maximum video rate that can be streamed succesfully by the peers is limited by the bandwidth they have available as well as by the connectivity between them. The number of peers that can be trusted to be used as seeders depends on factors that lie outside of the scope of this chapter, such as the presence of a trust or reputation system that keeps track of the behaviour of the peers accross sessions. However, if the injector has seeders at its disposal, either by deploying them himself or by promoting altruistic peers, the quality of the video for the peers is improved and streaming higher video bitrates becomes possible.

4.3 Simulations

We have written a BitTorrent simulator and extended it with our algorithm to evaluate the performance of our extensions. In this section, we will explain the setup of our simulations, followed by the evaluation of the impact of four parameters: the capacity of the uplink of the peers, the hook-in point, the piece size, and the number of seeders.

4.3.1 Simulation Setup

We do not aim to reproduce the exact same conditions in the simulations that will occur in our trial. Instead, we use the simulator to implement a reasonable scenario in order to test the impact of several parameters defined throughout this paper. In each simulation run, we simulate 10 minutes of time, and let peers arrive with an average rate of one per second (using a Poisson distribution) to share a video stream of 512 Kbit/s using BitTorrent pieces of 32 Kbyte, which amounts to two pieces per second. Each peer has a residence time of 0 – 10 minutes. For each set of parameters (data point) we evaluate, we performed ten simulation runs. The peers have an asymmetric connection with an uplink of 1–1.5 Mbit/s and a downlink of four times the uplink. The latency between each pair of peers is 100 – 300 ms. We will show in Chapter 5 that the percentage of peers that cannot accept incoming connections (due to the presence of a firewall or NAT) can have a significant impact on the performance, especially if this percentage is higher than 50%. The BitTorrent protocol does not define any firewall puncturing or NAT traversal techniques, and due to their complexity we consider both techniques to be outside the scope of our extension. Therefore, we repeat our simulations, with every peer having a probability of 0%, 50%, 60%, 70%, 80%, or 90% of not accepting incoming connections. All peers can, however, initiate outgoing connections and transfer data in both directions if the connection is accepted. To keep our model simple and for ease of discussion, we will consider any peer which cannot accept incoming connections to be firewalled, and vice versa. Peers not behind a firewall will be called *connectable*.

4.3.2 Uplink Bandwidth

The amount of available upload capacity of the peers is one of the critical factors for the performance of a P2P live video streaming algorithm. Regardless of the streaming algorithm, the peers (including the injector and the seeders) need at least an average upload bandwidth equal to the video bitrate in order to serve the video stream to each other without loss. In practice, more upload bandwidth is needed as not all of the available upload bandwidth in the system can be put to use. Even though one peer may have the data and the capacity to serve another peer that needs those data, the peers may not know about each other, or may be barred from connecting to each other due to the presence of firewalls.

Using our simulator, we tested the impact of the average available uplink band-

width. For each simulation, we define an average uplink bandwidth u for the peers. Each peer is given an uplink between $0.75u$ and $1.25u$. As our performance metric, we use the total amount of data played by all peers, instead of the more common metrics such as piece loss and prebuffering time. We do this for two reasons. First, we need a single metric to compare the performance between simulations. The piece loss in a system can be lowered by increasing the prebuffering time, and vice-versa, so performance cannot be accurately described by either metric alone. The total amount of played data is a single metric which avoids such a trade-off, since both an increased piece loss and an increased prebuffering time lower the amount of data played. Secondly, describing the piece loss figures for a set of peers in a single number is problematic, as the fraction of loss is easily skewed by peers which remain in the system for only a brief period of time. When considering the total amount of data played by all peers, peers which remain in the system only briefly have a lower impact.

Our metric has two downsides as well. First of all, for piece loss and prebuffering time figures, their optimal values are clear. However, for the total amount of data played, the optimal value depends on the arrival and departure rates of the peers. We therefore use the same arrival and departure patterns when testing different parameters. The best average performance over the ten patterns we see across our tests is used as our base line, and assign it a value of 1. In that case, we found 4.3 seconds of prebuffering time and $<< 0.01\%$ pieces loss. The results of all other tests are given relative to this base line. Secondly, since our metric captures both piece loss and prebuffering time, the difference between them cannot be distinguished. If the total amount of data played is low, one cannot tell whether there was a lot of piece loss, or whether peers took a long time before they started playing. Also, for long simulations, an acceptable piece loss can accumulate to high values which would not be acceptable if the same amount of loss was caused by an exceedingly long prebuffering time.

Figure 4.3 plots the total amount of data played as it depends on the available uplink bandwidth normalised with regard to the video bitrate (512 Kbit/s). As expected, the performance is poor when the peers have an average uplink smaller than the video bitrate. If there are no firewalled peers, the peers need roughly 1.5 times the video bitrate as their uplink in order to obtain the highest performance. However, once more than half of the peers are firewalled, the performance drops significantly and more bandwidth is needed. This result is consistent with our earlier findings on P2P data exchange in general [69].

In the remaining simulations, we give peers an uplink of 1 – 1.5 Mbit/s, which is

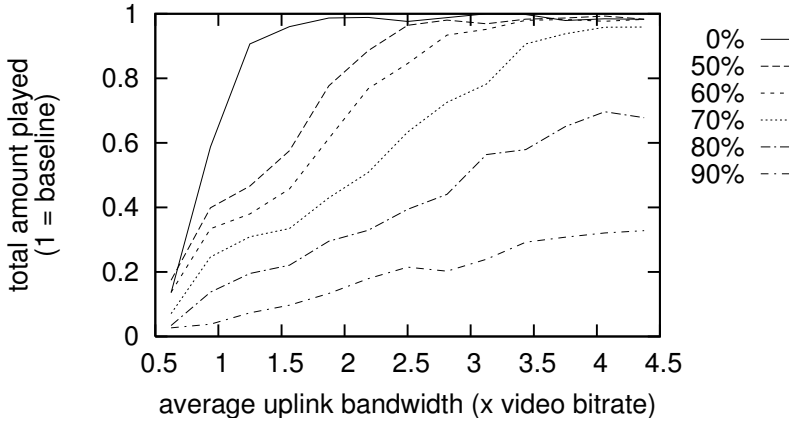


Figure 4.3: The total amount of data played versus the average uplink bandwidth, for several percentages of firewalled peers.

equal to 2.5 times the video bitrate, on average. We thus expect to provide acceptable performance if at most 60% of the peers is behind a firewall.

4.3.3 Hook-in Point

We measure the effect of changing the hook-in point by varying the size B of the buffer of the prebuffering phase introduced in Section 4.2.3 in a system with no firewalled peers. Figure 4.4 plots the measured performance. The average required prebuffering time is plotted against the buffer size B, as well as the average percentage of time a peer spent either losing pieces or stalling. As the size of the buffer B increases, the prebuffering time increases as more pieces have to be downloaded before playback can start. The amount of time spent by a peer losing pieces or stalling decreases when B increases, which is explained by the fact that a larger buffer provides each peer with more time to obtain missing pieces. A value for B of 10 seconds (= 20 pieces) is a good trade off, and this is the value we will use in our trial (see Section 4.4).

4.3.4 Piece Size

Figure 4.5 plots the performance obtained in our simulations when the size of the BitTorrent pieces varies. Each line represents a different percentage of peers behind a firewall. Again, the performance of the system degrades as the percentage of fire-

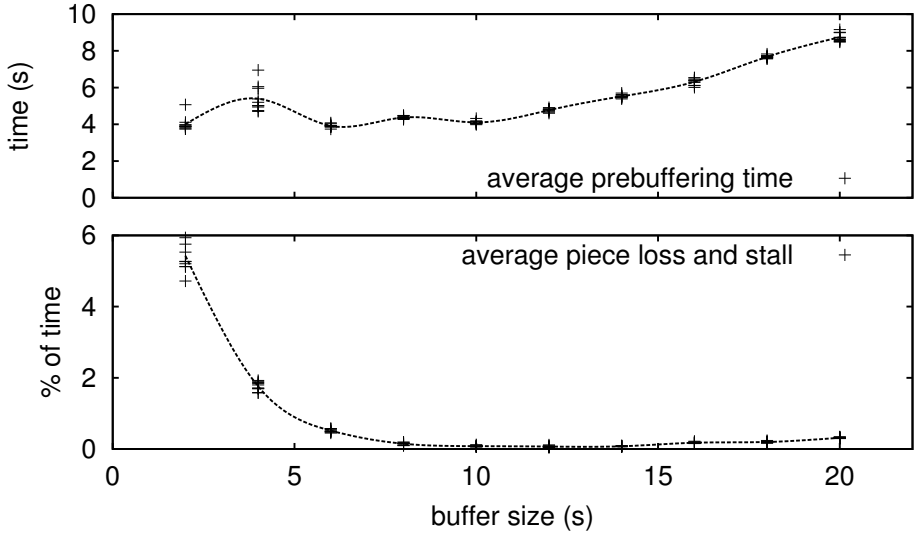


Figure 4.4: The average prebuffering time (top) and the average percentage of time spent losing pieces or stalling (bottom) versus the size of the buffer of the prebuffering phase.

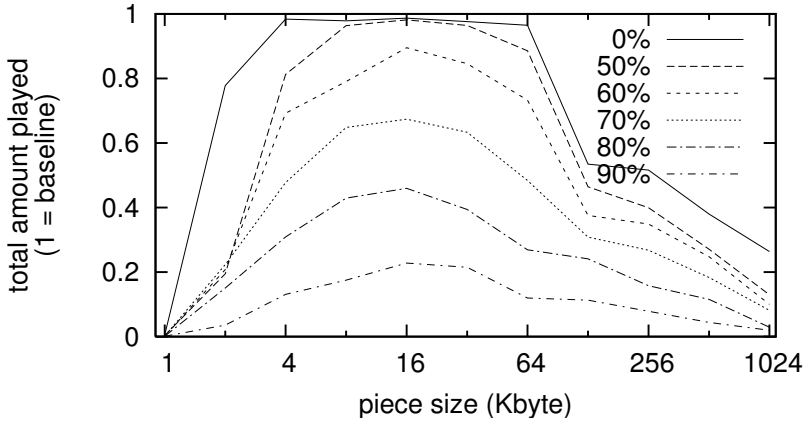


Figure 4.5: The total amount of played data versus the piece size, for several percentages of firewalled peers.

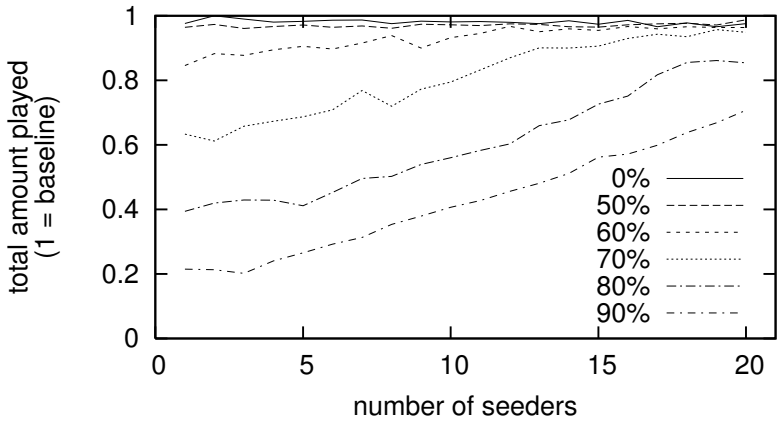


Figure 4.6: The total amount of played data versus the number of seeders, for several percentages of firewalled peers.

walled peers increases. Furthermore, both small and large piece sizes provide poor performance. Very small pieces introduce too much overhead, and large pieces require too long to download, or may never even be downloaded completely before their playback deadline. According to our simulations, the best performance is obtained using pieces of 16 Kbyte or 32 Kbyte in size. In our trial, we will use pieces of 32 Kbyte, since the 16 Kbyte simulations were not available at the start of the trial.

4.3.5 Number of Seeders

Finally, we test the impact of the number of seeders. The results of these tests are shown in Figure 4.6. We performed simulations with 1 to 20 seeders per swarm (against approximately 600 peers in total as one peer arrives every second for a period of 10 minutes), and with varying percentages of peers behind firewalls. Figure 4.6 indicates that increasing the number of seeders significantly increases performance, in case tha performance is poor due to a high percentage of peers behind firewalls.

4.4 Public Trial

In this section, we will first describe how our trial was set up. Then, we will discuss the main performance characteristics: the size of the swarm, the amount of piece loss and

stalling experienced by the peers, the prebuffering time they needed, and their sharing ratios. The sharing ratio of a set of peers is the ratio of the number of uploaded bytes and the number of downloaded bytes, which is a metric for their resource contribution and for the scalability of our solution.

4.4.1 Trial Setup

We have developed a P2P video player called the *SwarmPlayer*, which is based on the Tribler [80] core and the VideoLan Client (VLC) [4] video player. It runs a mature BitTorrent implementation plus our proposed extensions. In a public trial on 17th – 26th July 2008, we invited users to watch a specific live video stream using the SwarmPlayer. Over the course of 9 days, we saw 4555 users from around the globe tune in.

We were not able to obtain the rights to stream popular copyrighted content to a world-wide audience, which would attract a lot of viewers. Instead, we used an outdoor DV camera, aimed at the Leidseplein, which is a busy square in the center of Amsterdam. We transcoded the live camera feed to a 512 Kbit/s MPEG-4 video stream. The video stream was wrapped inside an MPEG Transport Stream (TS) container to make it easier for clients to start playing at any offset within the byte stream, and cut into 32 KByte BitTorrent pieces. Our extensions for live streaming were configured with a value of A (the window size of the injector) of 7.5 minutes, and with a value of B, the size of the buffer in the prebuffering phase, of 10 seconds. We deployed an injector as well as five seeders. The performance results in the following sections will exclude these peers.

To participate in our trial, a user had to download the SwarmPlayer and the tstream file representing our DV camera. Once the SwarmPlayer was started, it sent status updates to our logging server at 30 second intervals. The logging server checked for each peer whether it is firewalled by trying to connect to it. UPnP remote firewall control is supported.

We will compare the results of our trial with two other measurements of deployed live P2P streaming systems. The first, by Xie et al. [100], measures a deployment of the open protocol Coolstreaming [103] on up to 40,000 concurrent peers spread over an unknown number of swarms, delivering a 768 Kbit/s stream. The second, by Agarwal et al. [7], measures the deployment of a closed protocol on up to 60,000 concurrent peers within a single swarm, delivering a 759 Kbit/s stream. These mea-

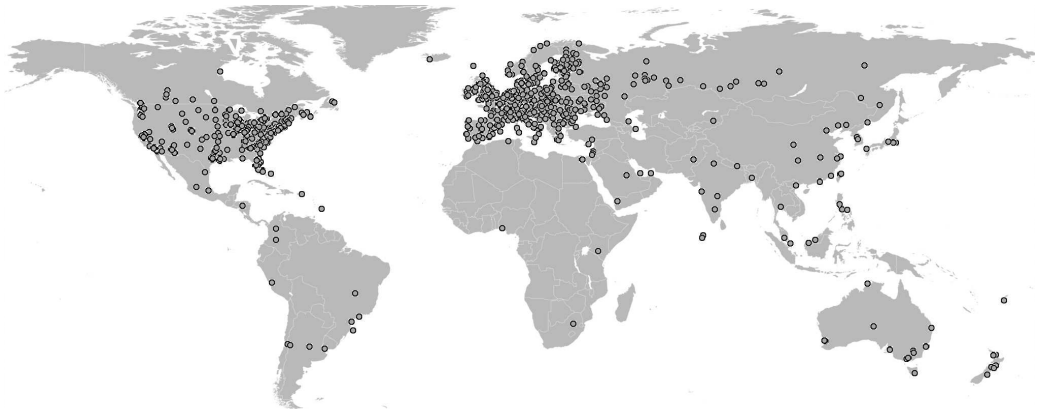


Figure 4.7: The locations of all users.

surements assume a different set up and measure a different set of users. Nevertheless, they will provide a source of comparison that allows us to interpret our results.

4.4.2 Performance over Time

During the trial, users from 4555 different IP addresses joined our swarm 6935 times in total. Figure 4.7 shows the locations of all the users that participated. These locations were derived from the IP addresses using publicly available databases. Our injector and seeders are located in the Netherlands.

Figure 4.8 plots the size of the swarm over time (subfigure (a)) as well as the average performance of the peers (subfigures (b)–(d)). Each data point is the average for an hour to improve readability. All time stamps are GMT.

The public trial ran from Thursday July 17th 2008 until Saturday July 26th 2008. Several events occurred during the course of the trial which have been marked in Figure 4.8. The trial was announced through several media (Ars Technica, BBC News, and others) before and after the first weekend. As the time line shows, the trial was not without problems. On Saturday 19th, a new SwarmPlayer had to be released which fixed a few bugs that we will mention later. On Tuesday 22nd, our DV camera went briefly off line, causing all peers to freeze playback. Once the camera was plugged back in, the system resumed without requiring any intervention. The injector itself ran uninterrupted during the whole trial.

The size of the swarm (Figure 4.8(a)) varies between 0 and 86 peers, although only up to 76 peers stayed long enough to be readable in the figure. Although we would have liked to see more concurrent users, repeating the trial was not feasible for us. Most of the peers arrived after the press release on Friday 18th, and a boost in the arrival rate can be clearly seen on Monday 21st, when the second press release was made. On most of the days, the swarm is at its smallest at night time. The contrast between the high number of arrivals (6935) and the rather small swarm indicates that most visits are brief. Indeed, Figure 4.9 plots the duration of all sessions from long to short. The median session duration is 100 seconds, with 560 sessions lasting longer than an hour. The minimum session duration we could measure is 30 seconds, because that is the interval with which peers send status reports to our logging server.

Figure 4.8(b) shows the average bitrate of pieces that were received on time and transferred to the video player. Two aspects are of interest. First, the low bit rates at night and the spikes at dawn are artefacts of our video encoder. Second, there is a drop in playback rate on Saturday 19th, which was caused by client malfunction. We released a new version of the SwarmPlayer the same day, and the system resumed its operation.

Figure 4.8(c) shows the percentage of pieces lost. Most of the piece loss occurred on Saturday 19th as is to be expected. Overall, the rest of the piece loss is low, and is mostly concentrated on a few peers, as is shown in Figure 4.10, which shows the percentage of pieces lost within the individual sessions. A small subset of the peers experience a high percentage of piece loss, but most peers experience almost no loss. There are at least two reasons why a peer may loose many pieces. First and foremost, since peers barter for pieces using the BitTorrent protocol, our extensions inherit any inefficiencies present in BitTorrent. Second, some peers may not have the necessary bandwidth to watch the video stream in the first place, due to having a narrow up- or downlink or due to side traffic. Xie et al. [100] measure an average piece loss of 1%, which is less than our average of 6.6%. On the other hand, Agarwal et al. [7] measure a median piece loss of 5%, which is more than our median of 0.4%.

Figure 4.8(d) shows the average percentage of time the peers were stalled. The peak on Tuesday 22nd is caused by our DV camera being disconnected. Note that the peak of 50% on Friday 18th is actually measured over only two peers, one of them stalling. Overall, the stall time typically fluctuates between 0% and 10%. We found most of this stall time to be present just after a peer started playback, which is an artifact of the SwarmPlayer not being able to predict how much data the video playback

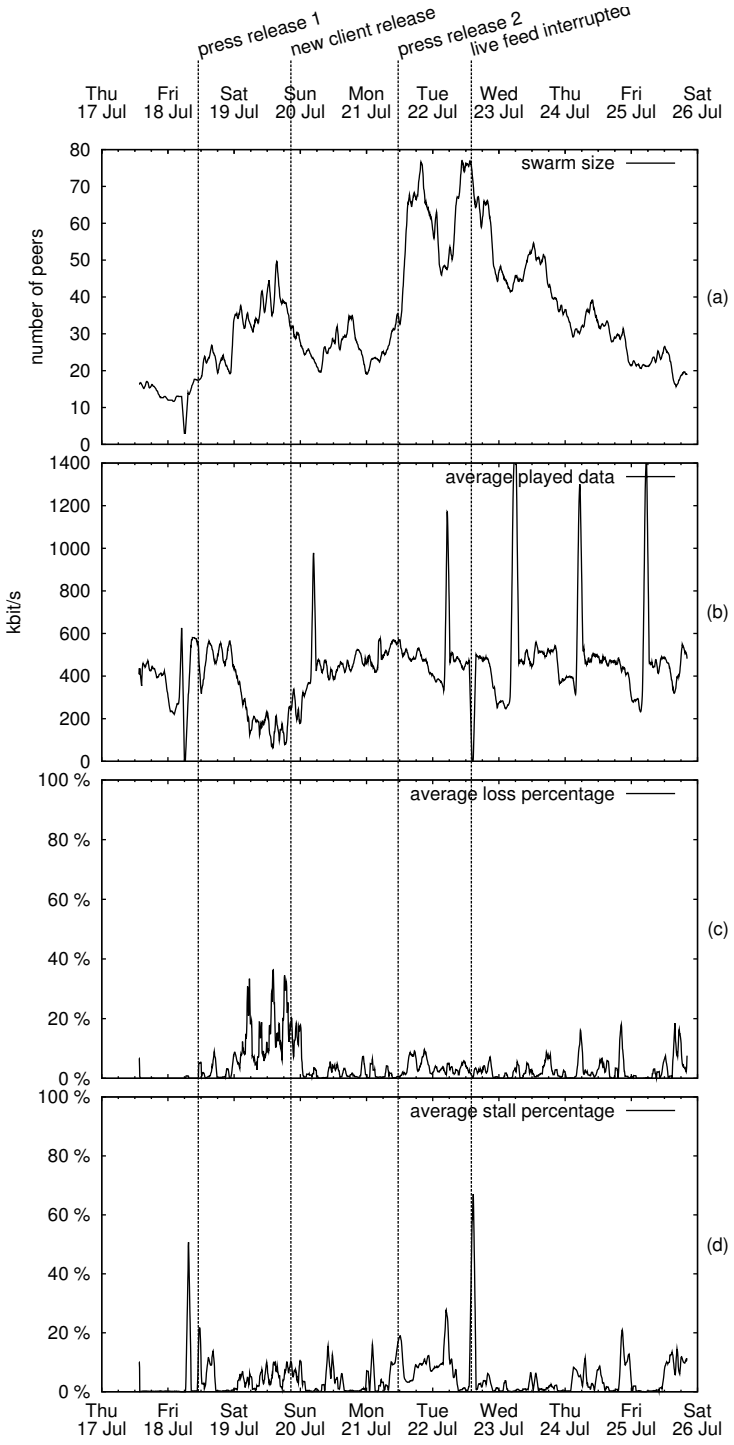


Figure 4.8: The performance during the trial.

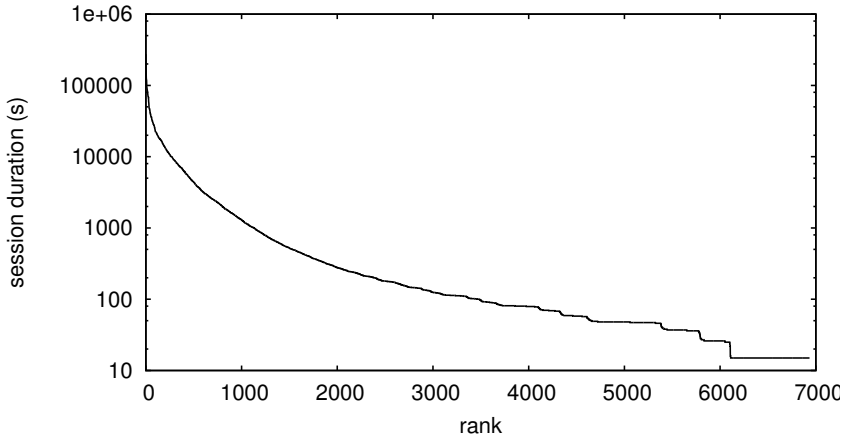


Figure 4.9: The duration of the user sessions, ranked according to length.

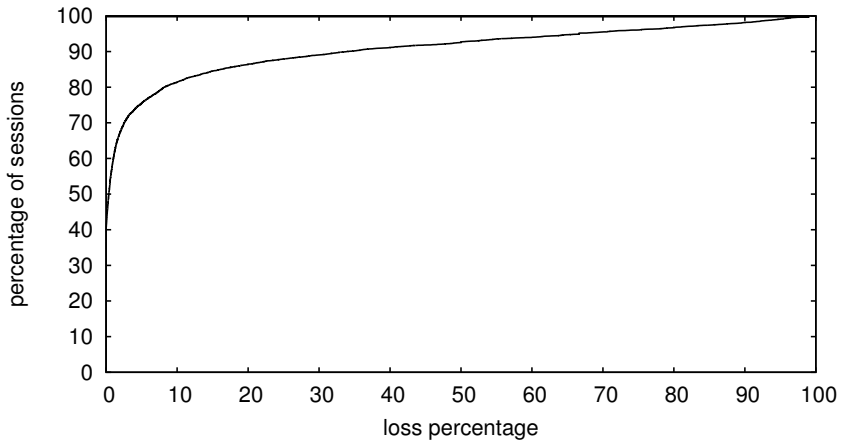


Figure 4.10: The cumulative distribution of the percentage of pieces lost per user session.

module (VLC) will discard before playback is started. If VLC discards too much data, the peer will have to stall in order to stay synchronised with its neighbours. Neither Xie et al. [100] nor Agarwal et al. [7] mention their peers stalling video playback. We assume their algorithms use a large enough buffer to accommodate for most scenarios, a fact that will be reflected in the differences in prebuffering times.

4.4.3 Prebuffering Time

The prebuffering time determines how long it takes for a peer to start playback. Even though a long prebuffering time allows the player to accumulate a large buffer and therefore minimise piece loss and stalling, we regard a short prebuffering time to be more desirable since it reduces the amount of time the user will have to wait before viewing the first frame. The distribution of prebuffering times required in our trials is shown in Figure 4.11. The median prebuffering time was 3.6 seconds, with 67% of the peers requiring less than 10 seconds. A few peers require substantially more prebuffering time. A possible explanation is that such peers do not provide enough upload bandwidth for BitTorrent to perform the tit-for-tat bartering. The relation between the prebuffering time and the average upload speed in each session is shown in Figure 4.12. A negative correlation is clearly visible, in which peers with low average upload speeds tend to require longer prebuffering times. Average upload speeds higher than the video bitrate are possible when the session is short, since all pieces can be downloaded simultaneously in the prebuffering phase.

Xie et al. [100] measure a median prebuffering time between 10 and 20 seconds, and Agarwal et al. [7] find a median prebuffering time of 27 seconds. Both of these figures are significantly larger than our 3.6 second median.

4.4.4 Sharing Ratios

The data that are not uploaded by our injector and our seeders is uploaded by the peers. We aim to distribute the burden of uploading fairly among the peers, to avoid having to rely on a subset of altruistic peers. As a measure of fairness, we use the *sharing ratio* of a peer (or a group of peers), which is defined as the number of uploaded bytes divided by the number of downloaded bytes. Peers with a sharing ratio smaller than 1 are net consumers, those with a sharing ratio larger than 1 are net producers.

In our trial, we found 61% of the IP addresses to be firewalled, and firewalled peers accounted for 52% of the on-line time of all peers. Figure 4.13 plots the cu-

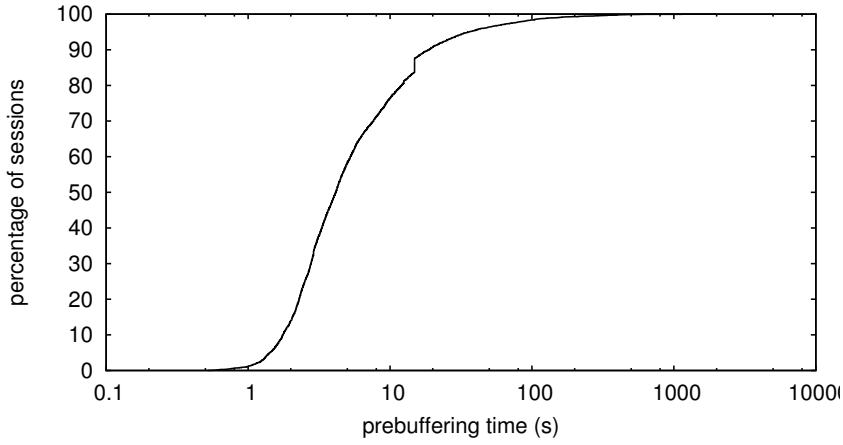


Figure 4.11: The cumulative distribution of the prebuffering time per user session.

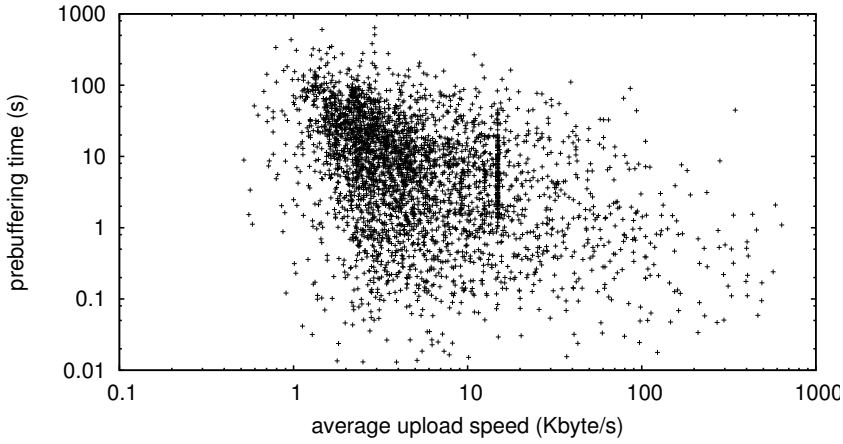


Figure 4.12: The prebuffering time against the average upload speed for all user sessions.

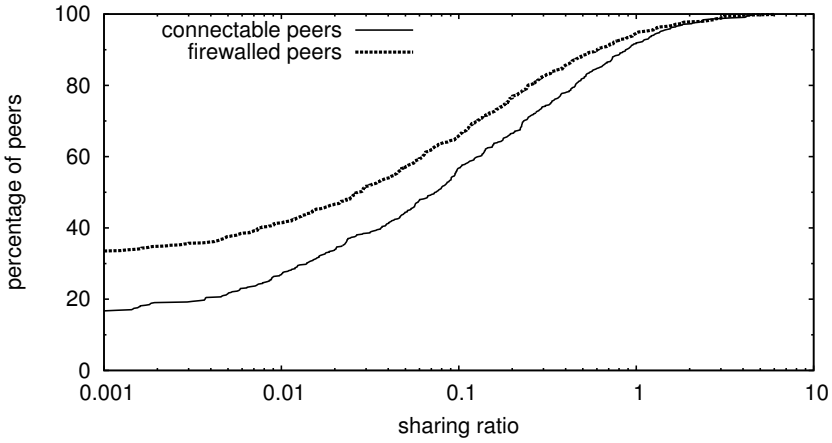


Figure 4.13: The cumulative distribution of the sharing ratios of the connectable and firewalled peers.

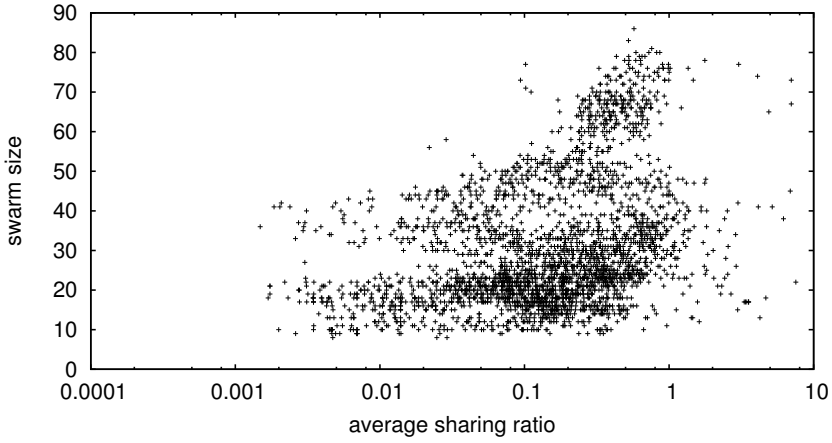


Figure 4.14: The swarm size against the average sharing ratio for every five-minute interval.

mulative distribution of the sharing ratios for both the firewalled and the connectable (non-firewalled) peers. The difference in sharing ratios is clearly visible. The connectable peers are able to upload much more data than the firewalled peers. In total, the connectable peers upload 0.41 as much as they download, while the firewalled peers upload only 0.18 times as much.

Our injector and seeders provided the rest of the data, which amounts to 74% of the pieces. Even though we employed five seeders, they were hardly used. The injector supplies 72% of the pieces, and the seeders only 1.6%. One reason for this bias is the fact that the injector obtains and announces each piece slightly before any seeders do. Any peer connected to both the injector and a seeder will thus see each piece appear at the injector first, and will request it immediately if possible. Apparently, the injector provided enough bandwidth so that the peers did not have to fall back on and request pieces from the seeders. The seeders were thus not needed to serve our limited number of users, but also did not have to provide more than a token amount of resources.

Figure 4.14 shows the sharing ratios of all peers for each five-minute interval. For small swarms, most peers receive their data from the injector and the seeders as they are the first to obtain each piece, and they have enough upload bandwidth to serve all peers. When the swarm grows, the peers start to forward the stream to each other. The average sharing ratio clearly improves for larger swarms, which is an indication for the scalibility of our approach. Note that sharing ratios higher than 1 are due to artefacts of measurement, in which data is uploaded in a different five-minute interval than it is downloaded, or one of the peers disconnects before reporting the number of downloaded bytes.

In Xie et al. [100], 70% of the peers were firewalled, and the other 30% of the peers provided 80% of the bandwidth exchanged by the peers, but they do not mention how much data are actually coming from their central servers. Their figures do however, like ours, indicate a significant skew in the resource contribution towards the connectable peers. The peers measured by Agarwal et al. [7] have roughly 10% of the data coming from their central servers. They report 84% of their peers to be firewalled; since these firewalled peers have to obtain the data from the 16% connectable peers, the latter will have to provide an amount of upload bandwidth equal to several times the video bitrate, which correlates with the sharing ratio distribution given in [7].

The performance of all measured systems, including ours, depends on the upload

bandwidth the injector and the connectable peers are able and willing to provide. Since a high percentage of firewalled peers in a deployed system seems unavoidable, the injector and the connectable peers may not be able to provide all of them with the video stream. Firewall traversal techniques will have to be deployed in order to increase the contribution of the firewalled peers.

4.5 Conclusions

In this chapter, we have presented extensions to the BitTorrent protocol which add live streaming functionality. Among other things, we added a rotating sliding window over a fixed set of pieces in order to project an infinite video stream onto a fixed number of pieces. The BitTorrent protocol provides our extensions with a resilience to free-riding through its use of tit-for-tat data exchange. When our extensions as well as Video-on-Demand extensions [70, 98] are added to a BitTorrent client, a single solution is created for streaming live and pregenerated video, as well as for off-line viewing of high-definition content. The overlap between these components reduces the time to implement them, and allows them all to benefit from future improvements to BitTorrent. The mechanisms we have presented to map a live data stream onto a fixed-sized file can be used in other peer-to-peer systems as well, if they need to be able to handle data streams of both a finite and an infinite size in a transparent manner.

We implemented our extensions into our BitTorrent client called Tribler, and launched a public trial inviting users to tune into our DV camera feed, in which we used values for several parameters that were found to be optimal in the simulations presented in the first part of this chapter. A total of 4555 users from around the globe joined our swarm. Even though we experienced minor startup problems, and at most only 86 concurrent users were present, we were able to provide most peers with a good quality of service. Especially the prebuffering time required by the peers was significantly lower (3.6 seconds median) than was measured in other deployed peer-to-peer systems. The percentage of chunk loss that we measured is comparable to the percentages that have been measured in other systems.

With our trial we have shown that an existing BitTorrent client can be extended to support live video streaming, which allows our streaming algorithm to be implemented on top of a wide variety of existing mature BitTorrent clients. Our trial suggests that our extensions are indeed capable of live video streaming with a performance equal to or exceeding that of other deployed peer-to-peer live video streaming

systems.

In our trial, we found 61% of the peers to be behind a firewall or NAT. According to our simulations, such a percentage has a negative significant negative impact on the performance. Deployed peer-to-peer systems measured by others seem to have a high percentage of peers behind a firewall or NAT as well. In fact, in the next chapter, we will consider the impact of firewalls and NATs in a more general setting, by showing that a lack of connectivity causes performance problems regardless of the distribution algorithm used.

Chapter 5

Bounds on Bandwidth Contributions

In the previous chapters, we assumed a simple model for the peers in a P2P network when we analysed and simulated the performance of the P2P algorithms. A model necessarily does not take all aspects of peers into consideration, in order to keep it tractable. Among other things, we assumed full connectivity: any two peers that are on-line at the same time can connect to each other. However, full connectivity is generally not the case on the Internet. Studies [7, 17, 23, 79, 99, 100, 101] have found 25%–93% of the peers to be behind a firewall or a Network Address Translator (NAT), which often block incoming connection requests. Since two peers who do not accept incoming connection requests can never connect to each other, the connectivity on the Internet is thus partial, and in some cases, extremely limited.

In this chapter, we study the effect of such limited connectivity on the load balancing within P2P networks in general. Most peers in a P2P file-sharing network will not contribute by uploading files if no countermeasures are taken [6]. Peers that download files without contributing anything in return are called *free-riders*, and one of the ways to avoid it is to give peers an incentive to upload as much data as they download. A common metric of fairness in P2P networks is the *sharing ratio* of a peer, which is defined as the total number of uploaded bytes divided by the total number of downloaded bytes.

We will prove that it is impossible for all peers to achieve a fair sharing ratio, that is, a sharing ratio equal to 1, if more than half of the peers are firewalled. As

the percentage of firewalled peers increases, the average sharing ratios of the firewalled peers rapidly drop, regardless of the amount of seeding they do or the amount of content they inject. This implies that free-riding cannot be avoided in such situations, which has a fundamental impact on algorithms that expect peers to be able to contribute as much as they consume.

One way to counter the connectivity problems caused by firewwalls and NATs is to employ techniques like *firewall puncturing* or *NAT traversal* [15, 39, 46], which enable two firewalled peers to establish a connection. No perfect puncturing technique is known, because firewall and NAT behaviour is not standardised.We will give a lower bound on the required effectiveness of such techniques for systems which require all of its peers to obtain a fair sharing ratio.

We validate our model of connectivity in P2P systems using simulations as well as real-world measurements. We simulate individual BitTorrent sessions, and analyse the actual average sharing ratios of the firewalled and the connectable peers. For real-world measurements, we have collected and analysed data on the behaviour of several BitTorrent communities, both open and closed. In an open community, anyone can join any swarm (the set of peers downloading the same file), without sharing ratio enforcement. In a closed community, only members can download files, and they are banned if their sharing ratio drops below a certain treshhold. On average, the BitTorrent protocol rewards good uploaders with a better download performance. Firewalled peers are limited in the amount of data they can contribute to the system as our theorems will show. We therefore expect the connectivity problems of the firewalled peers to be reflected in their download performance. Furthermore, we will show that in a closed community, in which fair sharing ratios are enforced for all peers, a majority of firewalled peers cannot be allowed. We will present data on the behaviour of real systems consistent with these observations.

This chapter is organised as follows. In Section 5.1, we will explain the basics behind firewalls and NATs. Then, in Section 5.2 we will introduce the network model we consider. In Section 5.3, we will derive bounds on the sharing ratio of both firewalled and connectable peers if no firewall puncturing or NAT traversal techniques are employed. In Section 5.4, we derive a lower bound on the effectiveness of the puncturing techniques to make a fair sharing ratio possible. In Section 5.5, we evaluate the sharing ratios of peers in BitTorrent sessions through simulation. In Section 5.6, we will present our data on the behaviour of real BitTorrent communities. Finally, in Section 5.7 we discuss related work and in Section 5.8, we draw our conclusions.

5.1 Firewalls and Puncturing

Although P2P algorithms are often designed assuming that every peer can accept incoming connections from other peers, in practice this is not always the case. In this section, we discuss the two main causes of this, which are firewalls and NATs, along with firewall puncturing techniques, which allow some firewalls and NATs to nevertheless accept incoming connections anyway.

5.1.1 Firewalls and NATs

A firewall is a piece of software or hardware which can be configured to block certain incoming connections. Firewalls are used for security purposes — services which are unintentionally exposed to the outside world can be a target for hackers. By using a firewall, the system administrator can decide which protocols, ports, or programs are allowed to accept inbound connections. As a result, some peers in a P2P network which operate behind such a firewall are unable to accept incoming connections.

A Network Address Translation gateway (NAT) is a router which translates network addresses between two address spaces. Such a setup is common if a consumer or corporation is given a single (public) IP address but has several computers he wishes to connect to the Internet. In that case, one computer (the NAT) is assigned the public IP address. All computers, including the gateway, are assigned a private IP address. The Internet traffic is routed through the NAT. For every outbound connection, the NAT keeps track of its origin within the private address space and routes all packets in either direction accordingly. For every inbound connection, the NAT cannot know to which computer the corresponding traffic must be routed.

A NAT is a popular default setup for broadband users, who are often unaware of this connectability problem or do not have the technical knowledge to configure their NAT. Because broadband users form a significant fraction of the users on the Internet, we conjecture that they form the bulk of the firewalls and NATs as well. Some NATs can be configured to route certain traffic to certain computers automatically by the use of UPnP [65], which is a 1999 standard to let desktop computers configure the NAT. However, the adoption of UPnP has been very slow. A 2007 measurement by Xie et al. [100] found only 19% of the peers to have UPnP enabled, even though 89% of the peers were firewalled or behind a NAT.

There are three reasons to assume that not all pairs of hosts will be connected.

First, UPnP may never be deployed on all NATs, and will be shipped disabled by default on others. Second, corporations will likely view UPnP to be a security hole, as it allows users to open a port within the private network to the rest of the Internet. Third, NATs have a side-effect of increasing security, because the rest of the Internet cannot access the computers behind the NAT directly. Once NATs can be configured to make such computers connectable, new security threats will arise and firewall usage will increase in response. A common firewall policy is to allow any outbound connections and block all incoming connections except to the services which are explicitly allowed. The firewalls are unlikely to be configured to allow any application to receive incoming connections from anywhere on the Internet, which will invariably lead to a certain fraction of peers with limited connectivity. The fraction of firewalled peers within a P2P system will thus continue to depend on the nature of the community.

We will, for ease of discussion, use the term 'firewalled peers' for both peers behind a firewall and those behind a NAT whose firewall or NAT is not configured to accept incoming connections. Such peers can initiate outbound connections, as well as upload and download to others. They cannot, however, connect to other firewalled peers, unless special techniques are used.

5.1.2 Firewall Puncturing

Techniques called *firewall puncturing* and *NAT traversal* can be used to establish a connection between two firewalled peers, which typically works as follows. Under the supervision of a (connectable) coordinating peer, both firewalled peers initiate a connection at the same time. When an outgoing connection is initiated, the NAT gateway knows where incoming packets should go if it judges them to be part of the same connection. When using UDP, which is a stateless protocol, a reasonable percentage of the firewalls can be punctured (for example, using STUN [87] or NUTSS [45]). Being able to puncture with stateful protocols like TCP is substantially harder [15, 39, 46], since both of the firewalls need to agree on the state to be established. This involves guessing, as either peer has to predict what state on the other end will be expected. Ford et al. measured an overall success rate of 82% for UDP puncturing and 64% for TCP puncturing [39], although their results varied wildly between different NAT hardware vendors. Note that the success rate will be lower if bidirectional communication is required between firewalled peers, as both peers need to support the implemented puncturing technique.

Although puncturing using UDP has a reasonable success rate, its use complicates P2P system design as it will have to implement its own stateful protocols on top of UDP. Some firewalls or NATs cannot be punctured because they do not allow puncturing for security reasons. Finally, NAT behaviour is not standardised, making the implemented techniques difficult to maintain and their future effectiveness uncertain.

5.2 Model and Notation

We consider a P2P network consisting of a set $\mathcal{N}$ of peers which will upload and/or download a file (or video stream) of L bytes. As only the amount of data exchanged will be relevant, it will not matter when these peers arrive or depart. The set $\mathcal{N}$ is split up into two disjoint sets, the set $\mathcal{F}$ of firewalled peers, which cannot accept incoming connections, and the set $\mathcal{C}$ of connectable peers, which can accept incoming connections. We assume a peer $p \in \mathcal{F}$ and $q \in \mathcal{C}$ can always connect by having the connection originate from p. Without puncturing techniques, no connections between peers in $\mathcal{F}$ are possible. Furthermore, we define $N \equiv |\mathcal{N}|$, $F \equiv |\mathcal{F}|$ and $C \equiv |\mathcal{C}|$, and we define $f \equiv F/N$ as the fraction of firewalled peers.

We will use two metrics for fairness. First, we define $S_{\mathcal{P}}$ to be the average sharing ratio of a peer in set $\mathcal{P}$. Second, we define the *debt* $\Delta(p)$ of a peer p is the number of bytes downloaded minus the number of bytes uploaded by p, and $\Delta(\mathcal{P}) \equiv \sum_{p\in\mathcal{P}} \Delta(p)$ is the debt of a group of peers $\mathcal{P}$. We will only consider the sharing ratios and the debts of the peers once all peers have completed the download.

The amounts of data contributed by the individual peers is typically skewed: some peers upload more than they download, and other peers download more than they upload. To obtain a fair resource contribution for all peers, *sharing ratio enforcement* can be introduced. The sharing ratio of a peer is the total number of bytes it has uploaded divided by the total number of bytes it has downloaded. A P2P system can be designed to expect users to aim for a sharing ratio of 1, representing a fair situation in which a peer on average has contributed as much as it has consumed. To obtain a fair sharing ratio, a peer can either inject new content or upload the file to others (including seeding, that is, continuing to upload the file after the download is completed).

5.3 No Firewall Puncturing

This section will provide bounds on the sharing ratio of both the firewalled and connectable peers if the P2P system does not employ firewall puncturing or NAT traversal techniques. The reasoning will be roughly as follows. Firewalled peers can only receive data from connectable peers. If there are more firewalled peers than connectable peers, the majority of firewalled peers can only upload to the minority of connectable peers, and a fair sharing ratio for the firewalled peers will be impossible to obtain. First, we will derive bounds on the sharing ratios of both firewalled and connectable peers. Then, we will discuss some practical implications of the derived results.

5.3.1 Sharing Ratio Analysis

It is useful to first derive the debt of each peer after all peers have finished downloading the file.

Lemma 1. *For the debt of the firewalled peers $\Delta(\mathcal{F})$, we have $\Delta(\mathcal{F}) \geq ((2f-1)N-1)L$.*

Proof. We consider two cases: the file has been injected by a firewalled peer, or by a connectable peer. In the first case, since the firewalled peers cannot form connections among each other, they have to obtain the file from the connectable peers. This creates a data flow from $\mathcal{C}$ to $\mathcal{F}$ of size $(F-1)L$ (the injector already has the file). The connectable peers can obtain the file from both sets of peers, creating a data flow from $\mathcal{F}$ to $\mathcal{C}$ of at most $C \cdot L$. Figure 5.1 illustrates these flows. The difference between these flows is thus bounded by

$$\begin{aligned} \Delta(\mathcal{F}) &\geq (F-1)L - C \cdot L \\ &= ((2f-1)N-1)L. \end{aligned}$$

In case the file was injected by a connectable peer, similar to the first case, we find

$$\begin{aligned} \Delta(\mathcal{F}) &\geq F \cdot L - (C-1)L \\ &= (f \cdot N - (1-f)N + 1)L \\ &> ((2f-1)N-1)L. \end{aligned}$$

□

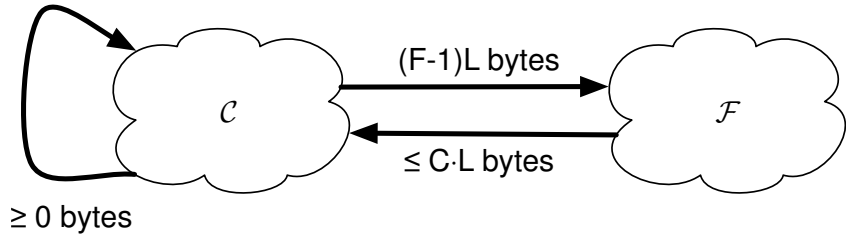


Figure 5.1: Data flows between connectable and firewalled peers, if the file was injected by a firewalled peer.

Note that the bound in Lemma 1 holds regardless of whether the injector is connectable or firewalled. The firewalled peers will end up with a positive debt ($\Delta(\mathcal{F}) > 0$) if $f > 0.5 + 0.5/N$. In that case, it is impossible for the average firewalled peer to obtain a fair sharing ratio, as the firewalled peers as a group have to download more than they can upload. The distribution of the upload burden over the firewalled and connectable peers determines the exact sharing ratio for each firewalled peer individually.

The bound on the amount of data the firewalled peers are able to upload results in bounds on the average sharing ratio for both firewalled and connectable peers. We shall not regard the injector's sharing ratio, for the following reasons. The injector only uploads data, giving it a sharing ratio of infinity. Also, some peers will not be able to upload any data (for example, consider a system with one injector and one downloader). These imbalances are expected to average out once the peers have downloaded multiple files. To keep the sharing ratio of the injector finite when considering a single file, we will assume that the injector downloads the file as well. This introduces a relative error in the order of $1/N$ in all the derived bounds, since the file will be downloaded by the injector superfluously. The following lemma gives bounds on the sharing ratios of the firewalled and the connectable peers.

Lemma 2. *For the average sharing ratios of firewalled and connectable peers, we have* $S_{\mathcal{F}} \leq (1/f) - 1$, *and* $1/(1-f) - 1 \leq S_{\mathcal{C}} \leq 1/(1-f)$, *respectively.*

Proof. Using Lemma 1, the average debt for the firewalled peers is $\Delta(\mathcal{F})/F \geq (2 - 1/f - 1/(f \cdot N))L$, which converges to $(2 - 1/f)L$ for large N. Since every peer downloads exactly L bytes, the average sharing ratio of the firewalled peers is thus $S_{\mathcal{F}} = (L - \Delta(F)/F)/L \leq 1/f - 1$.

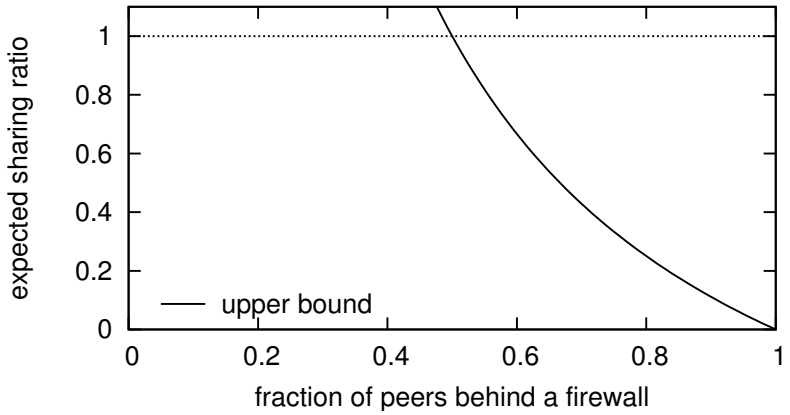


Figure 5.2: Bounds on the average sharing ratio of the firewalled peers against the fraction of firewalled peers. The horizontal dotted lines represent a fair sharing ratio of 1.

The connectable peers can be shown to have a sharing ratio if at least $1/(1-f)-1$ on average using the same method. The average sharing ratio of the connectable peers is also bounded from above. Together, the N peers download $N \cdot L$ bytes. If all these bytes are uploaded by connectable peers, then the average sharing ratio for the connectable peers is $S_C = N \cdot L/(C \cdot L) = 1/(1-f)$, otherwise it is smaller. □

The lower bound for the firewalled peers and the upper bound for the connectable peers are related. Every peer downloads the file once and has to upload it once, on average. As a result, the average sharing ratio of all peers is 1. However, for a subset of the peers this average does not necessarily hold.

Figures 5.2 and 5.3 plot the bounds of Lemma 2 against the fraction of firewalled peers f, for the firewalled peers and the connectable peers, respectively. These bounds are only met if all connectable peers obtain their data from the firewalled peers. In the case of $f < 0.5$, the bounds of Lemma 2 still hold, but the average sharing ratio of either group of peers will be closer to 1 if the data distribution algorithm does not force all data for connectable peers to originate from firewalled peers.

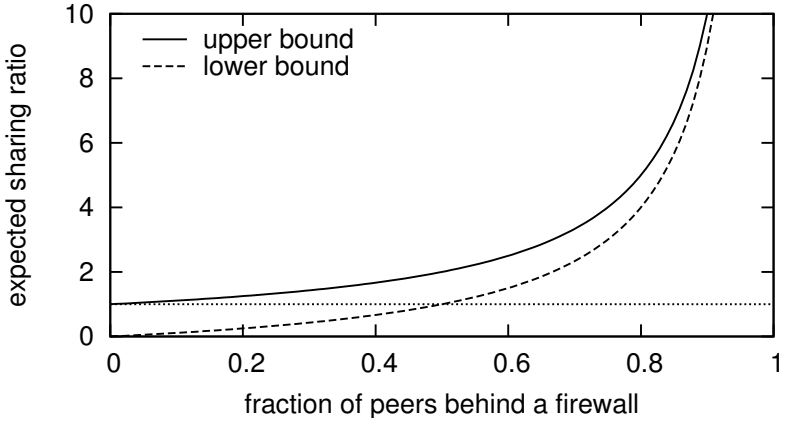


Figure 5.3: Bounds on the average sharing ratio of the connectable peers against the fraction of firewalled peers. The horizontal dotted lines represent a fair sharing ratio of 1.

5.3.2 Practical Implications

The bounds derived so far depend only on the connectivity of the peers in each swarm. The bounds even hold if a peer seeds (uploads the file to others after completing the download) in an attempt to restore its sharing ratio. The upload speeds of the peers are irrelevant as well, as only the amount of data is considered. Furthermore, the results hold for file downloading as well as video-on-demand, and can be trivially extended to live video streaming. For the latter, the duration a peer decides to watch the video stream has to be taken into account, but that duration can be assumed to be independent of the connectivity of the peer. Finally, our results hold regardless of the data distribution method used, and therefore cover swarm-based distribution methods (which we consider in this chapter) as well as tree-based distribution methods, as long as data is exchanged using unicast connections.

If $f > 0.5 + 0.5/N$, the connectable peers have to upload more than the firewalled peers. The injector could find this undesirable, and aim at an equal sharing ratio for all peers instead. Such a situation can be obtained if the injector increases its upload capacity to serve the firewalled peers in order to lower the sharing ratios of the connectable peers to the level of the firewalled peers. Although the average sharing ratio for both sets of peers will be below 1, fairness is nevertheless achieved as both sets have an equal uploading burden. Using Lemma 2, a lower bound on the upload

capacity for the injector can be derived as follows.

Theorem 1. *It is impossible for all peers to obtain an equal sharing ratio if the injector uploads less than* $(2-1/f)N \cdot L$ *bytes to the firewalled peers.*

Proof. To create an equal sharing ratio for all peers, the connectable peers have to upload at least $S_{\mathcal{C}} \cdot L - S_{\mathcal{F}} \cdot L$ bytes less on average. Using Lemma 2, for all connectable peers combined, this amounts to at least

$$\begin{aligned} C \cdot (S_{\mathcal{C}} \cdot L - S_{\mathcal{F}} \cdot L) &\geq C \cdot \left(\frac{1}{1-f} - 1 - \left(\frac{1}{f} - 1 \right) \right) L \\ &= \left(2 - \frac{1}{f} \right) N \cdot L \end{aligned}$$

bytes. These bytes have to be uploaded by the injector instead. □

If the fraction of firewalled peers f can be measured or estimated, the injector can thus predict a lower bound on the capacity needed to provide all downloaders with an equal sharing ratio. The actual required capacity is likely to be higher. If not all peers are on-line at the same time, some peers will be unable to upload data to each other. The injector will have to compensate for this if all peers are promised an equal sharing ratio.

5.4 Firewall Puncturing

In Section 5.1, we have discussed the general principles behind firewall puncturing. Whether a connection between two firewalled peers can be made depends on the types of firewall of both peers. There can be many different types of firewall for which a P2P distribution algorithm has implemented puncturing techniques. We will first analyse a system in which firewalls can be either punctured by everyone or not at all. We will then propose a more generic model.

Consider a system in which a firewall can be either punctured by all other peers, or by no one. Let $\mathcal{F}_a \subseteq \mathcal{F}$ be the set of peers with puncturable firewalls, and let $\mathcal{F}_b \equiv \mathcal{F} \setminus \mathcal{F}_a$ be the set of peers with non-puncturable firewalls. Also, define $F_a \equiv |\mathcal{F}_a|$ and $F_b \equiv |\mathcal{F}_b|$. Figure 5.4 shows the corresponding graph, where α to ε denote the total numbers of bytes transferred between the different groups of peers.

Theorem 2. *A fair sharing ratio for all peers is only possible if* $F_a/F \geq 2-1/f$.

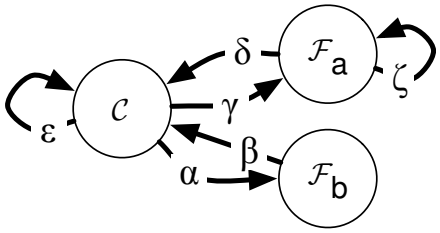


Figure 5.4: Possible data flows between the connectable peers $\mathcal{C}$ and firewalled peers $\mathcal{F}_a$ and $\mathcal{F}_b$ (puncturable and non-puncturable, respectively).

Proof. We will assume $L = 1$ without loss of generality. To obtain a fair sharing ratio, peers in $\mathcal{F}_a$ and $\mathcal{F}_b$ have to upload $F_a + F_b = F$ bytes, so

$$\beta + \delta + \zeta = F \tag{5.1}$$

has to hold. Peers in $\mathcal{C}$ download exactly C bytes, so

$$\begin{aligned} \beta + \delta + \varepsilon &= C \\ \Rightarrow \beta + \delta &\leq C. \end{aligned} \tag{5.2}$$

For the same reason,

$$\begin{aligned} \zeta + \gamma &= F_a \\ \Rightarrow \zeta &\leq F_a. \end{aligned} \tag{5.3}$$

Combining equations 5.1, 5.2, and 5.3 yields

$$\begin{aligned} F &= \beta + \delta + \zeta \\ &\leq C + F_a \\ \Rightarrow f \cdot N &\leq (1-f)N + F_a \\ \Rightarrow F_a/N &\geq 2f - 1 \\ \Rightarrow F_a/F &\geq 2 - \frac{1}{f}. \end{aligned}$$

□

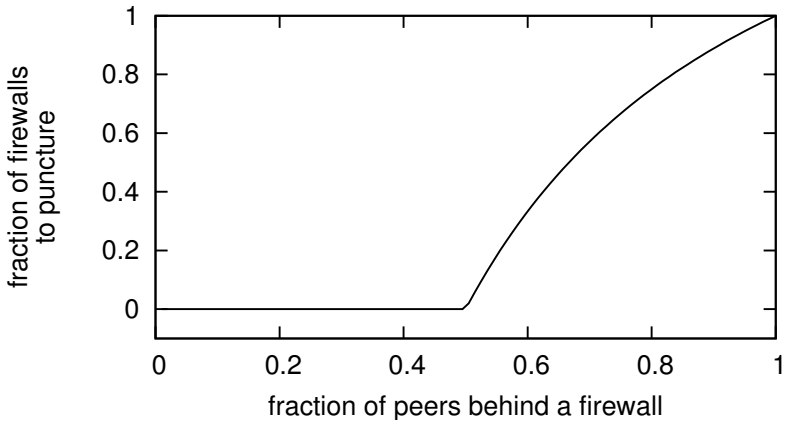


Figure 5.5: Lower bound on the fraction of firewalls that need to be punctured for a fair sharing ratio to be possible.

Figure 5.5 plots the lower bound of $2-1/f$ of the firewalls that need to be punctured against the fraction of firewalled peers f. For $f < 0.5$, a fair sharing ratio is (theoretically) possible without any firewall puncturing. The rapid rise in required firewall puncturing effectiveness is clearly visible once the fraction of firewalled peers is above 0.5.

The bounds in Lemmas 1–2 and Theorems 1–2 can be derived in a more general way by modelling a P2P system as follows. Let there be P types of firewall, of which one type represents having no firewall at all, and let P_i be the number of peers of firewall type $i, i = 1,\ldots,P$. Let $x_{ij} \geq 0$ be the number of bytes sent from peers of firewall type i to peers of firewall type j. Every peer has to download exactly L bytes, so

$$\sum_{j=1}^{P} x_{ij} = P_i \cdot L, \qquad i = 1,\ldots,P.$$

If a fair sharing ratio is desired, then all peers have to upload as many bytes as they download, resulting in

$$\sum_{j=1}^{P} x_{ji} = \sum_{j=1}^{P} x_{ij}, \qquad i = 1,\ldots,P.$$

By solving these equations for the x_{ij}, bounds can be derived for the average sharing

ratio of the peers in each group. As argued in Section 5.3.1, we have left out the injector, introducing small errors in the x_{ij}. To derive exact bounds, the injector can be added as a separate set of one peer. For $P = 2$, the model is consistent with the results derived in Section 5.3.

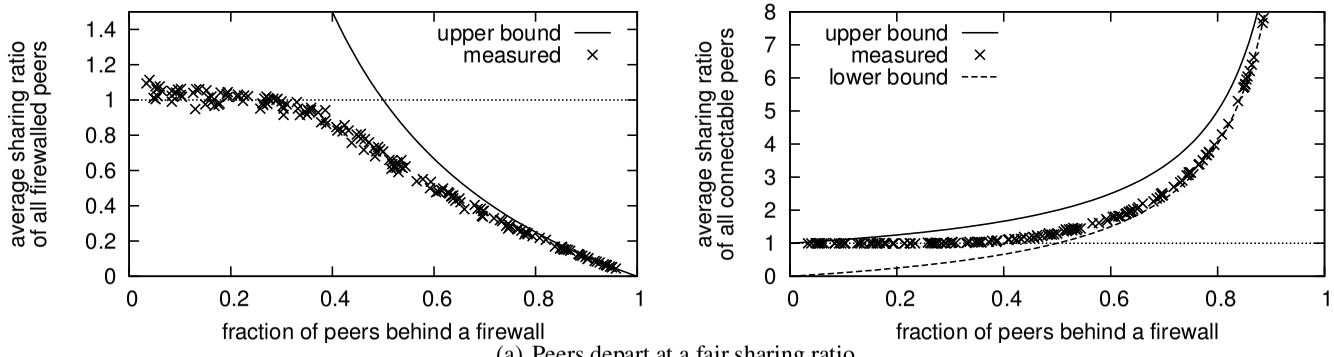
Whether a fair resource allocation can be obtained in a real setting depends on the P2P distribution algorithm as well as the arrival and departure pattern of the peers. After all, peers which are not on-line at the same time cannot connect to each other, regardless of firewalls or firewall puncturing techniques. The set of equations derived in this chapter can be used to derive bounds on the supported fraction of firewalled peers in each group, or to check feasibility of an implementation.

5.5 Simulated Behaviour

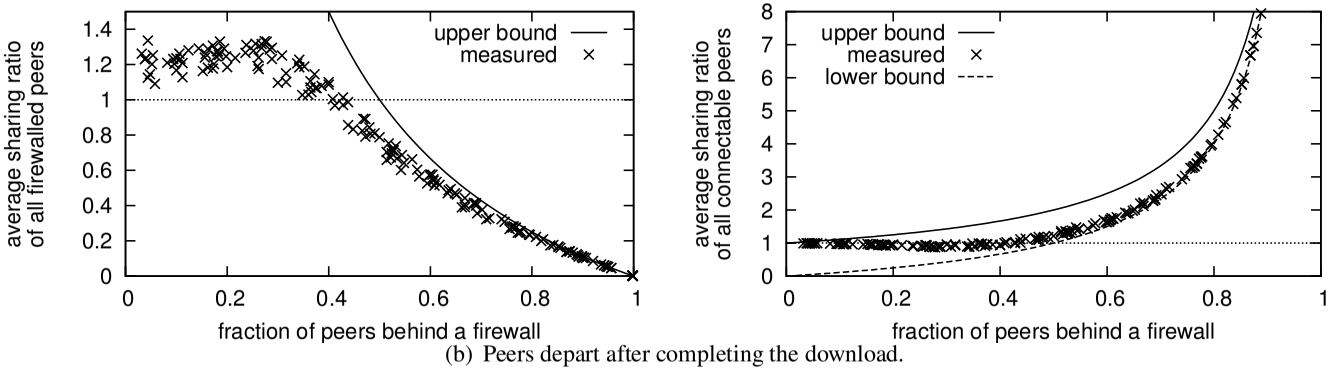
We use a discrete-event BitTorrent simulator to evaluate how close the actual average sharing ratios for the firewalled and connectable peers are to the bounds derived in Section 5.3. We let 500 peers arrive (1 per second, on average) to download a 10 MByte file. A randomly chosen subset of the peers are firewalled. The injector is always on-line and is not firewalled. Each peer can upload with 0.5 Mbit/s to 0.75 Mbit/s, and has a download speed four times as high. The latency between each pair of peers is 100 to 300 ms.

We evaluate two departing policies. In the first policy, we let each peer depart if it completed the download and has obtained a fair sharing ratio as well. In the second policy, we run the same simulations, but let each peer depart directly when it has completed the download, regardless of its sharing ratio. We performed 180 simulations with each policy.

For each session, we record the number of firewalled and connectable peers, and sum the total amounts of data sent and received by both groups. The average sharing ratio of either group is then derived and shown as a dot in the subfigures of Figure 5.6. The theoretical bounds as derived in Section 5.3 are shown as well. Each measurement falls within the derived bounds. The figures for the two departure policies are similar. At low fractions of firewalled peers, the average sharing ratios of both groups tend towards 1, which is consistent with BitTorrent rewarding good uploaders with a good download performance. At high fractions of firewalled peers, the firewalled peers upload an amount close to their theoretical maximum. Almost all of them obtain the



(a) Peers depart at a fair sharing ratio.



(b) Peers depart after completing the download.

Figure 5.6: The average sharing ratios for the firewalled peers and the connectable peers each session for two departure policies. The horizontal dotted lines represent a fair sharing ratio of 1.

file from the injector, since other connectable peers are rare and quickly meet the departure requirements.

For fractions of firewalled peers smaller than 0.5, the average sharing ratios for the firewalled peers in Figures 5.6(a) and 5.6(b) differ slightly. If peers depart right after they complete their download, the connectable peers will have less opportunity to upload to each other, thus allowing the firewalled peers to upload to them more. This policy is interesting in systems which keep track of the sharing ratios of each peer across several sessions. In such systems, the firewalled peers could for example join sessions with a low fraction of firewalled peers, in order to increase a low sharing ratio. Both figures for the connectable peers show that high sharing ratios for them are realistic when the fraction of firewalled peers is high.

5.6 Behaviour of Real Systems

In order to assess the validity of the model we have presented in Sections 5.2, 5.3, and 5.4 in real systems, we have collected data on the behaviour of peers, and in particular, of firewalled peers and seeders, in several BitTorrent communities. Below, we first explain the difference between open and closed BitTorrent communities, then we discuss our two ways of collecting data on the behaviour of BitTorrent communities, and finally we present the results on the behaviour of these communities with respect to firewalled peers and seeders.

5.6.1 BitTorrent Communities

Every BitTorrent community uses one or more centralised servers (*trackers*) that keep track of the peers in each swarm; peers report to the (a) tracker when they join a swarm, and they report again when they leave. BitTorrent communities can be open or closed. Open communities use public trackers which anyone can join, and they have no explicit mechanisms for sharing-ratio enforcement. Closed communities use private trackers to prevent outsiders from joining their swarms. In closed communities, every user has an account on a centralised server and sharing ratios are enforced by banning peers whose sharing ratios drop below a (secret) treshhold. Peers are expected to seed content in order to maintain or restore their sharing ratios. Unfortunately, we know of no BitTorrent community that publishes the sharing ratios of the individual peers, making it impossible to verify the bounds derived in this chapter di-

rectly. We can therefore offer only correlations between our theory and the measured results.

5.6.2 Data Collection

We have employed two methods for collecting data on the behaviour of BitTorrent communities, one in which we monitor a community for a period of time, and one in which we take snapshots of communities. As to the first method, we use a data set that we have collected in May 2005 [54], which consists of monitoring information of the operation of 1,995 BitTorrent swarms during one week in the (open) The Pirate Bay community.

We extend the analysis presented in [54] by extracting the behaviour of the firewalled peers from this data set. During the data collection, every two minutes, all peers that were reported to exist in each of the existing swarms were contacted. We consider a peer to be firewalled if it is repeatedly reported to exist but could never be contacted.

As to the second method of collecting data, we have taken snapshots of several BitTorrent communities in January 2008, which are listed in Table 5.1. These communities publish on their web sites the number of downloaders and seeders in each swarm, and we have collected these statistics at a single instant in time for one type of content (TV shows), because it was present in all communities. Nevertheless, we do consider the measurements to be representative as they contain swarms of varying ages. Of the communities in Table 5.1, TVTorrents also publishes which peers are actually present in which swarm, and whether they are firewalled or not. We collected these data for 557 swarms.

5.6.3 Behaviour of Firewalled Peers

In this section we present the behaviour of the The Pirate Bay (using data collected with our first method) and of TVTorrents (using the second method) with respect to firewalled peers. The TVTorrents data is used in Figure 5.7 (top) only. As can be seen in Figure 5.7 (top), which plots the cumulative distribution function (CDF) of the percentage of (unique) firewalled peers within each swarm, a substantial fraction of the peers in each swarm is firewalled. For the The Pirate Bay, the average swarm has 66% of its peers firewalled, and more than 96% of the swarms have more than half of the peers behind a firewall. In contrast, the average swarm in the closed TVTorrents

Nr	Community	Type	Swarms	Source
1	TheBox	closed	2,430	thebox.bz
2	TVTorrents	closed	2,363	tvtorrents.com
3	BTJunkie	closed	3,267	btjunkie.org
4	The Pirate Bay	open	7,368	thepiratebay.org
5	BTJunkie	open	16,400	btjunkie.org

Table 5.1: The communities for which we have used the second method of data collection. BTJunkie actually collects statistics on both open and closed communities.

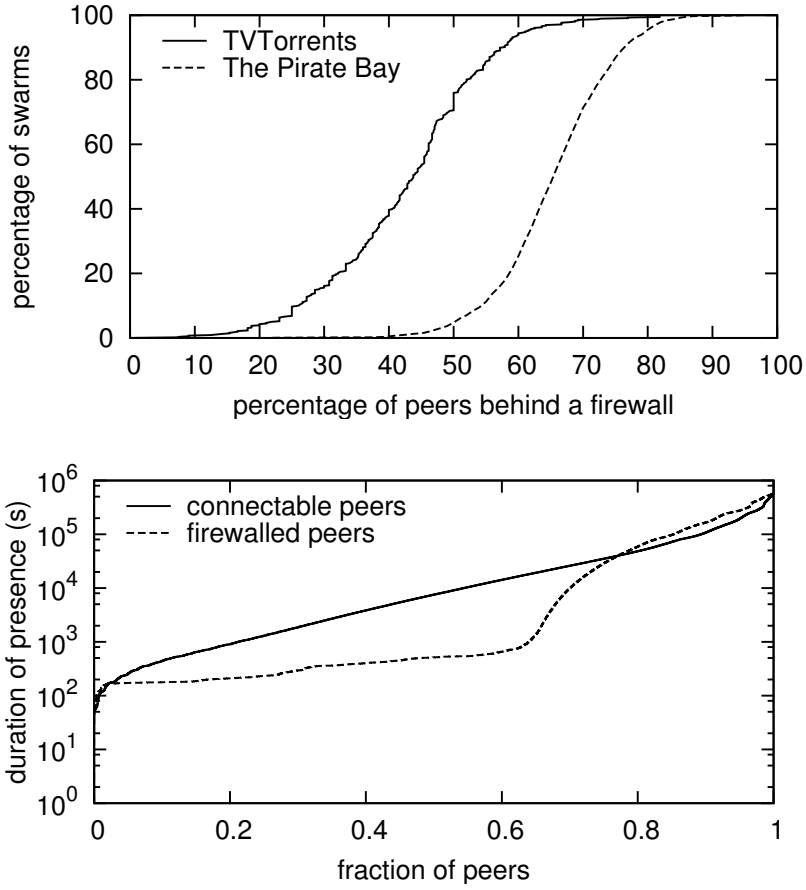


Figure 5.7: The CDF of the percentage of firewalled peers per swarm (top) and of the durations of the presence of the peers (bottom).

community only has 45% of its peers behind a firewall, and 24% of the swarms have more than half of the peers behind a firewall. The reason for the difference between these communities is beyond our abilities to measure, but the difference does correlate with our theory. Even though the measurements of both communities were taken 20 months apart from each other, measurements by others do not suggest a decrease in the fraction of firewalled peers over time [7, 17, 23, 79, 99, 100, 101]. We believe this difference to be due to the policy of banning peers with low sharing ratios in closed communities such as TVTorrents. First, our analysis in Section 5.3 shows that firewalled peers have more difficulty obtaining a fair sharing ratio, and so they have a higher probability of getting banned. Secondly, to reduce the risk of getting banned, a firewalled peer has an incentive to configure its firewall to accept incoming connections. A closed community thus favours users which have the technical knowledge to do so.

The BitTorrent protocol has been designed to approximate a fair sharing ratio by giving the best performance to peers that upload fastest. A low upload speed results, on average, in a low download speed. As a consequence, the firewalled peers will have a harder time finding connectable peers to upload to, often resulting in poor performance. Also, the firewalled peers do not necessarily have the same upload capacity as the connectable peers.

Figure 5.7 (bottom) plots how long the firewalled and connectable peers stayed in the system, measured as the difference between the first and the last time they were contacted. The firewalled peers are biased towards both a short and a long presence. A short presence can be explained by firewalled peers not being able to contact enough connectable peers at start-up, resulting in poor performance, after which the download is aborted by the user. When performance is acceptable, the firewalled peers stay in the system longer than the connectable peers. A possible explanation is that the firewalled peers have to complete their download with a lower download speed, again due to connectivity problems. On average, the presence of a firewalled peer was 12.8 hours, while the presence of a connectable peer was only 10.6 hours. For the peers that stayed for at least an hour, these averages rise to 38.5 and 17.1 hours, respectively. We found that at any moment, 72% of the on-line peers were firewalled on average, due to their longer presence in the system.

The distribution of the fraction of firewalled peers f as shown in Figure 5.7 (top) does not suffice to assess the actual impact of firewalled peers, since it does not provide insight into the absolute number of peers and the size of the file that is shared

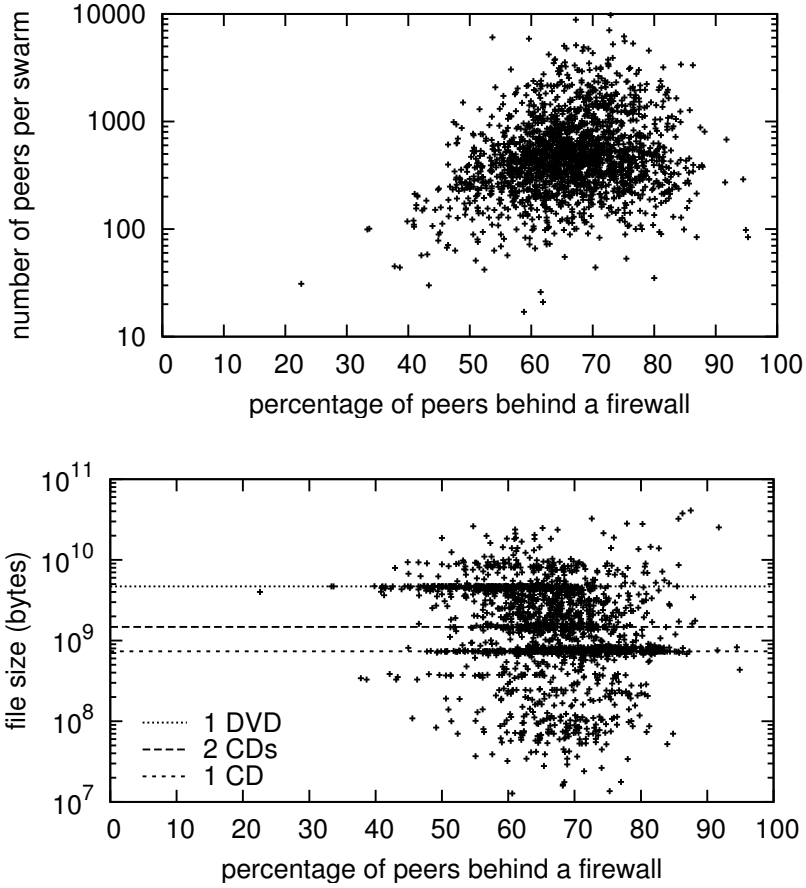


Figure 5.8: A scatter plot of the number of peers per swarm (top) and of the size of the file being downloaded (bottom) versus the percentage of firewalled peers.

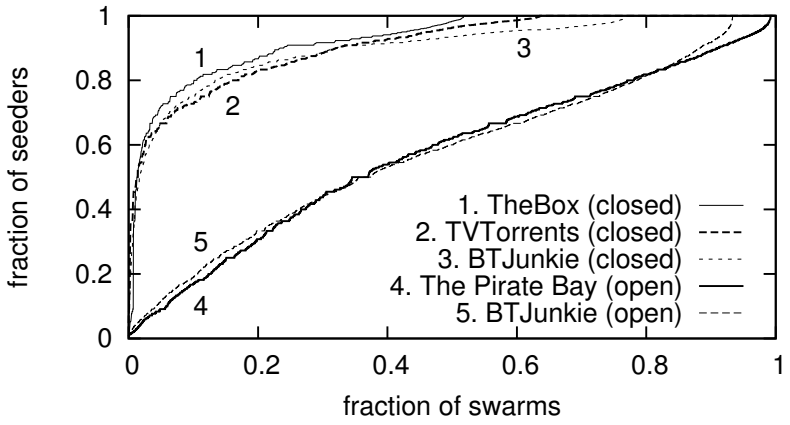


Figure 5.9: The fraction of peers which are seeding for open and closed communities.

in the swarms with $f > 0.5$. Therefore, we have evaluated the correlations between the fraction of firewalled peers and the swarm size, as well as the size of the shared file. Figure 5.8 (top) is a scatter plot of the number of unique peers in a swarm versus the percentage of firewalled peers in the swarm. There is no strong correlation (the correlation coefficient is 0.22), indicating that popular files (swarms with many peers) do not seem to have a significant bias in the percentage of firewalled peers. Figure 5.8 (bottom) shows the size of the file exchanged within a swarm against the percentage of firewalled peers in the swarm (the correlation coefficient is -0.26). There is a bias in file size as many swarms exchange files with a size corresponding to 1 or 2 CDs (703 Mbyte per CD) or to 1 DVD (4.37 GByte). This bias is not surprising as many of the files were actually movies, transcoded to fit on either media. The swarms exchanging DVDs contain a slightly smaller percentage of firewalled peers compared to the swarms exchanging CDs. We conjecture that one of the reasons for this phenomenon is that many users have ADSL connections. On the one hand, such connections are (relatively) slow, leading to a preference for CD versions of movies, and on the other hand, such connections typically employ a NAT-router.

5.6.4 Fraction of Seeders

In this section we present the behaviour of the BitTorrent communities listed in Table 5.1 (using data collected with our second method) with respect to seeders. Figure 5.9 shows the CDF of the fraction of seeders in the swarms at the time of snapshot.

The numbers for BTJunkie are split for their statistics on open and closed communities.

Open communities do not keep track of their users across swarms, so there is no reason for a peer to stay on-line to seed in a swarm after the download has finished. Some users will nevertheless seed, out of altruism, or because their client is configured to do so. A common default configuration for BitTorrent clients is to seed until the user aborts the program, or until a fair sharing ratio is reached for that swarm.

In the closed communities, swarms have higher fractions of seeders, and there is a higher fraction of swarms which contained only seeders at the moment of measurement. Once a peer joins such a swarm, it will experience a high download speed due to the many seeders present, but it will also have to compete with these seeders to restore its sharing ratio. The peers are forced by the sharing ratio enforcement to nevertheless try to restore their ratio by seeding as much content as possible. The firewalled peers will have to seed longer on average, as they can only connect to connectable arriving peers. The latter is a possible explanation for the many swarms in which seeds are idle, waiting for a peer to arrive and to start downloading from them.

Section 5.6.3 has shown that the average swarm has almost half its peers behind a firewall, implying that the distribution of the file data needs to be near-perfect if a fair sharing ratio is to be possible for the average firewalled peer. In theory, closed communities can optimise the communication between peers using sharing ratio information, but in practice, they do not. We expect the data distribution to be closer to our simulations than to a near-perfect distribution. A fraction of the firewalled peers will not be able to obtain a fair sharing ratio, and thus effectively will have to seed.

5.7 Related Work

To the best of our knowledge, we are the first to derive theoretical bounds on the sharing ratio that can be achieved in P2P networks. Algorithms designed for P2P data distribution generally do not take firewalls into account, even those that are designed with fairness in mind [51, 58, 59, 68, 70, 97]. Ripeanu et al. [85] confirm some of our findings and recognise that the limited connectivity of firewalled peers makes it harder for them to contribute. Also, the high fraction of seeders in closed communities has been recognised by them and others [11, 85]. However, neither study looked at the fundamental limits imposed by firewalls and the resulting necessity in closed communities of having many seeders in many swarms.

The percentages of firewalled peers we measured are consistent with previous measurements [7, 17, 23, 79, 99, 100, 101], but these vary widely in their results (between 25% and 93%). Firewall puncturing and NAT traversal techniques have been researched [15, 39, 45, 46, 87], and proven to be quite effective for the considered sets of firewalls and NATs. However, in practice, these techniques are non-trivial to implement and maintain, and often need a third (connectable) party to coordinate the puncturing between two firewalled peers. Also, for best results, one is forced to use UDP [87] instead of TCP, as the latter requires more complex puncturing techniques which depend on the types of firewall present in the network [15, 39, 46].

5.8 Conclusions

We have shown that in P2P file-sharing networks, there is a trade-off between security and fairness: peers behind firewalls do not accept incoming connections, and as a consequence, if there are too many of them, there is no way freeriding can be prevented. We have provided bounds on the sharing ratios that can be obtained by firewalled and connectable peers, which hold regardless of how long firewalled peer remain in the system to seed to other peers, and regardless of their connection speeds. Firewall puncturing and NAT traversal techniques may alleviate the freeriding problem, and we provided a lower bound on the fraction of firewalls that have to be punctured to be able to obtain a fair sharing ratio. Because firewall and NAT techniques evolve, a standard for both firewall puncturing and NAT traversal is required to guarantee fairness in P2P networks.

We have both run simulations and real-world measurements to validate our theory. In our simulations, we have shown the actual behaviour of the average sharing ratios of firewalled and connectable peers. When almost all peers are connectable, the BitTorrent protocol is sufficient to keep the system fair, as both groups have an average sharing ratio close to 1. The average sharing ratio of the firewalled peers decreases and converges to the theoretical upper bound as the fraction of firewalled peers increases. We have also collected and analyzed data on the behaviour of both open and closed P2P communities which reveal the real-world behaviour of firewalled peers. It turns out that the fraction of firewalled peers is significant: In closed communities, 45% of the peers are firewalled, versus 66% in open communities. A peer can increase its sharing ratio by seeding, and indeed we found a significantly higher fraction of seeders in the closed communities, indicating that a fair sharing ratio is not

easy to obtain. The observed behaviour is consistent with our analysis, and can aid in the design of new P2P data-distribution algorithms.

Chapter 6

Conclusion

WITH THE BANDWIDTHS currently available on the Internet, peer-to-peer (P2P) video streaming is a promising alternative to the classical client-server approach. The amount of bandwidth needed to serve a video stream is linear in the number of users, and in a client-server model, all of this bandwidth has to be provided by the servers. The bandwidth costs for the source thus scale linearly as well. A P2P video streaming solution reduces the load on the servers by inviting the users to provide their uplink bandwidth to help spread the content. The bandwidth costs for the servers can potentially be reduced to a constant.

However, the users in a P2P network have no inherent incentive to provide their upload bandwidth to help spread the video content. The quality of service for a user depends on what he downloads, not on what he forwards to others. Peers that forward little or no data to others are called free-riders. The subject of this thesis was to investigate and counter the effects of free-riding in P2P video streaming networks. We presented algorithms that do provide such an incentive by increasing the download performance of those users that provide a good upload performance to others. Furthermore, we analysed the bounds within which our algorithms, as well as P2P distribution algorithms in general, perform. In this chapter, we will present the conclusions of our research, alongside a brief summary of our work. Then, we will state our recommendations for future work.

6.1 Summary and Conclusions

We have described several algorithms which prevent or discourage free-riding for both live video streaming and for video-on-demand. These algorithms were analysed, simulated, emulated, and, in one case, deployed in the Internet. Furthermore, we provided theoretical bounds on the performance of these algorithms as well as on P2P streaming algorithms in general. We have come to the following conclusions:

1. It is possible to force peers to contribute as much as they consume in a live video streaming system. We have proven this by presenting our Orchard algorithm in Chapter 2. The Orchard algorithm is a video distribution scheme that requires (almost) every peer to upload as much data as it downloads. Orchard requires a video technique called Multiple Description Coding (MDC) to split the video stream into several substreams. A peer can decode any combination of these substreams to display the video stream in a quality proportional to the number of substreams it receives.

2. Free-riding can be tolerated but at the same time discouraged in a video-on-demand system. In Chapter 3, we presented the Give-to-Get algorithm, which provides video-on-demand as an extension to the popular BitTorrent protocol. In Give-to-Get, the peers that forward the most data are given preference by others to upload data to. The peers that upload the least amount of data will receive the worst quality of service. If there is contention for the available bandwidth, free-riding peers will thus receive little or no data, but if there is bandwidth in abundance, the video stream can be served to all peers.

3. Free-riding can be tolerated and discouraged in a live-video-streaming system as well. We presented another set of extensions to BitTorrent in Chapter 4, which provide live video streaming support. The extensions leave the BitTorrent incentives to upload video data unchanged. The resilience to free-riding therefore remains similar to the resilience inherent in BitTorrent.

4. P2P systems will suffer in performance if the connectivity between the peers is limited. In Chapter 5, we have proven hard bounds on the performance of any P2P distribution system, which depend solely on the connectivity between the peers. Measurements of deployed P2P systems indicate that the connectivity between the peers is often significantly reduced by the ubiquitous presence of

firewalls and NATs. When more than half of the peers is behind firewalls or NATs, as is often the case in deployed systems, the burden of forwarding the video stream is shifted towards the peers that are not behind firewalls or NATs. We have thus discovered that the presence of firewalls and NATs in the Internet has a significant impact on the performance of any P2P distribution system, including those that stream video.

6.2 Future Work

The design of a P2P video streaming algorithm is deceptively complex. Even though the basic requirements are simple (every peer needs to obtain the video stream in real time), the environment in which the algorithm has to operate is not. The set of peers in a deployed P2P system in the Internet turns out to be highly heterogeneous: the available bandwidth at a peer, its latency towards others, its connectivity, as well as the durations of its session, vary wildly between peers, and possibly even over time for the same peer. Models often omit many of these parameters to keep the design tractable. Deploying a P2P video algorithm to evaluate its performance is not easy due to the high number of test users required. As a result, the performance difference between simulations and deployed systems can be quite large while it nevertheless remains unnoticed by the authors. Therefore, we propose the following continuations based upon the findings in this thesis:

1. Flexible fairness measures need to be derived for video streaming in order to deal with asymmetric Internet connections. The fairness measures that we proposed in this thesis are based only on the most recent data exchanges between the peers. Peers with an asymmetric connection are not able to fully utilise their download capacity if they are expected to upload at a similar speed to others. Flexible fairness measures could allow the peers with an asymmetric connection to participate more efficiently. Even though such fairness measures exist, which amortise the bandwidth contribution and consumption over time, it is unknown how well they perform in a video streaming setting.

2. This thesis focuses on the prevention of free-riding in P2P video distribution systems, but other forms of attack on a P2P network are possible as well. For example, Give-to-Get is vulnerable to collusion attacks in which peers make

false claims about each other's contributions. Our work can be extended by incorporating protections against collusion attacks, but also against other attacks such as Sybil attacks [34]. Such protections can be added either by enhancing our algorithms or by solving the problems at a higher layer, for example, by introducing an identity system that can expose the misbehaviour of peers both within single and across multiple video streaming sessions.

3. The development of video codecs can aid P2P video streaming significantly. We used Multiple Description Coding (MDC) as a central technique in our Orchard algorithm. MDC provides scalability in the width of the video stream as well as enhanced resilience against data loss. However, the lack of implementations that support MDC makes it difficult to deploy P2P video streaming systems that require it. We find it likely that video codecs can be improved in several other ways as well, if the aspects of P2P video streaming systems are taken into consideration.

4. A unified model and commonly accepted test sets are needed in order to properly evaluate the performance of P2P video streaming systems, as well as to be able to compare them. Even though statistics of many behavioural and technical aspects have been collected, they are not combined into a single model describing the characteristics of peers in a P2P network. Such a model is needed, as P2P video streaming algorithms are very sensitive to many characteristics of the peers. The bandwidth available at the peers, the presence of firewalls, and the arrival and departure patterns of peers are examples of such characteristics which can have a significant impact on the performance of P2P video streaming.

5. A layered approach and framework is needed for P2P development. In this thesis, we base some of our algorithms on the BitTorrent protocol, which allows existing BitTorrent implementations to be adjusted to support video streaming. However, this approach is rare as most algorithms proposed in literature are designed and implemented from scratch. A complete implementation of a P2P video streaming algorithm requires a solution for several orthogonal problems such as peer discovery and meta data exchange. A common framework for P2P development would allow such problems to be solved separately, and provides a common base for the development of P2P video streaming solutions.

Bibliography

[1] DAS: Distributed ASCI Supercomputer. http://www.cs.vu.nl/das2.

[2] Freeband. http://freeband.nl.

[3] I-Share. http://ishare.ewi.tudelft.nl.

[4] VideoLan Client (VLC). http://videolan.org/.

[5] YouTube. http://youtube.com.

[6] E. Adar and B. Huberman. Freeriding on Gnutella. *First Monday*, 5(10), 2000.

[7] S. Agarwal, J. P. Singh, A. Mavlankar, P. Baccichet, and B. Girod. Performance and Quality-of-Service Analysis of a Live P2P Video Multicast Session on the Internet. In *Proc. of the 16th IEEE Intl. Workshop on Quality of Service (IwQoS)*, 2008.

[8] A. Alagoz, O. Ozkasap, and M. Caglar. SeCond: A System for Epidemic Peer-to-Peer Content Distribution. In *Proc. of the 7th Intl. Symp. on Computer Networks (ISCN)*, pages 248–253, 2006.

[9] Shahzad Ali, Anket Mathur, and Hui Zhang. Measurement of Commercial Peer-To-Peer Live Video Streaming. In *Workshop in Recent Advances in Peer-to-Peer Streaming*, 2006.

[10] Kevin C. Almeroth and Mostafa H. Ammar. On the Use of Multicast Delivery to Provide a Scalable and Interactive Video-on-Demand Service. *IEEE Journal on Selected Areas in Communications*, 14(6):1110–1122, 1996.

[11] N. Andrade, M. Mowbray, A. Lima, G. Wagner, and M. Ripeanu. Influences on Cooperation in BitTorrent Communities. In *Proc. of ACM SIGCOMM*, pages 111–115, 2005.

[12] Siddhartha Annapureddy, Saikat Guha, and Christos Gkantsidis. Is High-Quality VoD Feasible using P2P Swarming? In *Proc. of the 16th Intl. World Wide Web Conference*, pages 903–911, 2007.

[13] S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan. Resilient Multicast using Overlays. In *Proc. of ACM SIGMETRICS*, pages 102–113, 2003.

[14] Tim Berners-Lee, Robert Cailliau, Jean-François Groff, and Bernd Pollermann. World-Wide Web: The Information Universe. *Internet Research*, 2(1):52–58, 1992.

[15] A. Biggadike, D. Ferullo, G. Wilson, and A. Perrig. NATBLASTER: Establishing TCP Connections Between Hosts Behind NATs. In *Proc. of ACM SIGCOMM Asia Workshop*, 2005.

[16] S. Birrer and F.E. Bustamante. Nemo - Resilient Peer-to-Peer Multicast without the Cost. In *Proc. of SPIE, Multimedia Computing and Networking Conference (MMCN)*, volume 5680, pages 113–120, 2005.

[17] H. Burch and D. Song. A Security Study of the Internet: An Analysis of Firewall Behavior and Anonymous DNS. Technical Report CMU-CS-04-141, Carnegie Mellon University, 2004.

[18] R.W. Burns. *Television: An International History of the Formative Years*. Institution of Electrical Engineers, 1998.

[19] Miguel Castro, Peter Druschel, Ayalvadi J. Ganesh, Antony I. T. Rowstron, and Dan S. Wallach. Secure Routing for Structured Peer-to-Peer Overlay Networks. In *Proc. of the 5th USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, 2002.

[20] Miguel Castro, Peter Druschel, A-M. Kermarrec, A. Nandi, Antony I. T. Rowstron, and A. Singh. SplitStream: High-bandwidth multicast in a cooperative environment. In *Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, pages 298–313, 2003.

[21] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony I. T. Rowstron. Scribe: a large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8):1489–1499, 2002.

[22] Y. Chu, J. Chuang, and H. Zhang. A Case for Taxation in Peer-to-Peer Streaming Broadcast. In *Proc. of the ACM SIGCOMM workshop on Practice and Theory of Incentives in Networked Systems*, pages 205–212, 2004.

[23] Y. Chu, A. Ganjam, and T.S.E. Ng. Early Experience with an Internet Broadcast System Based on Overlay Multicast. In *Proc. of USENIX*, 2004.

[24] Y. Chu, S.G. Rao, and H. Zhang. A Case for End System Multicast. In *Proc. of ACM SIGMETRICS*, pages 1–12, 2000.

[25] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM Computer Communication Review*, 33(3):3–12, 2003.

[26] Bram Cohen. BitTorrent. `http://www.bittorrent.com/`.

[27] Bram Cohen. Incentives Build Robustness in BitTorrent. In *Proc. of the 1^{st} Workshop on Economics of Peer-to-Peer Systems*, 2003.

[28] C. Dana, D. Li, D. Harrison, and C.-N. Chuah. BASS: BitTorrent Assisted Streaming System for Video-on-Demand. In *Proc. of the 7^{th} Intl. Workshop on Multimedia Signal Processing*, pages 1–4, 2005.

[29] S. E. Deering and D.R. Cheriton. Host Groups: A Multicast Extension to the Internet Protocol. RFC 966, 1985.

[30] H. Deshpande, M. Bawa, and H. Garcia-Molina. Streaming Live Media over a Peer-to-Peer Network. Technical Report 2002-21, Stanford University, 2002.

[31] P. Dhungel, X. Hei, K. Ross, and N. Saxena. The Pollution Attack in P2P Live Video Streaming: Measurement Results and Defenses. In *Proc. of the Workshop on Peer-to-Peer Streaming and IP-TV*, pages 323–328, 2007.

[32] C. Diot, B. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment Issues for the IP Multicast Service and Architecture. *IEEE Network*, 14(1):78–88, 2000.

[33] T. Do, K. Hua, and M. Tantaoui. P2VoD: Providing Fault Tolerant Video-on-Demand Streaming in Peer-to-Peer Environment. In *Proc. of the IEEE Intl. Conf. on Communications*, volume 3, pages 1467–1472, 2004.

[34] J.R. Douceur. The Sybil Attack. In *Proc. of the 1st Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[35] P.T. Eugster, R. Guerraoui, A-M. Kermarrec, and L. Massoulie. Epidemic Information Dissemination in Distributed Systems. In *Computer*, volume 37, pages 60–67, 2004.

[36] Clive Evans, David Lacey, and David Harvey. *Client/Server: A Handbook of Modern Computer System Design*. Prentice Hall, 1995.

[37] R.A. Ferreira, M.K. Ramanathan, A. Awan, A. Grama, and S. Jagannathan. Search with Probabilistic Guarantees in Unstructured Peer-to-Peer Networks. In *Proc. of the 5th Intl. Conf. on Peer-to-Peer Computing*, pages 165–172, 2005.

[38] F.H.P. Fitzek, B. Can, R. Prasad, and M. Katz. Overhead and Quality Measurements for Multiple Description Coding for Video Services. In *Wireless Personal Multimedia Communications*, pages 524–528, 2004.

[39] B. Ford, P. Srisuresh, and D. Kegel. Peer-to-peer communication across network address translators. In *Proc. of USENIX*, page 13, 2005.

[40] P. Ganesan and M. Seshadri. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *Proc. of the 12th Intl. World Wide Web Conference*, 2003.

[41] L. Gao and D. Towsley. Supplying Instantaneous Video-on-Demand Services using Controlled Multicast. In *Proc. of the IEEE Intl. Conf. on Multimedia Computing and Systems*, volume 2, pages 117–121, 1999.

[42] Paweł J. Garbacki, Dick. H.J. Epema, and Maarten van Steen. An amortized tit-for-tat protocol for exchanging bandwidth instead of content in p2p networks.

In *Proc. of the 1st Intl. Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 119–128, 2007.

[43] Mark Goodyear, Hugh W. Ryan, Scott R. Sargent, and Timothy M. Boudreau. *Netcentric and Client/server Computing: A Practical Guide*. CRC Press, 1998.

[44] V.K. Goyal. Multiple Description Coding: Compression Meets the Network. *IEEE Signal Processing Magazine*, 18(5):74–93, 2001.

[45] S. Guha, Y. Takeda, and P. Francis. NUTSS: A SIP-based Approach to UDP and TCP Network Connectivity. In *Proc. of ACM SIGCOMM workshop on Future directions in network architecture*, pages 43–48, 2004.

[46] Saikat Guha and Paul Francis. Characterization and Measurement of TCP Traversal Through NATs and Firewalls. In *Proc. of the Internet Measurement Conference (IMC)*, pages 199–211, 2005.

[47] Y. Guo, K. Suh, J. Kurose, and D. Towsley. P2Cast: P2P Patching Scheme for VoD Services. In *Proc. of the 12th World Wide Web Conference*, pages 301–309, 2003.

[48] I. Gupta, A-M. Kermarrec, and A.J. Ganesh. Efficient and Adaptive Epidemic-style Protocols for Reliable and Scalable Multicast. In *IEEE Transactions on Parallel and Distributed Systems*, volume 17, pages 593–605, 2006.

[49] Fred Gurley. Unalienable Rights versus Union Shop. *Proc. of the Academy of Political Science*, 26(1):58–70, 1954.

[50] A. Habib and J. Chuang. Incentive Mechanism for Peer-to-Peer Media Streaming. In *Proc. of the 12th Intl. Workshop on Quality of Service (IWQOS)*, pages 171–180, 2004.

[51] M. Haridasan, I. Jansch-Porto, and R. van Renesse. Enforcing Fairness in a Live-Streaming System. In *Proc. of SPIE, Multimedia Computing and Networking Conference (MMCN)*, volume 6818, Article 68180E, 2008.

[52] 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 *Workshop on Internet Protocol TV (IPTV) Services over World Wide Web*, 2006.

[53] K.A. Hua, Y. Cai, and S. Sheu. Patching: a multicast technique for true video-on-demand services. In *Proc. of the 6th ACM Intl. Conf. on Multimedia*, pages 191–200, 1998.

[54] Alexandru Iosup, Paweł J. Garbacki, Johan A. Pouwelse, and Dick H.J. Epema. Correlating Topology and Path Characteristics of Overlay Networks and the Internet. In *Proc. of the 6th Intl. Workshop on Global and Peer-to-Peer Computing*, 2006.

[55] M. Jelasity and A-M. Kermarrec. Ordered Slicing of Very Large-Scale Overlay Networks. In *Proc. of the 6th Intl. Conf. on Peer-to-Peer Computing*, pages 117–124, 2006.

[56] Hemant Kanakia, Partho P. Mishra, and Amy R. Reibman. An adaptive congestion control scheme for real time packet video transport. *IEEE/ACM Transactions on Networking*, 3(6):671–682, 1995.

[57] Ming-Chieh Lee, Wei-Ge Chen, C.B. Lin, Chuang Gu, T. Markoc, S.I. Zabinsky, and R. Szeliski. A Layered Video Object Coding System using Sprite and Affine Motionmodel. *IEEE Transactions on Circuits and Systems for Video Technology*, 7(1):130–145, 1997.

[58] H. Li, A. Clement, E. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin. BAR Gossip. In *Proc. of the 7th USENIX Operating Systems Design and Implementation (OSDI)*, pages 191–206, 2006.

[59] Qiao Lian, Yu Peng, Mao Yang, Zheng Zhang, Yafei Dai, and Xiaoming Li. Robust Incentives via Multi-level Tit-for-tat. In *The 5th Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, 2006.

[60] C.S. Liao, W.H. Sun, C.T. King, and H.C Hsiao. OBN: Peering Finding Suppliers in P2P On-demand Streaming Systems. In *Proc. of the 12th Intl. Conf. on Parallel and Distributed Systems*, volume 1, pages 8–15, 2006.

[61] Thomas Locher, Patrick Moor, Stefan Schmid, and Roger Wattenhofer. Free Riding in BitTorrent is Cheap. In *Proc. of the 5th Workshop on Hot Topics in Networks*, 2006.

[62] Dmitri Loguinov and Hayder Radha. Measurement Study of Low-bitrate Internet Video Streaming. In *Proc. of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 281–293, 2001.

[63] Yue Lu, J. Jan David Mol, Ferdinand Kuipers, and Piet van Mieghem. Analytical Model for Mesh-based P2PVoD. In *Proc. of the 10th IEEE Intl. Symposium on Multimedia (ISM)*, pages 364–371, 2008.

[64] L. Mathy, N. Blundell, V. Roca, and A. El-Sayed. Impact of Simple Cheating in Application-Level Multicast. In *Proc. of the 23rd IEEE Intl. Conference on Computer Communications (INFOCOM)*, volume 2, pages 1318–1328, 2004.

[65] Microsoft Corporation. Universal Plug and Play Internet Gateway Device v1.01. 2001.

[66] J. Jan David Mol, Arno Bakker, Johan A. Pouwelse, Dick H.J. Epema, and Henk J. Sips. The Design and Deployment of a BitTorrent Live Video Streaming Solution. In *Proc. of the 11th IEEE Intl. Symposium on Multimedia (ISM)*, pages 342–349, 2009.

[67] J. Jan David Mol, Dick H.J. Epema, and Henk J. Sips. The Orchard Algorithm: P2P Multicasting without Free Riding. In *Proc. of the 6th Intl. Conf. on Peer-to-Peer Computing*, pages 275–282, 2006.

[68] J. Jan David Mol, Dick H.J. Epema, and Henk J. Sips. The Orchard Algorithm: Building Multicast Trees for P2P Video Multicasting Without Free-riding. *IEEE Transactions on Multimedia*, 9(8):1593–1604, 2007.

[69] J. Jan David Mol, Johan A. Pouwelse, Dick H.J. Epema, and Henk J. Sips. Free-riding, Fairness, and Firewalls in P2P File-Sharing. In *Proc. of the 8th IEEE Intl. Conf. on Peer-to-Peer Computing*, pages 301–310, 2008.

[70] J. Jan David Mol, Johan .A. Pouwelse, Michiel Meulpolder, Dick H.J. Epema, and Henk J. Sips. Give-to-Get: Free-riding-resilient Video-on-Demand in P2P Systems. In *Proc. of SPIE, Multimedia Computing and Networking Conference (MMCN)*, volume 6818, Article 681804, 2008.

[71] Joseph Needham. *Science and Civilisation in China*, volume 4. Cambridge University Press, 1965.

[72] Tsuen-Wan Johnny Ngan, Dan S. Wallach, and Peter Druschel. Incentives-Compatible Peer-to-Peer Multicast. In *The 2nd Workshop on the Economics of Peer-to-Peer Systems (P2PEcon)*, 2004.

[73] Martin A. Nowak and Karl Sigmund. Evolution of Indirect Reciprocity. *Nature*, 437:1291–1298, 2005.

[74] Organisation for Economic Co-operation and Development. *OECD Communications Outlook 2005*. OECD Publishing, 2005.

[75] V.N. Padmanabhan, H.J. Wang, and P.A. Chou. Resilient Peer-to-Peer Streaming. In *Proc. of the 11th IEEE Intl. Conf. on Network Protocols (ICNP)*, pages 16–27, 2003.

[76] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A.E. Mohr. Chainsaw: Eliminating Trees from Overlay Multicast. In *Proc. of the 4th Intl. Workshop on Peer-To-Peer Systems (IPTPS)*, pages 127–140, 2005.

[77] George Pallis and Athena Vakali. Insight and perspectives for content delivery networks. *Communications of the ACM*, 49(1):101–106, 2006.

[78] Fabio Pianese and Diego Perino. Resource and locality awareness in an incentive-based p2p live streaming system. In *Proc. of the 2007 Workshop on Peer-to-Peer Streaming and IP-TV (P2P-TV)*, pages 317–322, 2007.

[79] Johan A. Pouwelse, Paweł J. Garbacki, Dick H.J. Epema, and Henk J. Sips. The BitTorrent P2P File-Sharing System: Measurements and Analysis. In *Proc. of the 4th Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, pages 205–216, 2005.

[80] Johan A. Pouwelse, Paweł J. Garbacki, Jun Wang, Arno Bakker, Jie Yang, Alexandru Iosup, Dick H.J. Epema, Marcel Reinders, Maarten van Steen, and Henk J. Sips. Tribler: A Social-Based Peer-to-Peer System. In *Concurrency and Computation: Practice and Experience*, volume 20, pages 127–138, 2008.

[81] Johan A. Pouwelse, Jacco R. Taal, Reginald L. Lagendijk, Dick H.J. Epema, and Henk J. Sips. Real-Time Video Delivery using Peer-to-Peer Bartering Networks and Multiple Description Coding. In *Proc. of the IEEE Conf. on Systems, Man and Cybernetics*, pages 4599–4605, 2004.

[82] Louis Aimé Augustin Le Prince. Roundhay Garden Scene. film, 1888.

[83] Dongyu Qiu and R. Srikant. Modeling and Performance Analysis of BitTorrent-like Peer-to-Peer Networks. In *Proc. of ACM SIGCOMM*, pages 444–454, 2004.

[84] R. Rejaie, M. Handley, and D. Estrin. Architectural considerations for playback of quality adaptive video over the internet. In *Proc. of the 8th IEEE Intl. Conference on Networks (ICON)*, pages 204–209, 2000.

[85] Matei Ripeanu, Miranda Mowbray, Nazareno Andrade, and Aliandro Lima. Gifting technologies: A BitTorrent case study. *First Monday*, 11(11), 2006.

[86] Larry Roberts. The Arpanet and computer networks. *A history of personal workstations*, pages 141–172, 1988.

[87] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN-Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). RFC 3489, 2003.

[88] S. Saroiu, P.K. Gummadi, and S.D. Gribble. Measurement Study of P2P File Sharing Systems. In *Proc. of SPIE, Multimedia Computing and Networking Conference (MMCN)*, volume 4673, pages 156–170, 2002.

[89] S. Sen, J. Rexford, and D. Towsley. Proxy Prefix Caching for Multimedia Streams. In *Proc. of the 18th IEEE Intl. Conference on Computer Communications (INFOCOM)*, volume 3, pages 1310–1319, 1999.

[90] S. Sheu, K. Hua, and W. Tavanapong. Chaining: A Generalizing Batching Technique for Video-On-Demand Systems. In *Proc. of the Intl. Conf. on Multimedia Computing and Systems*, pages 110–117, 1997.

[91] S. Shin, J. Jung, and H. Balakrishnan. Malware Prevalence in the KaZaA File-Sharing Network. In *Proc. of the Internet Measurement Conference (IMC)*, pages 34–39, 2006.

[92] Atul Singh, Tsuen-Wan Ngan, Peter Druschel, and Dan S. Wallach. Eclipse Attacks on Overlay Networks: Threats and Defenses. In *Proc. of the 25th IEEE Intl. Conference on Computer Communications (INFOCOM)*, pages 1–12, 2006.

[93] D. Stutzbach and R. Rejaie. Characterizing Churn in Peer-to-Peer Networks. Technical Report CIS-TR-2005-03, University of Oregon, 2005.

[94] Jacco R. Taal and Reginald L. Lagendijk. Fair Rate Allocation of Scalable Multiple Description Video for Many Clients. In *Proc. of Visual Communications and Image Processing*, pages 2172–2183, 2005.

[95] D.A. Tran, K.A. Hua, and T. Do. ZIGZAG: An Efficient Peer-to-Peer Scheme for Media Streaming. In *Proc. of the 22nd IEEE Intl. Conference on Computer Communications (INFOCOM)*, pages 1283–1292, 2003.

[96] Athena Vakali and George Pallis. Content delivery networks: status and trends. *IEEE Internet Computing*, 7(6):68–74, 2003.

[97] V. Venkataraman, P. Francis, and J. Calandrino. Chunkyspread: Multi-tree Unstructured Peer-to-Peer Multicast. In *Proc. of the 5th Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, 2006.

[98] A. Vlavianos, M. Iliofotou, and M. Faloutsos. BiToS: Enhancing BitTorrent for Supporting Streaming Applications. In *IEEE Global Internet Symposium*, 2006.

[99] W. Wang, H. Chang, A. Zeitoun, and S. Jamin. Characterizing Guarded Hosts in Peer-to-Peer File Sharing Systems. In *Proc. of IEEE GLOBECOM*, volume 3, pages 1539–1543, 2004.

[100] Susu Xie, Gabriel Y. Keung, and Bo Li. A Measurement of a Large-Scale Peer-to-Peer Live Video Streaming System. In *Proc. of the IEEE Intl. Conf. on Parallel Processing Workshops*, page 57, 2007.

[101] Mao Yang, Zheng Zhang, Xiaoming Li, and Yafei Dai. An Empirical Study of Free-Riding Behavior in the Maze P2P File-Sharing System. In *Proc. of the 4th Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, pages 182–192, 2005.

[102] H. Yu, D. Zhang, B.Y. Zhao, and W. Zheng. Understanding User Behavior in Large Scale Video-on-Demand Systems. In *Proc. of the 1st ACM EuroSys*, pages 333–344, 2006.

[103] X. Zhang, J. Lieu, B. Li, and T.-S. P. Yum. DONet/Coolstreaming: A Data-driven Overlay Network for Live Media Streaming. In *Proc. of the 24th IEEE Intl. Conference on Computer Communications (INFOCOM)*, volume 3, pages 2102–2111, 2005.

[104] Ben Y. Zhao, Anthony D. Joseph, and John Kubiatowicz. Locality Aware Mechanisms for Large-scale Networks. In *Proc. of the Intl. Workshop on Future Directions in Distributed Computing (FuDiCo)*, pages 229–238, 2002.

Summary

TELEVISION AND FILM have become media that are able to reach billions of people. The Internet is increasingly being used to distribute video as well, as the potential audience is huge. However, in order to reach a large number of people over the Internet, a high amount of bandwidth is required to distribute the video, since every user needs to be sent a separate data stream. If the video streams are delivered from a central source, such as a website, the bandwidth costs are high or even prohibitive for large audiences. This problem exists for both live video streaming (like television) and for video-on-demand (like film), because both variants of video streaming require the same amount of bandwidth.

Peer-to-peer technology offers a solution to this problem. In peer-to-peer systems, the users (peers) forward the video stream among each other, such that as few of them as possible have to depend on the central source. Most of the peers will thus receive the video stream from other peers, and are expected to forward the video stream as well. Ideally, every peer forwards the same amount of data as it receives, because then the resource balance in the system is neutral with respect to both peers arriving and peers departing. Such a system can theoretically scale to millions of users without significantly increasing the burden on the central source.

However, by shifting the burden of distributing the video stream from the central source to the peers, the control over the quality of the delivered video streams is shifted to the peers as well. The peers may not be able to forward enough of the video stream for others to be able to receive the video stream at full quality. A typical set of peers on the Internet barely have enough uplink bandwidth to provide each other with a low-quality video stream. Furthermore, the quality of the video stream received by a peer does not necessarily depend on the amount of uplink bandwidth that peer provides to others. Since uplink bandwidth is a scarce resource, there exists an incentive for the peers not to share it. Peers that do not share their uplink bandwidth, or very little of

it, are called free-riders. In peer-to-peer systems that assume that all peers are benign, a large fraction of free-riders destroys the quality of service to other peers.

The objective of this thesis is to design and evaluate algorithms and systems in which free-riders are punished by receiving a low quality of service, or even by receiving no service at all. We present algorithms for live video streaming as well as video-on-demand, and evaluate their performance through analysis, simulation, emulation, and in one case by measuring a deployed system.

We start with a description of the background on which this thesis is based. Chapter 1 contains an introduction to video distribution over networks, as well as to peer-to-peer networks. From this introduction, research questions are derived, and linked to the chapters that follow.

In Chapter 2 we present the Orchard algorithm, which is a tree-based algorithm for live video streaming. Tree-based algorithms span a distribution tree over the peers over which the video stream is forwarded. The Orchard algorithm uses a video technique called *Multiple Description Coding* (MDC), that is used to split the video stream into several substreams. The peers forward these substreams, creating a different distribution tree for each substream. The peers forward and exchange substreams such that peers are forced to forward as much data as they receive. Free-riding is thus avoided. We provide an extensive performance analysis of Orchard, as well as emulation results, indicating that Orchard is capable of delivering decent performance.

Tree-based algorithms require that all peers need the same data at the same time, which is the case for live video streaming, but not for video-on-demand. For that reason, we turn to swarm-based algorithms in Chapters 3 and 4. In a swarm-based algorithm, the video stream is divided into pieces that are exchanged by the peers. Every peer strives to obtain all pieces, and holds on to them to serve others.

In Chapter 3, we present the Give-to-Get algorithm for the distribution of video-on-demand. Give-to-Get encourages peers to forward the video data to others by letting each peer serve the best forwarders first. Free-riders are thus served last, and will only receive video data if there is enough capacity in the network to serve them. An advantage of this approach is that peers that are willing but unable to upload much data (such as those with asymmetric links such as ADSL), can still be served if the amount of available bandwidth in the network allows it. The Give-to-Get algorithm is an extension of the popular BitTorrent file-sharing protocol, making it easy to implement it by extending an existing BitTorrent implementation. Again, we provide an extensive performance analysis as well as emulation results.

In Chapter 4, we present another extension to BitTorrent, but in this case for live video streaming. We map the live stream of potentially infinite length onto a finite set of pieces that are distributed using BitTorrent. Further modifications are needed for the peers to verify the origin of the pieces that the source continuously injects into the network, often replacing outdated pieces in the finite set. We provide performance analysis through simulation, and note that the fraction of peers behind a firewall or NAT is a significant inhibitor of the achieved performance. To properly asses the actual performance of our extension, we implemented it and deployed it in a trial, with participants from around the globe. We show that the performance of our extension is on par with previous trials of other live streaming algorithms. In our trial, like in the previous trials we compare our results with, a large percentage of peers was indeed behind a firewall or NAT.

The impact of firewalls and NATs on performance is the subject of Chapter 5, in which we derive bounds on the performance of peer-to-peer distribution algorithms imposed by a limited connectivity in the network. Such limited connectivity is most commonly caused by firewalls and NATs, that make the peers behind them hard to reach unless complex countermeasures are taken. Such peers can only maintain connections initiated by themselves. We prove that if more than half of the peers are behind a firewall or NAT, some peers will be forced to free-ride since there there are no peers they can connect to that still need data. We analyse data from our own trial and previous trials, that are consistent with previous studies, and show that in fact most peers often are behind a firewall or NAT. Our theoretical results on the bounds on the achievable performance in the presence of firewalls and NATs hold for any peer-to-peer distribution algorithm, making them useful tools for designing and deploying peer-to-peer algorithms.

In Chapter 6, we formulate our conclusions and summarize the answers to our research questions. Furthermore, we provide directions for future research building upon our work. The algorithms presented in this thesis can be implemented to serve as a basis for video streaming on the Internet, and be extended to provide good performance in a wider variety of cases than we were able to test. Furthermore, our derived performance impact of limited connectivity can be used to aid the design and deployment of video-streaming as well as file-sharing algorithms.

Samenvatting

TELEVISIE EN FILM zijn uitgegroeid tot media die dagelijks miljarden mensen bereiken. In toenemende mate wordt ook het Internet gebruikt om beeldmateriaal te verspreiden, om nog gemakkelijker een groot publiek te kunnen bereiken. Maar om videobeelden aan een groot publiek te kunnen leveren via het Internet is veel bandbreedte nodig, omdat de videostroom naar elke gebruiker apart verzonden moet worden. De videostroom vanaf een centraal punt aan alle gebruikers leveren is daarom erg kostbaar en vaak zelfs onmogelijk. Dit geldt zowel voor het verspreiden van *live* videobeelden (televisie) als voor het verspreiden van vooraf opgenomen videobeelden (film), omdat met beide varianten uiteindelijk evenveel bandbreedte gemoeid is.

Peer-to-peer technologie brengt een oplossing voor dit probleem. In peer-to-peer systemen geven de gebruikers (peers) de videobeelden aan elkaar door, zodat zo weinig mogelijk gebruikers afhankelijk zijn van een centraal punt. De meeste gebruikers ontvangen dus de videobeelden van andere gebruikers, en worden geacht om ze verder door te geven. In het ideale geval geeft elke gebruiker evenveel data door als hij ontvangt, omdat dan het systeem in balans blijft als er gebruikers arriveren of vertrekken. Een systeem waarin dat het geval is kan miljoenen gebruikers bedienen zonder dat het centrale punt substantiëel belast wordt.

Echter, als de gebruikers belast worden met het verspreiden van de videobeelden, wordt de kwaliteit van de videostroom die de gebruikers ontvangen afhankelijk van andere gebruikers. Die andere gebruikers hebben niet altijd genoeg bandbreedte om de videostroom foutloos te leveren. Een gemiddelde groep gebruikers op het Internet heeft nauwelijks genoeg bandbreedte om elkaar van lage kwaliteit videobeelden te voorzien. Daarnaast is de kwaliteit van de videostroom die een gebruiker ontvangt niet afhankelijk van de hoeveelheid bandbreedte die hij beschikbaar stelt. Uitgaande bandbreedte is schaars, en daarom zal een gebruiker geneigd zijn om zo min mogelijk bandbreedte te hoeven leveren. Gebruikers die hun uitgaande bandbreedte niet delen,

of zo min mogelijk, worden *profiteurs* (free-riders) genoemd. In peer-to-peer systemen die ervan uitgaan dat alle gebruikers goedaardig zijn is een grote groep profiteurs fataal voor de kwaliteit van de videostroom die de goedaardige gebruikers ontvangen.

Het doel van dit proefschrift is het ontwerpen en evalueren van algoritmes en systemen waarin profiteurs slecht beeld krijgen of zelfs helemaal niets kunnen ontvangen. We presenteren algoritmes voor het verspreiden van zowel live als vooraf opgenomen videobeelden, en evalueren hun prestaties met behulp van analyse, simulatie, emulatie, en in één geval door het gedrag van een werkend systeem te meten.

We beginnen met een beschrijving van de context waarin dit proefschrift zich bevindt. Hoofdstuk 1 bevat een inleiding in de distributie van videostromen over netwerken in het algemeen en peer-to-peer netwerken in het bijzonder. Vanuit de introductie worden de onderzoeksvragen geformuleerd, en verbonden met de hoofdstukken die volgen.

In Hoofdstuk 2 presenteren we het *Orchard* (Boomgaard) algoritme, wat een boom-algoritme is voor het verspreiden van live videobeelden. Een boom-algoritme spant een distributieboom over de gebruikers die de videostroom willen ontvangen. Het Orchard algoritme gebruikt een videotechniek genaamd *Multiple Description Coding* (MDC), waarmee de videostroom opgesplitst kan worden in meerdere substromen. De gebruikers geven vervolgens deze substromen door, zodat er een distributieboom ontstaat voor elke substroom. De substromen worden zodanig uitgewisseld dat gebruikers gedwongen worden evenveel data door te geven als zij ontvangen. Peers die hun bandbreedte niet ter beschikking stellen ontvangen daardoor geen videobeelden. We verschaffen zowel een uitgebreide prestatieanalyse van Orchard als emulatieresultaten, die aangeven dat Orchard goed presteert.

In boom-algoritmes moeten alle gebruikers dezelfde data op hetzelfde moment nodig hebben, wat het geval is voor live videobeelden, maar niet voor vooraf opgenomen videobeelden. Daarom stappen we in Hoofdstuk 3 en 4 over op zwerm-algorithmes. In een zwerm-algoritme wordt de video stroom verdeeld in stukjes die uitgewisseld worden door de gebruikers. Elke gebruiker probeert alle stukjes te bemachtigen, en bewaart ze om ze later aan andere gebruikers te kunnen leveren.

In Hoofdstuk 3 presenteren we het *Give-to-Get* (Geef-om-te-Ontvangen) algoritme voor het verspreiden van vooraf opgenomen videobeelden. Give-to-Get moedigt het doorgeven van de videobeelden aan door eerst de gebruikers te bedienen die de meeste data doorgeven. Profiteurs worden dus als laatste bediend, en ontvangen daardoor alleen videobeelden als er genoeg capaciteit is in het netwerk om ze te kunnen

bedienen. Het voordeel van deze aanpak is dat als er genoeg capaciteit is, ook de gebruikers bediend kunnen worden die wel videobeelden zouden willen doorgeven, maar zo weinig uitgaande bandbreedte hebben of kunnen leveren dat ze niet te onderscheiden zijn van profiteurs. Denk hierbij bijvoorbeeld aan gebruikers met een asymmetrische Internetverbinding, zoals ADSL. Het Give-to-Get algoritme is een uitbreiding op het populaire BitTorrent protocol voor het delen van bestanden. Give-to-Get is daardoor eenvoudig te implementeren door een bestaand BitTorrent programma uit te breiden. Wederom maken we een uitgebreide prestatieanalyse en doen we emulaties.

In Hoofdstuk 4 presenteren we nog een uitbreiding op BitTorrent, maar dan voor het verspreiden van live videobeelden. We projecteren de live videostroom, die oneindig lang kan zijn, op een eindige verzameling stukjes die vervolgens met behulp van BitTorrent uitgewisseld wordt. Ook zijn er veranderingen nodig om de gebruikers in staat te stellen de oorsprong van de stukjes te controleren, die continu oude stukjes vervangen in de eindige verzameling. Uit onze simulaties blijkt dat de prestaties van onze extensie lijden als er veel gebruikers achter een *firewall* of *NAT* zitten. Om de prestaties van onze uitbreiding goed te kunnen bepalen hebben we haar geïmplementeerd in een werkend systeem en gebruikt in een experiment op het Internet, waar mensen van over de hele wereld aan deelnamen. We laten zien dat de prestaties van onze extensie vergelijkbaar zijn met die van experimenten die door anderen gedaan zijn. In ons experiment, net als in de andere experimenten, zat er inderdaad een groot gedeelte van de gebruikers achter een firewall of NAT.

De invloed van firewalls en NATs op de prestaties is het onderwerp van Hoofdstuk 5, waarin we grenzen afleiden aan de prestaties van peer-to-peer distributiealgoritmes die veroorzaakt worden door een beperkte connectiviteit in het netwerk. Een beperkte connectiviteit wordt vooral veroorzaakt door firewalls en NATs, omdat die de gebruikers die zich daarachter bevinden moeilijk bereikbaar maken, tenzij er complexe technieken worden toegepast. Zulke gebruikers kunnen namelijk alleen verbindingen onderhouden die zij zelf hebben opgezet. We bewijzen dat als meer dan de helft van de gebruikers achter firewalls en NATs zitten, sommige gebruikers gedwongen worden om zich als profiteur te gedragen, omdat zij geen verbinding kunnen leggen met gebruikers die data nodig hebben. We analyseren de uitkomst van zowel ons experiment uit Hoofdstuk 4 als van studies van anderen, en laten zien dat inderdaad de meeste gebruikers zich achter firewalls of NATs bevinden. Onze theoretische resultaten aangaande de grenzen aan de haalbare prestaties gelden voor alle peer-to-peer distributiealgoritmes, waardoor zij van pas komen bij het ontwerpen en verspreiden

van peer-to-peer distributiealgoritmes.

In Hoofdstuk 6 geven we onze conclusies, en vatten we de antwoorden op onze onderzoeksvragen samen. Tevens stellen we richtingen voor toekomstig onderzoek voor die voortbouwen op ons werk. De algoritmes die in dit proefschrift gepresenteerd zijn kunnen worden geïmplementeerd om als basis te dienen voor videodistributie over het Internet, en kunnen worden uitgebreid om in meer situaties dan wij konden testen goede prestaties te leveren. Tenslotte kunnen onze grenzen aan de prestaties door een beperkte connectiviteit gebruikt worden als hulpmiddel bij het ontwerpen en verspreiden van zowel videodistributiealgoritmes als algoritmes om bestanden te delen.

Curriculum Vitae

JACOB JAN DAVID MOL was born on 19th September 1978 in Delft, the Netherlands. He grew up in Nootdorp, and finished his secondary education (Gymnasium) at College het Loo in Voorburg in 1996. He went on to study Technical Computer Science at Delft University of Technology, where he received his MSc in 2004 in the Parallel and Distributed Systems group. His master's thesis was on the subject of "Resource Allocation for Streaming Applications in Multiprocessors" and was based on work he performed at Philips Research in Eindhoven.

After obtaining his MSc, Jan David started a PhD track in the same Parallel and Distributed Systems group, which resulted in this thesis. Currently, he has moved to Diever, and is researching and developing the LOFAR software radio telescope at Stichting Astron in Dwingeloo, which in several ways will be the largest radio telescope of the world. His publications to date are the following, in reverse chronological order:

- John W. Romein, P. Chris Broekema, J. Jan David Mol, Rob V. van Nieuwpoort, The LOFAR Correlator: Implementation and Performance Analysis. In *Proc. of the 15th ACM Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2010 (to appear).

- J. Jan David Mol, Arno Bakker, Johan A. Pouwelse, Dick H.J. Epema, Henk J. Sips, The Design and Deployment of a BitTorrent Live Video Streaming Solution. In *Proc. of the 11th IEEE Intl. Symposium on Multimedia (ISM)*, pp. 342–349, 2009.

- J. Jan David Mol, Johan A. Pouwelse, Dick H.J. Epema, Henk J. Sips, Free-riding, Fairness, and Firewalls in P2P File-sharing. In *Proc. of the 8th IEEE Intl. Conf. on Peer-to-Peer Computing*, pp. 301–320, 2008.

- Yue Lu, J. Jan David Mol, Fernando Kuipers, and Piet van Mieghem, Analytical model for Mesh-based P2PVoD. In *Proc. of the 10th IEEE Intl. Symposium on Multimedia (ISM)*, pp. 364–371, 2008.

- J. Jan David Mol, Johan A. Pouwelse, Michiel Meulpolder, Dick H.J. Epema, and Henk J. Sips, Give-to-Get: Free-riding-resilient Video-on-Demand in P2P Systems. In *Proc. of SPIE, Multimedia Computing and Networking Conference (MMCN)*, vol. 6818, article 681804, 2008.

- J. Jan David Mol, Dick H.J. Epema, Henk J. Sips, The Orchard Algorithm: Building Multicast Trees for P2P Video Multicasting Without Free-riding. In *IEEE Transactions on Multimedia*, 9(8):1593–1604, 2007.

- Orlando Moreira, J. Jan David Mol, and Marco Bekooij, Online Resource Management in a Multiprocessor with a Network-on-Chip. In *Proc. of the 22nd ACM Symposium on Applied Computing (SAC)*, pp. 1557–1564, 2007.

- J. Jan David Mol, Dick H.J. Epema, Henk J. Sips, The Orchard Algorithm: P2P Multicasting Without Free-riding. In *Proc. of the 6th IEEE Intl. Conf. on Peer-to-Peer Computing*, pp. 275–282, 2006.

- Orlando Moreira, J. Jan David Mol, Marco Bekooij, and Jef van Meerbergen, Multiprocessor Resource Allocation for Hard-real-time Streaming with a Dynamic Jobmix. In *Proc. of the 11th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 332–341, 2005.