Challenges, Design and Analysis of Peer-to-Peer Video-on-Demand Systems

LUCIA D'ACUNTO

g1045)

Challenges, Design and Analysis of Peer-to-Peer Video-on-Demand Systems

Lucia D'Acunto

Stellingen

Challenges, Design and Analysis of Peer-to-Peer Video-on-demand Systems

Lucia D'Acunto

June 7, 2012

1. Het incentive mechanisme van BitTorrent is niet geschikt voor video-on-demand. [*Dit proefschrift*]

2. In een P2P swarming systeem worden de prestaties van een peer achter een firewall of NAT negatief beïnvloed in het voordeel van peers die algeheel bereikbaar zijn. [*Dit proefschrift*]

3. P2PVoD systemen zijn niet eindeloos schaalbaar in het geval van een flashcrowd. [*Dit proefschrift*]

4. In elk praktisch uitvoerbaar systeem kan worden ingebroken.

5. Het geven van de juiste hoeveelheid begeleiding aan promovendi is een uitdaging: te weinig begeleiding leidt tot richtingsloosheid, terwijl te veel begeleiding hun creativiteit kan beperken.

6. De grootte van een onderzoeksgroep is geen garantie voor onderzoek van hoge kwaliteit. Slimme mensen, zorgvuldige organisatie en de juiste visie zijn de belangrijkste succesfactoren.

7. Niet anonieme recenties zal de kwaliteit van de geleverde aanmerkingen op wetenschappelijke artikelen verbeteren.

8. Communicatie is belangrijker dan intelligentie.

9. In Nederland heb je altijd tegenwind, onafhankelijk van de richting waarin je fietst.

10. Als lekker Nederlands eten bereid wordt door een Italiaan is het nog steeds heerlijk. Andersom is dat niet noodzakelijk waar.

Deze stellingen worden opponeerbaar en verdedigbaar geacht en zijn als zodanig goedgekeurd door de promotor, Prof.dr.ir. H.J. Sips.

Propositions

Challenges, Design and Analysis of Peer-to-Peer Video-on-demand Systems

Lucia D'Acunto

June 7, 2012

1. The incentive mechanism of BitTorrent is not suitable for video-on-demand. [*This thesis*]
2. In a P2P swarming system, residing behind a firewall or a NAT reduces a peer's performance much to the benefit of those who are globally reachable. [*This thesis*]
3. P2PVoD systems cannot scale indefinitely when hit by a flashcrowd. [*This thesis*]
4. Every practically feasible system can be hacked.
5. Giving the right amount of supervision to PhD students is a challenging task: too little can lead them out of track, while too much can inhibit their creativity.
6. The size of a research group is not a guarantee for high quality research. Smart people, neat organization and the right vision are the keys.
7. Non-anonymous reviews will improve the quality of the comments provided to a scientific article.
8. Communication is more important than intelligence.
9. When you bike in the Netherlands you have the wind always against, independently from the direction you choose.
10. Delicious Dutch food cooked by Italian people is still delicious. The opposite is not necessarily true.

These propositions are regarded as opposable and defendable, and have been approved as such by the supervisor, Prof.dr.ir. H.J. Sips.

Challenges, Design and Analysis of Peer-to-Peer Video-on-Demand Systems

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 donderdag 7 juni 2012 om 15:00 uur

door **Lucia D'ACUNTO**

ingenieur in de technische informatica, University of Naples Federico II
geboren te Nocera Inferiore, Italy

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. J.A. Pouwelse	Technische Universiteit Delft, copromotor
Prof.dr. P. Rodriguez	Telefonica Research, Spain
Prof.dr. Y. Dai	Peking University, China
Prof.dr.ir. B.R.H.M. Haverkort	Universiteit Twente
Prof.dr. K.G. Langendoen	Technische Universiteit Delft
Prof.dr.ir. P.F.A. Van Mieghem	Technische Universiteit Delft
Prof.dr.ir. G.J.P.M. Houben	Technische Universiteit Delft, reservelid

Published and distributed by: Lucia D'Acunto
E-mail: lucia.dacun@gmail.com

Keywords: Peer-to-Peer systems, Video-on-Demand, Quality-of-Service, BitTorrent, Modeling, Simulation.

Printed in The Netherlands by: Wöhrmann Print Service.

Advanced School for Computing and Imaging

P2P NEXT

The work described in this thesis has been carried out in the ASCI graduate school. ASCI dissertation series number 256.

This work was supported by the European Community's Seventh Framework Programme through the P2P-NEXT project (grant no. 216217).

Acknowledgments

I would like to start by thanking my PhD promotor, Henk Sips. Henk, thank you for having always encouraged me during my PhD. Your positive attitude and support motivated me to keep going. I'm also thankful for the many career advices and contacts you gave me towards the end of my PhD and for the nice talks about non-work-related topics, like languages and sailing.

Johan, I would like to thank you for the freedom you gave me in my research. Thanks for not insisting for me to work on a brand new topic every other day and instead letting me focus on developing my own ideas.

I'm also grateful that I had the chance to work on a daily basis with Tamás. Although not being officially my supervisor, you were always there to discuss with me work-related as well as non-work-related topics. I always felt understood and supported by you. You are a very kind and sensible person and I was lucky to have your friendship. I wish you the best luck for your academic career in Hungary.

Michel, I would like to thank you for helping me getting started with doing scientific research. You guided me with writing my first peer-reviewed paper. Thanks to your help, our paper was accepted and I got the necessary confidence that I could complete a PhD track. Also, I would like to thank you for your post-PhD advices, which helped me in getting my current job position.

Rameez, thanks for bringing a different view in our group. Your original ideas helped me to think out of the box, and influenced much of the content of this thesis. I also enjoyed a lot the late-afternoon talks (and jokes) we had, together with Tamás. I hope that everything goes alright with your new life in Pakistan.

Mihai, thanks for being a loyal office-mate, and for warning anyone from sitting at my desk in my absence. Also, I'm grateful you were always available to review my work and talk about many other topics, like learning dutch and finding girlfriends.

Adele, thank you for involving me in the early stages of your work, and for the many gifts from China. I will also remember all the hours spent making Totoro plushies with you and Jie. It was hard work but in the end it was worth it!

Nitin, thanks for being a good listener. I have always found our chats very useful. You are one of the best young researchers that I know and I'm confident you'll be having a brilliant research career.

Boudewijn, thanks for being patient with me and always keen to discuss my research topics every now and then. I had a good time with you and I really appreciated your advices.

Niels, it was nice to have you in the office for over a year. Thanks for helping me with many dutch-related issues and I hope you are doing fine in your new office.

Rahim, I would like to thank you for your friendliness. I also very much appreciated the Iranian cookies you always brought to me. I will miss them!

Dimitra and Riccardo, thanks for the many coffees and chats at the Aula's café.

Flutra and Boxun, thank you for the times we went running after work. It was nice to break from the office routine.

Arno, thanks for always taking care that we wouldn't mess up with the Tribler code, and also for all the interesting chats at lunch time.

Andrei, thanks for the many late-afternoon stories. now keep me updated via mail!

I also need to thank some of our former colleagues. Dave, thanks for aways pushing us to focus on research. Naza, thank you for introducing our group to the Brazilian way of living, your parties were always lots of fun. Viktor, thanks for having been the weird one, and for starting the Thursday muffin tradition, which I kept alive after you were gone.

Paulo, thank you for always taking care of the many technical and DAS-related issues I encountered during my PhD. I would also like to thank Munire, for fixing my computer when it broke down one month prior the completion of my thesis, and Rina, for being always kind and cheerful.

Loubna, thanks for choosing me for the "instemming" in my first house. It was nice to be your neighbor for over a year, sharing the joys and sorrows of doing a PhD in front of a cup of Arabic coffee. I will miss you when you are gone and I wish you good luck with your new life in UK.

To the younger PhD students: Bogdan, Jie, JianBin, Yong and Siqi, it was nice meeting you and I wish you to have as a great time in your PhD as I did. Good luck to all of you!

I also owe a lot to my parents, who supported me in the decision of coming to live abroad. I know it must have been very hard for you to separate from your daughter, and I thank you for understanding and supporting my choices, as well as for the distant help you have always provided during the years.

I would also like to thank my sister Erica, who, although being the younger sister, very often acted as the elder of the two, comforting me when I needed it and giving me her wise advices. I'm also very happy that you moved to Utrecht, so we can share the joys and sorrows of living in the Netherlands.

Finally, I would like to thank Jelle for his constant support during all these years. Thanks for being always there listening, and putting up with my mood swings and depressed times. I know that it must have been hard for you sometimes, and I'm thankful you never gave up.
Love, Lucia.

Contents

Chapter 1

Introduction

In recent years, significant research effort has focused on the effective use of a peer-to-peer (P2P) architecture to provide large-scale video-on-demand (VoD) services [13,20,26,30,65]. Current major VoD services like YouTube, Hulu, and NetFlix, which supply hundreds of thousands of videos to millions of users everyday, make massive use of central servers for content provisioning. However, a P2P-based approach, with its natural scalability, could drastically cut the costs of service providers. It has been demonstrated that, for instance, the MSN video server load can be reduced by roughly 95% through the use of a P2P-based technology [19]. Furthermore, non-profit organizations like Wikipedia, that cannot afford deploying the necessary servers to provide large-scale on-demand media services, would greatly benefit from such an approach [18].

Despite the potential benefits, providing on-demand services using a P2P architecture is also a challenging task. In fact, similar to the P2P live streaming case, some quality-of-service (QoS) requirements have to be fulfilled, i.e. providing users with a high playback continuity and a short startup delay. However, the data access pattern in the two cases is different. In live streaming, peers have a shared temporal content focus, meaning that they are interested in watching the same part of the video file, independently from their joining time. This gives peers a number of opportunities to collaborate among each other to download a stream. On the other hand, in the VoD case, users requesting the same video at different times will be interested in viewing different parts of it. As a consequence, it is more difficult for peers to help each other and offload the service provider.

Since many of the early P2P systems have been built to efficiently distribute large files among the participating users, trying to adapt the designs of these systems to the video streaming case came as a natural choice. In particular BitTorrent, which has been shown to make nearly optimal use of peers' upload bandwidth [14], has inspired many P2P protocols for VoD [10,20,33,51,52,63]. In the context of traditional file-transfer, BitTorrent achieves a high utilization of peers' upload bandwidth by means of a smart file retrieval mechanism and contribution-based downloader selection. However, BitTorrent was not originally created for streaming and adapting its design to VoD poses two conflicting goals: satisfying the fundamental QoS requirements for streaming, while still maintaining the high efficiency of the original BitTorrent protocol. For this reason, previous work on this matter has mainly focused on BitTorrent's file retrieval strategy: the file should be received in a somewhat sequential fashion to allow a view-as-you-download functionality, but different peers should own different parts of it, in order to help each other.

In this thesis, we show that, to provide the necessary QoS to the users, BitTorrent's peer selection mechanism should also be modified for its use in VoD. In fact, this mechanism, based on bandwidth reciprocity between two peers, poses further challenges in heterogeneous P2PVoD systems. When peers have different upload bandwidths, for example, it can happen that nodes with low upload capacities cannot reach a download speed high enough to sustain the video playback rate, while those with higher upload capacities download at rates much higher than actually needed. Likewise, peers residing behind NATs or firewalls blocking unsolicited inbound connections might experience poor performance, much to the benefit of fully connectable nodes. In this thesis, we propose a solution that allows peers to *"relax" their reciprocity-based mechanism when they are already experiencing a good enough service*, consequently serving low-capacity or unconnectable peers when enough bandwidth is available.

We further focus on the problems introduced by a sudden surge of new nodes joining a P2PVoD system, a phenomenon known as *flashcrowd*. We show, both theoretically and experimentally, that flashcrowds can drastically reduce the scalability and the QoS provided by P2PVoD systems. Based on this analysis, we propose a number of flashcrowd-handling algorithms that significantly improve peer QoS during flashcrowds.

1.1 P2PVoD Systems

In this thesis, we aim at providing insights into the challenges and the design of P2P systems for on-demand video streaming. We especially focus on a BitTorrent-based design, as BitTorrent has proven to be very efficient in utilizing peer bandwidths, while requiring only little effort from the deployer, both from a management and a bandwidth supply point of view. Furthermore, our study is bounded to the problem of distributing a single video file, although several of our contributions can offer insights that apply to multi-swarm (or multi-channel) systems. In the following paragraphs, we first introduce the requirements for a P2PVoD system, and then we proceed with an overview of BitTorrent and BitTorrent-based P2PVoD systems.

1.1.1 Fundamental requirements for VoD

Any VoD system should try to meet, for each of its users, the following QoS requirements:

QoS requirement 1: smooth playback continuity

QoS requirement 2: short startup delay

A smooth playback continuity means that the stream must be received at least as fast as it is viewed. Hence, a VoD system should try to maximize the number of downloaders that maintain a download rate of at least R, where R is the playback rate of the video file distributed in the system. We argue that this should be the primary goal of such a system, because allowing a peer to start the playback implies a commitment to provide a satisfactory playback continuity to that peer. On the other hand, no commitment has been established with a peer who has just joined and has not started viewing the video yet. Hence, when the bandwidth is scarce (e.g. during a sudden surge of requests), it is more important to serve those peers that have already started playing, rather than allowing new nodes in the system. Furthermore, by doing so, we also avoid allowing in the system peers whose playback continuity cannot be guaranteed due to bandwidth scarcity. On the other hand, when there is enough bandwidth available, the secondary goal a VoD system is minimizing the startup delay of joining peers.

1.1.2 Scalability of P2PVoD systems

P2P systems are based on the principle that users are both *consumers* and *producers* of resources. In the specific case of P2PVoD systems, the primary resource is bandwidth. Since users need to receive the video stream at a certain minimum rate R to guarantee smooth playback continuity (1st QoS requirement for VoD), it follows that, for the P2PVoD system to be scalable, the peers should contribute enough bandwidth to allow each downloader achieve a download rate of at least R. We note that, while it is possible that some peers will remain in the system after their download is complete and offer bandwidth without consuming any, their actual presence and amount contributed are highly unpredictable, being dependent on user behavior. Therefore, in this thesis, we only consider scalable those systems where the bandwidth offered by the downloaders is enough to sustain all of them at the video playback rate. Specifically, having defined the average peer upload bandwidth with μ, in this thesis we focus on *scalable P2PVoD systems* where, by definition, $R \leq \mu$ holds.

1.1.3 BitTorrent

BitTorrent is a widely popular P2P protocol for file-transfer. In BitTorrent, files are split into pieces, allowing peers which are still downloading content to serve the pieces they already have to others. Corresponding to each file available for download, there is a central component called *tracker* that keeps track of the nodes currently in the system. When a new peer joins, it contacts the tracker to obtain a list of a random subset of these nodes. Each node establishes persistent connections with a large set of peers (typically between 40 and 80), called its *neighborhood*, and uploads data to a subset of this neighborhood.

Specifically, each peer equally divides its upload capacity into a number of *upload slots*. There are two types of upload slots: *regular unchoke slots* and *optimistic unchoke slots*. Regular unchoke slots are assigned according to a strategy based on direct reciprocity: peers prefer other nodes that have recently provided data to them at the highest speeds. Each peer re-evaluates the allocation of its regular unchoke slots every *unchoke interval* δ (generally 10 seconds). Different from the regular unchoke slots, the optimistic unchoke ones are assigned to randomly selected nodes. Also, their allocation is re-evaluated every *optimistic unchoke interval*, which is generally set to 3δ. Optimistic unchoke slots serve the purposes of (i) having peers discover new, potentially faster, nodes to unchoke so as to be reciprocated, and (ii) bootstrapping newcomers (i.e. peers with no pieces yet) in the system. The number of upload slots opened by each peer is usually a fixed constant [9,21,42].

Each peer keeps its neighborhood informed about the pieces it owns. The information received from this neighborhood is used to request pieces of the file according to the Local Rarest First policy. This policy determines that each peer requests the pieces that are the rarest among its neighbors. The emergent effect of this *piece selection policy* is that less-available pieces get replicated fast among peers and each peer obtains first the pieces that are most likely to interest its neighbors [8].

1.1.4 BitTorrent-based P2PVoD

Prior work on adapting BitTorrent to VoD mainly addresses piece selection [10,20,33,51,52,63]. These studies focus on finding a policy that can achieve a good trade-off between the need of sequential download progress, needed for stream continuity, and high piece diversity, needed for efficiency. Although the approaches proposed to tackle this problem vary, they all have in common that the resulting mechanism combines in-order piece selection with rarest-first piece selection. Furthermore, Carlsson *et al.* [48] also propose a policy to reduce startup delays where newly joined peers are prioritized, and Yang *et al.* [65] study a number of strategies that a peer can use to choose which other node a piece request should be sent to, in order to balance the load among requested nodes and increase the likelihood of receiving the needed pieces before their playback is due.

1.2 Challenges in the design of P2PVoD systems

The decentralized, dynamic, and heterogeneous nature of P2P, together with the QoS requirements of a VoD application, poses several challenges in the design of P2PVoD systems. In this section, we take a close look at these challenges, with a particular focus on those faced by a BitTorrent-based P2PVoD design.

1.2.1 Incentives for contribution

The efficiency of P2P systems comes from users contributing their resources (files and bandwidth). However, studies have shown that there are many users who are not willing to do so, e.g. in Gnutella [11]. Therefore, a lot of effort in the P2P research community has focused on how to incentivize user contribution [21,25,42,43,46,55].

While most of these works revolve around file-transfer, it is important to notice that a high level of user contribution is even more important for P2PVoD systems, due to the QoS requirements of VoD. In fact, as was demonstrated by Habib *et al.* [28], the QoS experienced by peers at low levels of contribution is often below the requirements of a VoD application. Since BitTorrent has been shown to achieve high levels of user contribution [14], its incentive mechanism, based on direct bandwidth reciprocity between two interacting peers, has been considered for P2PVoD designs as well [10,20,51,52,63]. However, Mol *et al.* [33] have argued that this mechanism is not suitable for VoD applications because, due its somewhat sequential download nature, it may be difficult for peers with lower level of download progress to reciprocate peers with higher level of progress. Therefore, they proposed a different incentive scheme based on *indirect reciprocity*, in which peers prefer uploading to other nodes that have *forwarded* pieces to others at the highest rate in the past. In this thesis, we demonstrate that neither of the two incentive mechanisms is perfectly suitable to VoD, and explore methods that better match with the QoS requirements of the application.

Finally, we note that a number of other, more sophisticated, incentive mechanisms have been proposed, which employ accounting systems to keep track of the contribution of each user (e.g. [28, 61]). However, the presence of an accounting system, be it centralized or distributed, introduces several other challenges, with respect to system management, information dissemination, and security, which fall outside the scope of our work. Therefore, we do not consider those designs in this thesis.

1.2.2 Peer heterogeneity

Internet access technology has become extremely heterogeneous nowadays. Different access means, as well as different ISP infrastructures, result in a high bandwidth heterogeneity among Internet hosts. Furthermore, recent times have also witnessed an increase in the usage of firewalls and Network Address Translators (NATs), which cause many hosts not to be reachable at any routable address. In this section, we briefly discuss the implications of this heterogeneity for BitTorrent-based P2PVoD systems.

Bandwidth heterogeneity

The incentive schemes introduced in the previous subsection are based on some kind of bandwidth reciprocity between interacting peers. This directly affects the download performance of the users, when they have different upload capacities. In fact, it has been observed that the emergent behavior of such incentive mechanisms is that peers with higher upload capacities typically achieve higher download speeds [9]. This is a desirable behavior in the context of traditional file-transfer, because users wish to complete their downloads as soon as possible, incentivizing them to contribute as much bandwidth as possible. However, we note that, in a VoD application, users gain little utility from having a download speed higher than the necessary to preserve stream continuity. Therefore, using this kind of incentives in P2PVoD systems where peers have highly heterogeneous bandwidths can lead to situations where, even if there is enough aggregate upload capacity to serve all peers in the system, not all peers experience a satisfactory QoS. This happens when high-capacity peers receive download rates substantially higher than the video playback rate, while low-capacity peers experience download rates that are too low to allow a smooth playback. Thus we need incentive schemes which explicitly take VoD into account.

Connectability

NATs and firewalls are becoming a default setting among home users. However, these devices represent an obstacle to the operation of P2P applications, since their presence causes the hosts sitting behind them not to be reachable at any routable address, unless the specific NAT/firewall has been configured to do so. For P2P swarming systems, where peers collaborate to download a file, it means that not every peer is able to exchange data with every other peer in the system, and this can create an unwanted imbalance in how bandwidth is distributed across the users. An immediate consequence for P2PVoD systems is that the QoS provided to the users will also be similarly imbalanced. Since these *unconnectable* nodes represent a considerably large amount of P2P users (between 40% and 60%, according to recent measurement studies [32,45,47]), the bandwidth dynamics and, consequently, the QoS performance of P2PVoD systems will likely be dominated by the effects of their presence.

1.2.3 Flashcrowds

Peer dynamics have always been an issue for P2P systems. In particular, a *flashcrowd*, i.e. a sudden surge of peers requesting content, could drastically affect the scalability and performance of P2P systems, since the incoming peers all demand for bandwidth while having little or nothing to give in return. The impact of these phenomena is even more severe for P2PVoD systems, because of the need of fulfilling QoS requirements. It is evident that, accommodating such a large number of newcomers within a stringent time constraint, while still maintaining the QoS of existing users high, is extremely difficult.

1.3 Problem statement

In this thesis, we address the research problem of how to deliver high quality on-demand video streaming using a decentralized P2P infrastructure that can be heterogeneous and highly dynamic as well. In order to accomplish this, we need to explore which factors pose challenges to the design of P2PVoD systems and analyze their impact. This high-level problem statement can be detailed in the following research questions:

How do NATs and firewalls affect the performance of P2P swarming systems? As mentioned earlier (Section 1.2.2), NATs and firewalls represent a major obstacle to the operation of P2P applications. While it is clear that the hosts residing behind NATs or firewalls that have not been configured to receive inbound connections cannot be directly contacted by other hosts in the Internet, it is not yet well understood how this phenomenon affects the performance of P2P swarming systems, where peers need to collaborate to download a file of interest. The fact that not every peer is able to exchange data with every other peer may have implications on how bandwidth is distributed across the users. For P2PVoD systems, this means that the QoS provided to each user may also be affected by that specific user being or not connectable. The percentage of peers residing behind a NAT or firewall may also influence how bandwidth is distributed and, ultimately, users QoS. Therefore, in order to understand and improve the performance of P2P swarming systems in general and P2PVoD systems in particular, it is important to understand how the presence of these unconnectable peers affect the bandwidth dynamics in these systems.

How do currently proposed BitTorrent-based P2PVoD protocols work in practice? Due to its high efficiency, BitTorrent has inspired a number of P2P protocols for VoD. Although the approaches followed in these P2PVoD proposals vary, they all strive to achieve a good trade-off between the individual need of sequential download progress (needed for streaming) and high piece diversity (needed for efficiency). However, we note these approaches have been evaluated under different circumstances, and in limited scenarios. Hence, some methods might perform better than others under a certain set of conditions and worse under another. Furthermore, it is unclear how well each of them would work in real world conditions, where, for instance, peers have heterogeneous bandwidths and may reside behind a NAT or a firewall. Similarly, it is still unknown to what extent each approach really maintains the original BitTorrent's incentives for cooperation. Exposing the pros and cons of each approach can guide in selecting the most appropriate protocol to use in a given environment. Likewise, understanding the behavior of different BitTorrent-based P2PVoD protocols will help researchers and system designers in improving current approaches and/or designing better ones.

Is it possible to improve the QoS of heterogeneous BitTorrent-based P2PVoD systems while retaining the original BitTorrent's incentive for cooperation? As mentioned in Section 1.2.2, the use of BitTorrent's original incentive mechanism in a VoD application can lead to a situation where high-capacity peers receive download rates substantially higher than the video playback rate, while low-capacity peers experience download rates that are too low to allow a smooth playback. Similarly, peers behind a firewall or NAT can experience much lower performance as compared to fully connectable nodes. On the other hand, having high-capacity (fully connectable) peers altruistically "help" lower-capacity (unconnectable) ones will weaken the incentive to contribute: peers may easily freeride by pretending of having a low bandwidth capacity or residing behind a NAT or firewall, thereby leading to a "tragedy of the commons". The challenge is to make peers help those in need, while still guaranteeing that contributing more bandwidth or being fully connectable are attractive properties.

How do BitTorrent-based P2PVoD systems scale during flashcrowds? Flashcrowds, i.e. sudden surges of newly joined peers, may have serious impact on P2PVoD systems, since the newcomers all demand for bandwidth while having not much to give in return. Therefore, knowing which parameters influence the scalability of these systems, as well as how they do so, is important for the appropriate management of P2PVoD systems.

Can decentralized BitTorrent-based P2PVoD systems provide high QoS under flashcrowds? P2P systems are decentralized entities, where each peer acts individually towards reaching a certain goal. Therefore, it is natural to wonder whether the nodes in such systems are able to quickly detect and respond to a sudden surge of newcomers, and distribute the limited available bandwidth in such a way to keep providing a good QoS to existing users. Furthermore, it is also interesting to investigate the role of the initial source of the video file, which is one of the few peers possessing a complete copy of the file and often can be the bottleneck in the file distribution.

1.4 Research contributions and thesis outline

The contributions of this thesis are the following:

Modeling and analysis of the peer connectability problem in P2P swarming systems (Chapter 2) We present a mathematical model to study the performance of a P2P swarming system in the presence of unconnectable peers. We quantify the average download speeds of peers and find that unconnectable peers achieve a *lower* average download speed compared to connectable peers, and this difference increases hyperbolically as the percentage of unconnectable peers grows. More interestingly, we notice that connectable peers actually *benefit* from the existence of peers behind NATs/firewalls, since they alone can enjoy the bandwidth that those peers offer to the system. Inspired by these observations, we propose a new policy for the allocation of the system's bandwidth that can mitigate the performance issues of unconnectable peers. In doing so, we also find an intrinsic limitation in the speed improvement that they can possibly achieve. This chapter is largely based on our work published in [36].

BitTorrent-based P2P approaches for VoD: a comparative study (Chapter 3) We propose a simulation based methodology which aims at putting forward a common base for comparing the performance of different BitTorrent-based P2PVoD protocols under a wide range of conditions. We show that, despite their considerable differences, existing approaches all share some characteristics, such as that their bandwidth reciprocity based methods to incentivize cooperation do not always yield an optimal overall performance. Furthermore, we demonstrate that in these protocols there is a trade-off between QoS and resilience to freeriding and malicious attacks. Overall, our findings provide important implications for both service providers and future system designers. On the one hand, our results can guide service providers in the selection of the most appropriate protocol for a given environment. On the other hand, exposing the flaws of current approaches will help researchers in improving them and/or designing better ones.

Peer selection strategies for improved QoS in heterogeneous BitTorrent-based P2PVoD systems (Chapter 4) In this chapter, we extend the original peer selection mechanism of BitTorrent with techniques that allow peers to relax their reciprocity-based peer selection and choose more random nodes when their current QoS is high. In this way, more peers can be granted a good QoS and freeriding is tolerated only when bandwidth resources are abundant. Similarly, unconnectable peers, who would typically be penalized much to the advantage of connectable ones, can also benefit from such an approach. To demonstrate the benefits of our solutions, we present extensive simulations of the introduced techniques. This chapter is largely based on our work published in [37].

Analyzing the scalability of BitTorrent-based P2PVoD systems during flashcrowds (Chapter 5) We analyze how the scalability of a general BitTorrent-based VoD system is affected by a flashcrowd. Our study shows that, at the very beginning of a flashcrowd, the system scale is intrinsically related to two fundamental system parameters, (i) the initial service capacity and (ii) the efficiency of piece exchange of the underlying P2P protocol. Finally, we illustrate the impact of peers turning into seeders (i.e peers that have finished downloading and remain in the system to upload) on the system scale. This chapter is largely based on our work published in [39].

Bandwidth allocation in BitTorrent-based P2PVoD systems during flashcrowds (Chapter 6) We analyze how bandwidth should be allocated in a BitTorrent-based P2PVoD system hit by a flashcrowd in order to fulfill the QoS requirements of the application. In particular, we find that there is a limit in the number of peers that can be admitted in the system over time, if we want to guarantee a smooth playback continuity to previously joined users. Furthermore, we show that there is a trade-off between having the initial source minimize the upload of pieces already injected recently and high peer QoS. Based on the insights gained from our analysis, we devise some *flashcrowd-handling* algorithms for the allocation of peer bandwidth to improve peer QoS during flashcrowd. We validate the effectiveness of our proposals by means of simulations. This chapter is largely based on our work published in [38].

Conclusions and future work (Chapter 7) We summarize our most important conclusions in the last chapter, and provide suggestions for future work based on our findings.

Chapter 2

Modeling and analysis of the peer connectability problem in P2P swarming systems

Over the last years, there has been a significant rise in the use and development of P2P technology, and a large number of analytical and measurement-based studies have been conducted on analyzing the properties and the performance of P2P networks in general, as well as in specific applications, such as file sharing [41, 53, 66] and video streaming [15, 32–34]. However, recent times have also witnessed an increase in the usage of firewalls and NATs. These devices break the original model of IP end-to-end connectivity across the Internet, causing many peers not to be reachable at any routable address, and therefore introducing complications in P2P communication. Nevertheless, so far little research has been conducted on how the problems that firewalls and NATs cause to P2P communication affect different P2P systems. Mol *et al.* [47] show that the presence of NATs/firewalls in a P2P file-sharing system correlates to the phenomenon of free-riding and suggest that unconnectable peers might expect to achieve lower performance. Furthermore, they measured a high fraction (above 60%) of unconnectable peers in both a popular public BitTorrent community and in a P2P streaming system [32], which is also confirmed by other recent studies [17, 45, 60]. Skevik *et al.* [35] crawled several BitTorrent swarms and discovered that unconnectable peers have a lower average speed than connectable peers. The model proposed by Liu *et al.* [24] shows evidence that, in BitTorrent, the performance of peers is affected by their connectability.

In this chapter, we extend this line of work and present a study of the download performance of a generic P2P swarming system in the presence of unconnectable peers. Our analysis aims at (i) providing a better understanding of the properties (and predicting the behavior) of P2P systems in a realistic environment (that includes the presence of NATs and firewalls), so to stimulate the (ii) design of P2P protocols that improve peer QoS. The contributions made in this chapter are the following:

1. We present a general model that captures the effects of NATs and firewalls in a broad class of P2P swarming systems, including the most popular file-sharing applications, like BitTorrent [21] and eDonkey [29], as well as a number of streaming systems derived from BitTorrent – e.g. BitTos [10], SwarmPlayer [33] (Section 2.4).

2. We show that the difference in speed between connectable and unconnectable peers is a function of the fraction of unconnectable peers and increases hyperbolically as this fraction grows. Also, we show that connectable peers actually *benefit* from the coexistence with unconnectable peers (Section 2.5).

3. We propose a different bandwidth allocation policy that tries to reduce the performance gap between the two types of peers and provide better QoS to unconnectable peers; in doing so we discover that an upper bound exists for the improvement that can be achieved by means of *any* bandwidth allocation policy (Section 2.7).

2.1 Problem description

This section provides a short description of NATs and firewalls and the problems they cause to P2P communication. Also, some techniques that have emerged to circumvent these issues and their applicability are discussed.

2.2 NATs and firewalls

A *Network Address Translator* (NAT) is a device that allows multiple machines on a private network to communicate with the Internet using a single globally unique IP address. This is accomplished by modifying the network information (namely IP address and port) in the packets that transit through it. For an internal host to be able to receive packets from outside, a mapping must be created in the NAT (i.e. the NAT must allocate an external port for the traffic directed/generated by that internal host). This is dynamically done by the NAT itself when an internal host initiates a connection. Unsolicited inbound connection attempts are dropped, since the NAT has no way of knowing to which internal host the packet should be forwarded to. Even though NATs were originally introduced as a temporary solution for alleviating the IPv4 address shortage, they are still widely used and will probably continue to be part of the Internet in the future.

A *firewall* is a device that inspects the network traffic passing through it and filters (i.e. denies or permits) the transmission based on a set of rules. In its most general implementation, a firewall allows outbound connections but blocks inbound connections.

In the rest of this chapter we will, for ease of discussion, use the term *unconnectable* for peers, either behind a firewall or a NAT, whose firewall/NAT is not configured to accept inbound connections. Likewise, we will use the term *connectable* to identify the peers who are able to accept inbound connections. Finally, we will refer to *connectability* as the property of a peer of being either unconnectable or connectable.

2.3 Workarounds

Many workarounds to the NAT/firewall problem have been proposed:

- **manual configuration**. In some cases it is possible to configure NATs and firewalls so to receive inbound connections, for example by having the user manually enable connections on certain ports.
- **UPnP**. Another way to permit inbound connections is by means of UPnP [5]. UPnP is a standard protocol that allows NATs and firewalls to open ports and route certain traffic to certain hosts automatically.
- **hole punching**. Techniques known as *hole punching* or *NAT traversal* can be used to establish a connection between two unconnectable peers. These techniques generally require the assistance of a connectable third party (either a well-known server or another peer), having already a connection with both the unconnectable peers. The third party provides each peer with the other peer's endpoint information (i.e. the external address and port allocated by its NAT) and then both peers initiate a connection at the same time. In this way each peer's NAT/firewall will allow the packets coming from the other peer, if they are considered to be part of a locally initiated connection.

2.3.1 Open issues

The presented workarounds happen to fail in various cases and for different reasons:

- **manual configuration**. Home users are often unaware that their NAT/firewall limits the capability of receiving incoming connections or do not have the knowledge on how to configure it to allow incoming traffic.
- **UPnP**. A recent measurement study [60] shows that the percentage of users having UPnP enabled is below 20%. Furthermore, the NATs and the firewalls using it are vulnerable to attacks, since UPnP lacks a standard authentication method.
- **hole punching**. Many approaches have been proposed to implement hole punching [7, 16,58], but the results vary in relation to the particular NAT vendor, since NAT/firewall

Table 2.1: Model parameters.

Notation	Definition
n	number of classes of peers.
x_i	number of leechers in class i.
y_i	number of seeders in class i.
B_{ji}	bandwidth allocated to class i by peers in class j.
B^u_{ji}	bandwidth allocated to class i by unconnectable peers in class j.
B^c_{ji}	bandwidth allocated to class i by connectable peers in class j.
b_{ji}	average download speed of a peer in class i due to the contribution of peers in class j.
b^u_{ji}	average download speed of an unconnectable peer in class i due to the contribution of peers in class j.
b^c_{ji}	average download speed of a connectable peer in class i due to the contribution of peers in class j.
α_i	fraction of unconnectable peers in class i, constant among leechers and seeders.
δ_{ji}	fraction of B^c_{ji} that is allocated to the group of unconnectable leechers in class i.

behavior is not standardized, as well as in relation to the transport protocol used (UDP or TCP) [16]. Furthermore, some NATs and firewalls do not allow hole punching at all for security reasons.

To conclude, none of the techniques proposed in the literature so far solves the NAT/firewall problem completely.

2.4 Modeling unconnectable peers

In this section, we present a mathematical model for a P2P swarming system in which a fraction of peers are unconnectable. Our model is general enough to capture the main aspects of different swarming systems, and as such can be straightforwardly applied to popular file-sharing P2P applications, like BitTorrent [21] and eDonkey [29], and to many streaming systems derived from BitTorrent (e.g. BitTos [10], SwarmPlayer [33]).

In our analysis, all peers are able to initiate outgoing connections, whereas connectable peers can accept incoming connections and unconnectable peers can not. Therefore we assume that an unconnectable peer u can only be connected to a connectable peer c, by having the connection originate from u.

2.4.1 Notation and assumptions

We consider a system where peers are sharing a single file. We assume the file to be partitioned into pieces, allowing multi-part downloading, thus peers which are still in the process of downloading, but already have part of the file, can serve this part to others. We refer to these peers as *leechers*, while we call *seeders* those that have finished downloading the file

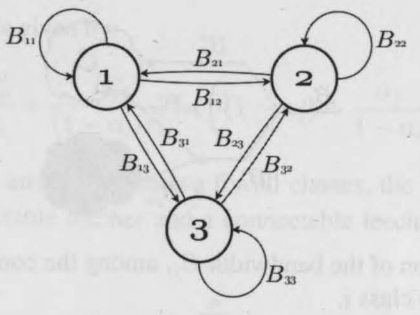


Figure 2.1: The allocation of bandwidth in a P2P swarming system with 3 classes.

and still remain in the system, contributing to the system's bandwidth capacity. We study peers' performance when the system is in steady state (i.e. when the number of peers in the system remains constant).

In accordance with most of the recent Internet access technologies (e.g. ADSL) and measurement studies of existing P2P systems [41], we make the common simplification that the download capacity is not a bottleneck.

We model the P2P system as a multi-class network. A class is a set of peers which are considered *equal* with respect to the particular system, both in the bandwidth they get and in the bandwidth they provide. Classes are a useful feature which allows our model to be easily applicable to many P2P swarming systems. For instance, in BitTorrent, classes can be groups of peers having similar upload capacities, as the peer selection mechanism employed by each peer can not distinguish between two peers having roughly the same bandwidths [21].

In our analysis we make the assumption that each class i receives a given amount of service from each other class j, in terms of download bandwidth, and provides in return a given amount of upload bandwidth. We define B_{ji} as the bandwidth that class j allocates to class i and b_{ji} as the average download speed that a peer in class i gets from peers in class j.

The notation of our model is reported in Table 2.1 while Fig. 2.1 illustrates the allocation of bandwidth in a P2P system with of 3 classes.

2.4.2 Bandwidth distribution and download speed

The total download speed d_i of a leecher in class i is determined by the contributions it gets from the peers in all classes, i.e.:

$$d_i := \sum_{j=1}^{n} b_{ji}. \tag{2.1}$$

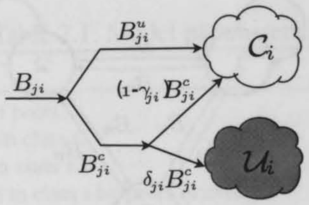


Figure 2.2: The allocation of the bandwidth B_{ji} among the connectable ($\mathcal{C}_i$) and the unconnectable ($\mathcal{U}_i$) leechers in class i.

In a system where all peers are connectable, each leecher in class i receives an equal share b_{ji} of the total bandwidth B_{ji}, that class j allocates to class i, i.e.:

$$b_{ji} = \frac{B_{ji}}{x_i}. \tag{2.2}$$

In a system where some peers are unconnectable, B_{ji} is equal to the sum of two components, B^u_{ji} and B^c_{ji}, from the unconnectable and the connectable peers respectively:

$$B_{ji} = B^u_{ji} + B^c_{ji}. \tag{2.3}$$

Assuming that each peer (connectable and unconnectable) offers the same amount of bandwidth to the system, we have

$$B^u_{ji} := \alpha_j B_{ji}, \tag{2.4}$$

$$B^c_{ji} := (1 - \alpha_j) B_{ji}, \tag{2.5}$$

where α_j is the fraction of unconnectable peers in class j.

Let δ_{ji} be the fraction of B^c_{ji} that is allocated to the group of unconnectable leechers in class i (Fig. 2.2). Since the bandwidth of the connectable peers B^c_{ji} is evenly distributed among all leechers in class i, it follows that $\delta_{ji} = \alpha_i$. Adversely, the bandwidth of the unconnectable peers B^u_{ji} is distributed among connectable leechers only.

The average download speed of an unconnectable leecher in class i, due to the contribution of peers in class j, is then given by:

$$b^u_{ji} = \frac{B^c_{ji}}{x_i} = (1 - \alpha_j) \frac{B_{ji}}{x_i}. \tag{2.6}$$

Likewise, the average download speed of a connectable leecher in class i, due to the contribution of peers in class j, is given by:

$$b_{ji}^c = \frac{B_{ji}^c}{x_i} + \frac{B_{ji}^u}{(1-\alpha_i)x_i} = \left((1-\alpha_j) + \frac{\alpha_j}{1-\alpha_i}\right)\frac{B_{ji}}{x_i}. \tag{2.7}$$

Once the quantities b_{ji}^u and b_{ji}^c are known for all classes, the average download speeds d_i^u and d_i^c, of an unconnectable leecher and a connectable leecher in each class i, can be calculated as follows:

$$d_i^u = \sum_{j=1}^{n} b_{ji}^u, \tag{2.8}$$

$$d_i^c = \sum_{j=1}^{n} b_{ji}^c. \tag{2.9}$$

2.5 Analysis

It is clear from Eqs. (2.6) and (2.7) that connectability influences peers performance and, more specifically, the download speed of unconnectable leechers is lower than that of connectable leechers. Furthermore, we can make the following observations:

1. **As the fraction of unconnectable peers grows, the difference in performance between unconnectable and connectable leechers increases hyperbolically.**

   The difference between the average download speeds of a connectable leecher and an unconnectable leecher can be expressed as follows:

$$\Delta_{ji} := b_{ji}^c - b_{ji}^u = \frac{\alpha_j}{1-\alpha_i}\frac{B_{ji}}{x_i}. \tag{2.10}$$

   To analyze the trend of Δ_{ji} as a function of the fractions of unconnectable peers in class j and class i, we distinguish between two cases:

   (i) $i \neq j$: when B_{ji} is the bandwidth going from a class j to another class i in the system, Δ_{ji} grows hyperbolically with α_i and linearly with α_j, as it is visualized in Figs. 2.3(a) and 2.3(b).

   (ii) $i = j$: when B_{ji} is the bandwidth that a class i provides to itself, Δ_{ji} grows hyperbolically with α_i (Fig. 2.4).

   Hence, it follows that, as the fraction of unconnectable peers in a given class i grows, the difference in performance between connectable and unconnectable leechers in

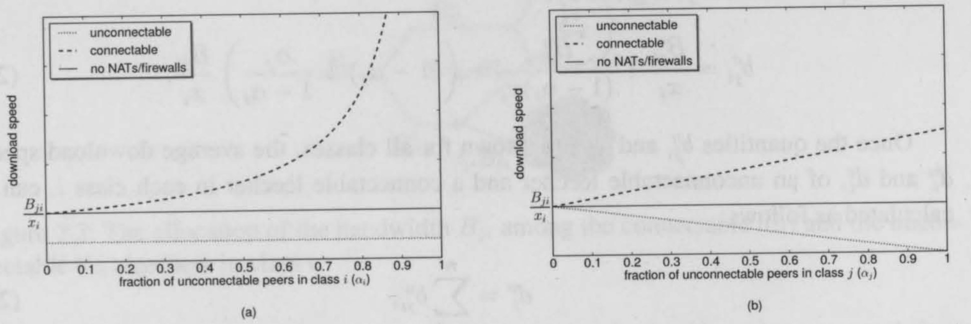


Figure 2.3: The average download speed of a connectable leecher and an unconnectable leecher in class i due to the contribution of peers in class j ($i \neq j$) (a) in relation to α_i, and (b) in relation to α_j.

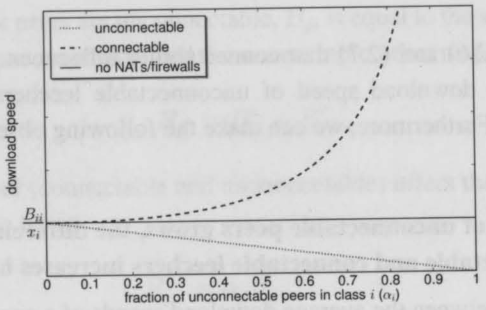


Figure 2.4: The download speed of leechers in class i, due to the contribution of peers in class i itself, in relation to α_i.

that class undergoes hyperbolic growth.

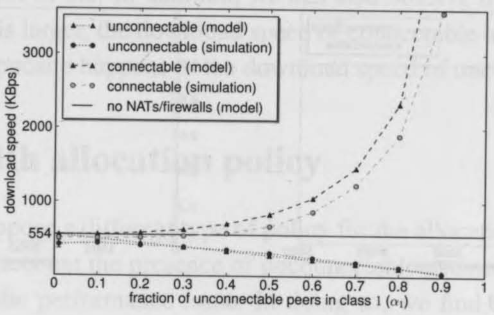


Figure 2.5: The download speed of leechers in class 1 in relation to α_1, as resulting from the simulations and the model predictions.

2. **Connectable leechers actually benefit from the presence of unconnectable peers.** Intuitively, one might think that the presence of unconnectable peers in a system would degrade, on average, *every* peer's performance. However, our analysis shows that their presence is beneficial to some peers in the system, namely the connectable peers.

   If we define G_{ji} as the difference between the speed achieved by a connectable leecher in a system with NATs/firewalls and that achieved by a connectable leecher in a system without NATs/firewalls:

$$G_{ji} := b_{ji}^c - b_{ji}, \tag{2.11}$$

   from Eqs. (2.2) and (2.7), we have that:

$$G_{ji} = \left(\frac{\alpha_j}{1-\alpha_i} - \alpha_j \right) \frac{B_{ji}}{x_i} = \frac{\alpha_i \alpha_j}{1-\alpha_i} \frac{B_{ji}}{x_i} \geq 0. \tag{2.12}$$

   Hence, the download speed of connectable leechers in a system with unconnectable peers is higher than in a system without unconnectable peers, as can be observed in Figs. 2.3(a), 2.3(b) and Fig. 2.4.

2.6 Validation in BitTorrent

To assess the validity of our approach, we have evaluated our model employing a discrete-event simulator that accurately emulates the behavior of BitTorrent at the level of piece transfers. More specifically, we have performed various simulations of simple BitTorrent swarms with two classes (class 1 and class 2) and number of peers ranging from 100 to 600. For the analytical analysis, we have computed the bandwidths B_{ji} according to the

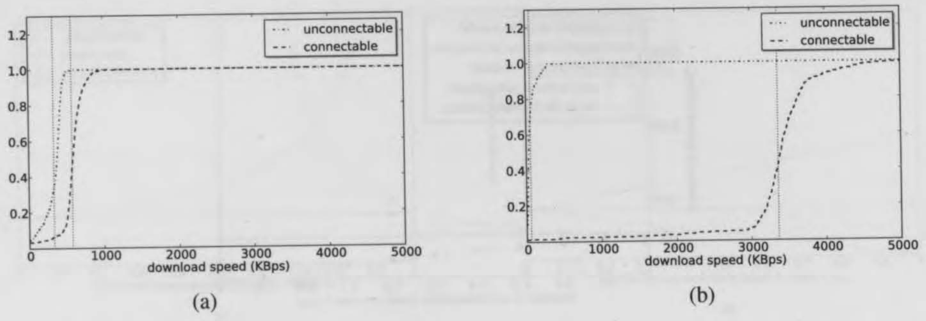


Figure 2.6: The download speed distribution (CDF) for both connectable and unconnectable leechers in class 1 from a simulation run when (a) $\alpha_1 = 0.4$ and (b) $\alpha_1 = 0.9$.

BitTorrent model proposed by Meulpolder *et al.* [44], and then derived the average download speeds of an unconnectable peer and a connectable peer as described in Eqs. (2.8) and (2.9). Fig. 2.5 plots the simulation results against the model predictions for a swarm consisting of 550 peers. Class 1 has 500 peers with upload capacity μ_1 = 512KB/s, and class 2 has 50 peers with upload capacity μ_1 = 1024KB/s. In both classes, seeders represent 10% of the peers. The average download speed of peers in class 1, per type (connectable or unconnectable), is computed during the steady state. For each value of α_1, we repeated the simulation 10 times, and calculated the mean for the speeds over all runs. The confidence intervals (computed with confidence level 95%) for the mean values of the speeds for each simulation run are also plotted, but they are hardly visible due to their small size. Other simulations performed with different setups (e.g. different swarm size and peers capacities) gave similar results.

We can notice that, although the trends described by our model still hold, the predictions are somehow a little "pessimistic", since the performance gap between unconnectable and connectable leechers, as resulting from the simulations, is often smaller than expected. This might depend on the fact that, as α_1 grows, unconnectable leechers stay longer in the system, becoming a more reliable source for pieces in respect to connectable leechers, joining and leaving faster. As a consequence, a connectable leecher will, on average, interact with unconnectable leechers with a probability higher than α_1, which determines a reduction in the performance gap between the two types of peers.

Furthermore, in our model we made the assumption that, at any moment, all the available upload bandwidth can be consumed by the peers, which is not the case in real systems (for instance, the upload bandwidth of the newly joined peers cannot be used by anyone until they have completed the download of their first piece).

Figure 2.6 shows the cumulative distribution of peers' speeds in a typical simulation run, for different values of α_1. The dotted vertical lines represent the mean values for the speeds. We can observe that the gap between the speeds of the two types of peers increases when α_1 goes from 0.4 to 0.9. In addition, we can also observe that, when the fraction of unconnectable peers is larger, the download speed of connectable leechers exhibits a higher variance, while the opposite happens to the download speed of unconnectable leechers.

2.7 Bandwidth allocation policy

In this section, we propose a different type of policy for the allocation of the system's bandwidth that takes into account the presence of unconnectable peers, as well as their fraction, in order to mitigate the performance issue. In doing so, we find that there is an intrinsic limitation in the speed improvement that unconnectable leechers can achieve, regardless of the bandwidth allocation policy employed.

2.7.1 Preferential uploading

Bandwidth allocation policies can be designed to achieve two distinct and often conflicting goals: a) maximizing the performance of individual peers, and b) maximizing social welfare (the performance across all peers) [57]. It has been argued that the value judgments of system designers determine which goal the system is designed to achieve [54].

One option is to give priority to goal a) and, for instance, leave the system as it is, with connectable leechers downloading faster. This strategy is also justified by the fact that, being them globally reachable, connectable peers could potentially contribute more in distributing the shared file. However, a number of studies (see, for instance, [35] and [41]) show that the peers downloading at high speed are also those leaving the system sooner, thus providing only little contribution in return.

Another option is to pursue goal b) and try to maximize social welfare, rather than the performance of individual peers, so that an overall satisfactory quality of service is reached in the system. In VoD for example, peers are watching the video file while still in the process of downloading it. Hence, there is no need for a connectable leecher to download much faster than an unconnectable leecher (i.e. at a download rate that considerably exceeds the video playback rate). Instead, it is necessary to provide all peers with a sufficient download speed that allows them to enjoy the experience of the video without any stall times.

Here we opt to improve social welfare and therefore we seek a strategy for allocating peers bandwidth that (i) minimizes the performance gap between connectable and unconnectable leechers and (ii) prevents the opposite situation to be created (i.e connectable leechers downloading slower than unconnectable leechers). This can be achieved by providing an identical average performance to both types of peers. More precisely, the bandwidth in the system can be allocated such that, for any class i, the following holds:

$$b^u_{ji} = b^c_{ji} = b_{ji}, \text{ for } j = 1, 2, ..., n \tag{2.13}$$

For Eq. (2.13) to hold, we need a different policy for the allocation of (connectable) peers bandwidth, i.e. a different value for the allocation factor δ_{ji}.

However, in trying to allocate the bandwidth B^c_{ji} differently, we came across an important result, which is outlined in the following theorem.

Theorem. *It is impossible for all leechers in class i to obtain an equal bandwidth contribution from the peers in class j if $\alpha_i + \alpha_j > 1$.*

Proof. In order to achieve Eq. (2.13), the bandwidth going to the group of unconnectable leechers should be $\alpha_i B_{ji}$. From Fig. 2.2 and Eq. (2.5), this bandwdith equals to:

$$\delta_{ji} B^c_{ji} = \delta_{ji}(1 - \alpha_j) B_{ji}. \tag{2.14}$$

Therefore:

$$\delta_{ji} = \frac{\alpha_i}{1 - \alpha_j}. \tag{2.15}$$

Since δ_{ji} represents the fraction of B^c_{ji} which is allocated to the unconnectable leechers, it must hold that $\delta_{ji} \leq 1$. This is only true when:

$$\alpha_i + \alpha_j \leq 1. \tag{2.16}$$

When this is not the case, *it is impossible to achieve the equality* in Eq. (2.13) and there is no way from preventing unconnectable leechers from having a worse average performance than connectable leechers. □

In the particular case in which the fraction of unconnectable peers is the same in all n classes ($\alpha_i = \alpha$ for $i = 1, 2, ..., n$), Eq. (2.16) becomes:

$$\alpha \leq 0.5. \tag{2.17}$$

Interestingly, this is the same upper bound as the one found by Mol *et al.* [47] for the unconnectable peers to have a fair sharing ratio. We believe that ours and their observations are a consequence of the same phenomenon. When the number of unconnectable peers exceeds that of the connectable peers, an unbalanced situation is created with a twofold implication. On one hand, the bandwidth provided by connectable peers to unconnectable peers is not sufficient to sustain them (thus causing their performance to drop). On the other hand, the unconnectable peers are too many to upload, to a minority of connectable peers, as much as they need to download. This results in a lower sharing ratio achieved by the unconnectable peers.

Table 2.2: Ratios b^c_{ii}/b^u_{ii} for the standard uploading policy vs the preferential uploading policy for different values of α_i.

α_i	standard	$\beta_i = 0.5$	$\beta_i = 1$	$\beta_i = 4$	$\beta_i = 16$	all peers
0.3	1.61	1.36	1.25	1.09	1.03	1.00
0.4	2.11	1.61	1.42	1.14	1.04	1.00
0.5	3.00	2.00	1.67	1.22	1.06	1.00
0.6	4.75	3.61	3.19	2.58	2.34	2.25
0.7	8.78	7.39	6.82	5.94	5.58	5.44

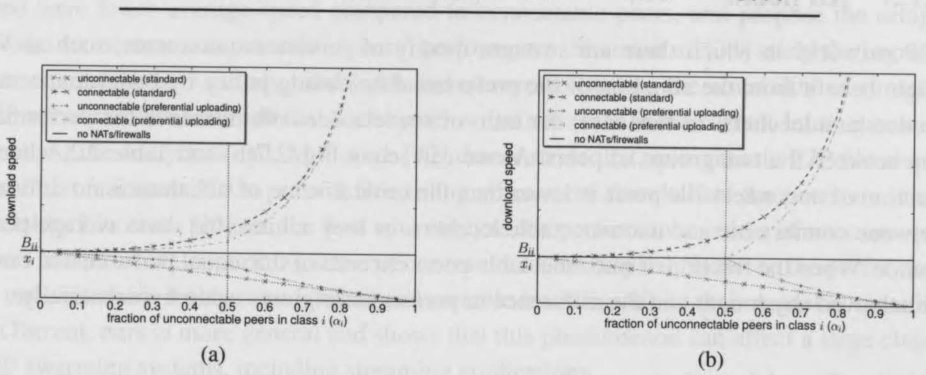


Figure 2.7: The download speed of leechers in class i, due to the contribution of peers in class i itself, when (a) only seeders use the new policy ($\beta_i = 1$) and when (b) both leechers and seeders do.

In the next two sections we analyze the performance of the two groups of peers (unconnectable and connectable) when the preferential uploading policy is implemented by the seeders only and by all peers respectively.

2.7.2 Seeders only

In relatively healthy networks (for instance where there is a high fraction of seeders, or a low fraction of unconnectable peers, or no stringent quality of service requirements), a reasonable welfare for all peers can be reached even by having only the (connectable) seeders employ the new bandwidth allocation policy. Fig. 2.7(a) shows the average speeds of leechers in class i when the new policy is used, for a scenario in which the number of seeders is equal to that of leechers. Table 2.2 reports the ratios b^c_{ii}/b^u_{ii} between the speed of connectable and unconnectable peers for different values of α_i when different policies are used. The left-most column shows the results when the standard uploading policy is used, while the next four columns show the results when the preferential uploading is used by seeders only (for different seeders/leechers ratios β_i). Finally, the right-most column shows the results when the preferential uploading is used by all peers. For the "seeders only" case, we can notice that, the lower the seeders/leechers ratio, the closer the preferential uploading policy is to the standard uploading policy. On the other hand, the larger the seeders/leechers ratio, the lower the gap between the average speed of an unconnectable and a connectable leecher. Also we observe that, when the fraction of unconnectable peers exceeds 0.5, the two average speeds tend to diverge.

2.7.3 All peers

P2P networks in which there are stronger quality of service requirements, such as VoD, might benefit from the adoption of the preferential uploading policy by both (connectable) seeders and leechers. In this case, the ratio of seeders does not influence the performance gap between the two groups of peers. As we can see in Fig. 2.7(b) and Table 2.2, when the fraction of unconnectable peers is lower than the critical value of 0.5, there is no difference between connectable and unconnectable leechers, as they achieve the same average performance. When the fraction of unconnectable peers exceeds of 0.5, equal performance can not be achieved any longer and the difference in performance grows again hyperbolically.

2.7.4 Considerations

Connectable peers can estimate the fraction of unconnectable peers in the system by making a connection back to the peers that connect to them. In this way, they can adapt their own allocation factor δ_{ji} depending on both the fraction of unconnectable peers and the specific bandwidth allocation policy.

However, we want to point out that it is necessary to be extremely careful in the design of a system where peers use a preferential uploading policy. Even though the policy has the specific purpose of restoring the performance gap between connectable and unconnectable leechers, the fact that peers actually *prefer* uploading to unconnectable leechers can lead to the opposite situation, i.e. connectable leechers downloading slower than unconnectable leechers. This can happen, for example, if connectable peers overestimate the fraction of unconnectable peers. Furthermore, it has to be noticed that connectable peers only get connected to the specific unconnectable peers that initiate the connection to them. If connectable leechers cannot get from these peers the needed pieces, they might experience a low quality of service.

2.8 Related work

Several analytical studies have been proposed to model various aspects of P2P systems. Ge *et al.* [66] analyze performance and scalability of a generic P2P system and suggest that it can tolerate a significant amount of free-riders. Qiu *et al.* [53] use a fluid model to study the performance of BitTorrent when peers have homogeneous capacities and Meulpolder *et al.* [44] extend this work to incorporate the dynamics of peers having different bandwidths. Guo *et al.* [41] study the performance of BitTorrent through modeling and trace analysis and they discover some fairness issues related to fast and slow peers. However, none of these studies analyze the effects of firewalls and NATs in P2P systems.

So far, only little research has been conducted on the topic. Skevik *et al.* [35] examine data collected through a BitTorrent crawler, finding evidence that unconnectable peers indeed have lower average speed compared to connectable peers, and propose the setup of proxies to increase unconnectable peers' performance. Shami at al. [34] study the impact of peer characteristics (namely bandwidth and connectability) on the scalability of streaming P2P networks and conclude that such systems can not scale in the current Internet environment. At the analytical end, Mol *et al.* [47] demonstrate that, in a generic P2P file-sharing system, it is impossible to prevent free-riding when more than half of the peers are unconnectable. Liu *et al.* [24] are the first who analyze the performance issues of unconnectable peers in BitTorrent, confirming some of our results on the disparity between the performance of unconnectable leechers and that of connectable leechers. While their model is specific to BitTorrent, ours is more general and shows that this phenomenon can affect a large class of P2P swarming systems, including streaming applications.

2.9 Conclusion

In this chapter, we have presented a mathematical model for a generic P2P swarming system in which a certain fraction of peers are located behind a firewall or NAT. We have analytically shown that being unconnectable lowers the performance of a peer to the advantage of those that are connectable, and this difference increases hyperbolically as the fraction of unconnectable peers increases. More importantly, we have shown that, whatever policy for the allocation of connectable peers bandwidth is used to reduce this gap, it will still be impossible to provide equal performance to unconnectable and connectable leechers if more than a certain fraction of peers are unconnectable.

can tolerate a significant amount of free-riders. Qiu et al. [53] use a fluid model to study the performance of BitTorrent when peers have homogeneous capacities, and Meulpolder et al. [44] extend this work to incorporate the dynamics of peers having different bandwidths. Guo et al. [41] study the performance of BitTorrent through modeling and trace analysis and they discover some fairness issues related to fast and slow peers. However, none of these studies analyze the effects of firewalls and NATs in P2P systems.

So far, only little research has been conducted on the topic. Skevik et al. [35] examine data collected through a BitTorrent crawler, finding evidence that unconnectable peers indeed have lower average speed compared to connectable peers, and propose the setup of proxies to increase unconnectable peers' performance. Sharma et al. [28] study the impact of (peer characteristics namely bandwidth and connectability) on the scalability of streaming P2P networks and conclude that such systems can not scale in the current Internet environment. At the analytical end, Mol et al. [47] demonstrate that, in a generic P2P file-sharing system, it is impossible to prevent free-riding when more than half of the peers are unconnectable. Liu et al. [36] are the first who analyze the performance levels of unconnectable peers in BitTorrent, confirming some of our results on the disparity between the performance of unconnectable leechers and that of connectable leechers. While their model is specific to BitTorrent, ours is more general and shows that this phenomenon can affect a large class of P2P swarming systems, including streaming applications.

2.9 Conclusion

In this chapter, we have presented a mathematical model for a generic P2P swarming system in which a certain fraction of peers are located behind a firewall or NAT. We have analytically shown that being unconnectable lowers the performance of a peer to the advantage of those that are connectable, and this difference increases hyperbolically as the fraction of unconnectable peers increases. More importantly, we have shown that, whatever policy for the allocation of connectable peers bandwidth is used to reduce this gap, it will still be impossible to provide equal performance to unconnectable and connectable leechers if more than a certain fraction of peers are unconnectable.

Chapter 3

BitTorrent-based P2P approaches for VoD: a comparative study

As mentioned in the introduction of this thesis, BitTorrent was not designed for streaming and adapting it to VoD poses two conflicting goals: satisfying the fundamental QoS requirements for streaming, while still maintaining the high efficiency of the original BitTorrent protocol.

Most of the VoD designs inspired by BitTorrent have tackled the problem by revising the components of BitTorrent commonly acknowledged for its high efficiency: piece selection and peer selection. In particular, greater attention has been paid to piece selection, as the use of local rarest-first would result in long startup delays [50]. The piece selection policies for VoD proposed in literature can be broadly classified into: *window-based*, *probabilistic*, and *priority-based*. Window-based policies work by defining a sliding window, just ahead the playback position, within which pieces are downloaded, generally according to a local rarest-first rule. With probabilistic policies, pieces are downloaded according to some probability function, which normally is biased towards the first piece not yet downloaded. Finally, priority-based policies give priority to pieces which are close to being played. For what concerns peer selection, most VoD proposals [10, 20, 51, 63] maintain the default BitTorrent's policy, based on direct reciprocity. On the other hand, Mol *et al.* [33] argue that this policy is not the best fit for VoD applications, as it may be difficult for peers with a lower level of progress to reciprocate peers with a higher level of progress. To remedy that, they propose a new peer selection policy based on *indirect reciprocity*, in which peers prefer uploading to other nodes that have forwarded pieces to others at the highest rate in the past.

However, all these approaches have been evaluated under different and limited scenarios. Hence, some methods might perform better than others under a certain set of conditions and worse under another. Furthermore, it is unclear how well each of them would work in real world conditions, where, for instance, peers have heterogeneous bandwidths and may reside behind a NAT or a firewall. Similarly, it is still unknown to what extent each approach really maintains the original BitTorrent's incentives for cooperation and whether it is as secure against malicious attacks. Exposing the pros and cons of each approach can guide in selecting the most appropriate protocol to use in a given environment. Likewise, understanding the behavior of different BitTorrent-based VoD protocols will help researchers and system designers in improving current approaches and/or designing better ones.

In this chapter, we take a first step in answering these questions. We study and compare different peer selection and piece selection policies used by the BitTorrent-based VoD approaches proposed so far, under a wide range of conditions reflecting real world environments, and under different system workloads. Our analysis shows that different approaches all share some characteristics, such as that the current methods used to incentivize cooperation, based on bandwidth reciprocity, are not suitable for scenarios with heterogeneous peers or peers behind firewalls and NATs. Furthermore, we demonstrate that a trade-off exists between QoS on one hand, and resilience to freeriding and malicious attacks on the other. In particular, we find that indirect reciprocity is more robust against freeriding than direct reciprocity, as it was conjectured in [33], but it is less secure against attacks. For what concerns piece selection, the probabilistic policy generally seems to provide the best compromise between QoS and resilience to attacks and freeriding, at least in a system where all participating peers retrieve the file in streaming mode. On the other hand, when peers doing streaming coexist with peers doing traditional file transfer, the former generally experience worse QoS than they would in a system with only streaming, independently from the specific protocol adopted by the VoD nodes. On the contrary, peers doing file transfer reach faster download rates than they would if there were no streaming nodes. In particular, the worst performance for streaming nodes (and thus the best performance for file transfer ones) is achieved when peers employ the window-based policy.

To summarize, in this chapter we make the following contributions:

1. We propose a simulation based methodology which aims at putting forward a common base for comparing the performance of different BitTorrent-based P2P protocols for VoD under a wide range of conditions (Section 3.3);
2. We find out that, in general, a trade-off exists between QoS and freeriding resilience and between QoS and security – i.e. a more QoS oriented design is more susceptible to freeriding and/or malicious attacks (Sections 3.4-3.5);
3. We discover that the current approaches to incentivize cooperation result in overall bad performance when peers have heterogeneous bandwidths or are unconnectable (i.e. behind a firewall or NAT) (Sections 3.6-3.7);
4. Furthermore, we show that the coexistence of nodes doing streaming and nodes doing traditional file transfer is disadvantageous for the former and advantageous for the latter (Section 3.8);

5. Finally, we discuss the implications of our findings for future system designs and for service providers (Section 3.9).

3.1 Related work and motivation

Previous works on P2P VoD systems mostly focused on studying and improving their performance, under the assumption that every peer would donate all its upload capacity and could communicate with every other peer in the system. Early studies [19, 27, 50], for example, show that, under this assumption on the nodes, the P2P approach is potentially effective in distributing on-demand content and providing users with good QoS. On the same line of work, Yang *et al.* [65] focus on the load balancing problem among P2P nodes as well as on the efficient scheduling of piece requests in order to further enhance QoS.

However, we note that these previous efforts do not take into account some important factors that can drastically reduce the performance and scalability of P2P systems. The first of these factors is user contribution. It has been argued that peers will not contribute their resources (namely files and upload bandwidth) unless they are given an incentive to do so. A study conducted on the Gnutella P2P system [11] seems to confirm this hypothesis, as it reports that a high percentage ($> 70\%$) of users freeride (at the time, Gnutella did not provide any incentives for users to contribute). On the other hand, it has been shown that in BitTorrent, which has an embedded incentive mechanism, only 10% of the users freeride [67]. BitTorrent also makes nearly optimal use of peers' upload bandwidth [14]. Therefore, it is no surprise that it has attracted a lot of research in the past decade, and many recent works on P2P VoD have been inspired by its design [10, 20, 33, 51, 52, 63].

Another important aspect that has received little attention in P2P streaming literature is the heterogeneity of P2P nodes. Part of this heterogeneity is intrinsically related to users' Internet access means, which may result in different bandwidth capacities as well as reduced connectability of some nodes (hosts residing behind a firewall or a NAT, for example, cannot receive inbound connections, unless the firewall/NAT is configured to do so). In open systems there is also another kind of heterogeneity related to the protocol used. In the BitTorrent ecosystem, for example, it is not uncommon to have some nodes performing traditional file transfer, while some others are requesting the same file in streaming mode (nowadays, the streaming functionality is supported by many BitTorrent clients, such as BitTorrentDNA [1], μTorrent [2], and Tribler [4]).

To the best of our knowledge, there is only one previous work that recognized the importance of some of the aforementioned aspects for VoD systems [61]. In particular, the authors focus on incentives and peer unconnectability. To incentivize users to contribute, they propose the setup of a special infrastructure to keep track of individual peer contribution. For what concerns unconnectable nodes, they introduce an approach to make these peers discoverable by connectable ones. However, we will show in Section 3.7 that, already when the fraction of unconnectable nodes is a mere 30%, this is not enough to solve the problem.

The study presented here is different and complementary to these previous efforts in that it aims at understanding to what extent a P2P (and, more specifically, a BitTorrent-inspired) approach is suitable for VoD, when the aforementioned factors come into play.

3.2 Protocol policies for BitTorrent-based VoD

In this section, we give an overview of the piece selection and peer selection policies used in the BitTorrent-based VoD protocols we consider.

3.2.1 Piece selection policies

A piece selection policy determines the next piece of the video file a peer selects for download. The piece selection policies for VoD proposed in literature try to find a trade-off between in-order download (necessary for QoS) and high bartering opportunities among peers (to ensure an efficient peer bandwidth utilization). In order to do so, they are all equipped with a *sequentiality parameter*, which can be tuned to favor one aspect or the other. In this work, we consider three categories of piece selection policies: 1) window-based 2) probabilistic, and 3) priority-based, as described below.

Window-based piece selection (WIN)

Window-based solutions typically employ a sliding window within which pieces are chosen [52]. The window advances from the beginning to the end of the file according to the sequential download progress at the local peer. Normally, the window starts at the first piece not yet downloaded. Within the window, rarest-first piece selection is often applied. Naturally, a smaller window ensures close to sequential piece retrieval, but reduces piece diversity and, hence, bartering opportunities among peers. On the other hand, a larger window allows for higher bartering opportunities among peers but increases both their startup delays and their chance of not being able to download the pieces before their playback is due. The size (in pieces) of the window represents the sequentiality parameter of this piece selection policy and we denote it with w.

Probabilistic piece selection (PROB)

In probabilistic piece selection, pieces are chosen in relation to some probability distribution, generally with a bias towards the first pieces not yet downloaded. In this work, we consider the policy proposed in [48], where a Zipf probability distribution is used. Specifically, the probability that a peer p selects to download a piece k is proportional to $(k+1-k_0)^{-\theta}$ where k_0 is the index of the first piece peer p has not yet downloaded. Similar to the window size w

for the window-based policy, θ represents the sequentiality parameter of this piece selection policy and can be tuned to provide close to sequential piece retrieval, but low bartering opportunities among peers (large θ), and vice versa (small θ).

Priority-based piece selection (PRIO)

With priority-based approaches, priority is given to pieces which are close to be played. We use the method presented in [33], where a peer, whose current playback position is p, will request a piece i on the first match in the following list of sets of pieces (known as *priority sets*):

- high priority: $p \leq i < p + h$: in-order piece selection if the local peer has already started playback, rarest first otherwise;
- mid priority: $p + h \leq i < p + 5h$: with rarest first piece selection;
- low priority: $p + 5h \leq i$: with rarest first piece selection.

In these definitions, h denotes the size (in pieces) of the high priority set and represents the sequentiality parameter of this piece selection policy. Similarly to the previous policies, the parameter h can be tuned to give more emphasis to sequential piece retrieval (large h) or high bartering opportunities among peers (small h).

3.2.2 Peer selection policies

A peer selection policy determines how a node selects another node to upload data to. In BitTorrent-based systems, the peer selection policy has the task of incentivizing peer cooperation, and therefore it is usually designed to favor good uploaders. In this work, we will consider two policies, based on *direct reciprocity* and *indirect reciprocity*, respectively.

Direct reciprocity

When a node uses peer selection based on direct reciprocity, it will upload to other nodes that have recently uploaded to it at the highest rates. As mentioned in the Introduction, this is the standard peer selection policy employed in BitTorrent.

Direct reciprocity is easy to implement: each peer takes its decisions only based on the information locally available (i.e. the measured upload rates of other nodes) and no long-term memory is required (normally peers re-evaluate the upload rates of other nodes every 10 seconds). Many studies show that it works successfully in the context of traditional file transfer [8, 31, 67]. However, it has been argued that in P2P VoD systems, due to the somewhat in-order download progress of peers, it is more difficult for peers with lower degrees of progress to reciprocate peers with higher degrees of progress [33]. Consequently,

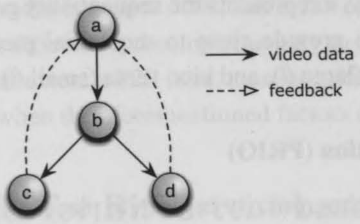


Figure 3.1: Data flow and feedback flow for indirect reciprocity.

it would be more difficult for a peer to detect whether another peer is freeriding or simply has no interesting piece to barter for.

Indirect reciprocity

When a node uses peer selection based on indirect reciprocity, it will upload to other nodes that have recently *forwarded* data to others at the highest rates. In this work, we use the method introduced in the *give-to-get* (G2G) protocol [33]. In G2G, a peer a discovers the forwarding rate of a child node b by periodically asking its grandchildren about the pieces received from b. Note that the child b is not asked directly as it could make false claims. This process is depicted in Figure 3.1.

Indirect reciprocity is intuitively more suitable to a VoD scenario, since it does not necessarily require nodes with lower levels of progress to reciprocate nodes with higher levels of progress. However, since each peer needs to gather information from other nodes, it is more costly to implement (especially in presence of NATs and firewalls) and is potentially more vulnerable to various types of attacks, as we will show in Section 3.5.

3.3 Methodology

In this section, we describe the approach we use to compare and study different VoD protocols. First, we introduce some of the terms we will employ in our analysis, as well as the model of the system we consider. Then, we illustrate in detail the experimental setup for our analysis and how we tuned the sequentiality parameters of the piece selection policy considered.

3.3.1 Definitions

In this subsection, we introduce the most important definitions which are used throughout the whole chapter. The first three definitions are related to user behavior, which are followed by definitions closer to the system level.

A peer whose upload capacity is set to 0 is called *freerider*. A freerider follows the specific protocol in all aspects. Freeriders generally correspond to users who configure their network access in order not to share their upload capacity. The behavior of freeriders is termed as *freeriding*. A *malicious peer* does not follow (some of) the rules of the specific protocol. Finally, an *honest peer* follows all the rules of the specific protocol and has upload capacity larger than 0. Honest peers do not try to manipulate the protocol and, whenever possible, share all their upload capacity.

An *unconnectable peer* does not possess a globally reachable address. Usually, this happens when the peer resides behind a NAT or a Firewall that is not configured to accept incoming connections. A *missed piece* is a piece whose download cannot be completed before the time it is due to be played. A node p is said to be *interested* in another node q when q possesses a piece that p does not and has not missed. Similarly, q is said to be *interesting* for p.

3.3.2 System model during steady state

We consider a system where the participating peers can retrieve the stream of a particular video file, of playback rate R and size F, which is split into n pieces of identical size. The system is assumed to be in steady state[1], with peers joining at a constant rate λ and leaving as soon as their download is complete. For the purpose of the analysis, we also assume that the download capacity of peers is not a bottleneck and can be considered to be infinite. The average upload capacity of peers in the system is denoted by μ. In addition to the peers, the system contains a number of servers (or *seeders*) which contribute an aggregate upload capacity of U_s.

Given the above notation, it is easy to see that the expected download speed u of a peer in steady state is

$$u = \frac{U_s}{N} + \mu, \tag{3.1}$$

where N is the number of peers in the system. From Little's Law it follows that

$$N = \lambda \frac{F}{u}, \tag{3.2}$$

where F/u is the expected download time, and thus N increases as the arrival rate λ of peers increases.

Given the above observation, it is clear that, if the server bandwidth U_s is constant, the download speed u of peers in steady state decreases as their arrival rate λ increases. This would lead to an unfair comparison among scenarios characterized by different arrival

[1] a system is said to be in steady state when, although peers might join and leave, the total number of peers remains constant over time.

rates, with peers in scenarios with small arrival rates having faster download speeds and, consequently, better QoS, than peers in scenarios with higher arrival rates. To make the comparison fair, the server bandwidth needs to be dimensioned to the peer arrival rate such that, in all scenarios, peers reach a certain desired steady state download speed denoted with u^*:

$$U_s = \left(1 - \frac{\mu}{u^*}\right) \lambda F,$$

where this formula is obtained by combining Eqs. (3.1) and (3.2). For the system to be able to provide a good QoS, it is necessary to have $u^* \geq R$. In particular, the closer u^* is to R, the lower the server bandwidth. If we express u^* as a function of R as follows

$$u^* = \gamma R, \text{ with } \gamma \geq 1$$

then the required server bandwidth U_s can be calculated as

$$U_s = \left(1 - \frac{\mu}{\gamma R}\right) \lambda F.$$

In this way, once the characteristics of the system are known, the service provider only needs to set a value for the γ parameter that suits the needs of the system.

3.3.3 Experimental setup

We compare the performance of different schemes of peer and piece selection policies by means of simulations.

For this purpose, we have extended the BitTorrent simulator designed by Microsoft Research [14], in order to support VoD. This is a very detailed simulator, where all the elements of a BitTorrent system are modelled with great accuracy, from the creation of the overlay to the exchange of piece between peers. The overlay is created by means of a tracker module operating in the classical BitTorrent fashion outlined in the introduction to this thesis: when it is contacted, the tracker returns a list of random nodes to the requesting peer. Due to its great level of accuracy in reproducing the behavior of BitTorrent systems, this simulator has been widely used, also for simulating BitTorrent-like VoD protocols [37, 63, 65]. Our extension supports all the piece selection policies presented in Section 3.2.1 and allows for the system to either adopt direct reciprocity or indirect reciprocity as peer selection policies. We have made our extension available at `http://www.pds.ewi.tudelft.nl/dacunto/research` for those interested in continuing this research.

The settings for our simulations are shown in Table 3.1.We consider a case where the service provider has set up one seeder (or server) which is always online. For the computation of the server bandwidth U_s, we use $\gamma = 1.3$, in order to compensate for fluctuations in

Table 3.1: Simulation Settings

Parameter	Value
Video playback rate R	800 kb/s
Video length L	50 min
Simulation time	between 250 and 750 min ($5L$ and $15L$)
Piece size P_S	256 kB
Initial buffer B	20 pieces (PROB) / w (WIN) / h (PRIO)
Upload rate μ	1000 kb/s ($1.25R$)
# upload slots	4

download speeds and to account for the fact that downloaders are not always able to upload at their full capacities. To decide when playback can safely commence, we use the strategy introduced in [20]. Specifically, a peer will start playback only when it has obtained all the pieces in an initial buffer of size B and its current *sequential progress*[2] is such that, if maintained, the download of the file will be completed before playback ends. The buffer size B is equal to w or h for the window-based and the priority-based piece selection policies, respectively, and is set to 20 pieces for the probabilistic one. The values for w and h used in the simulations have been selected according to the method presented in the next subsection.

The performance of any combination of peer and piece selection policies is analyzed using the following two metrics:

- *continuity index*, defined as the ratio of pieces received before their deadline over the total number of pieces;
- *startup delay*, defined as the time a user has to wait before playback starts.

Each simulation run is executed 15 times, and then average values and confidence intervals (with confidence level of 95%) for the above metrics are computed. The lower the arrival rate, the longer it takes for the system to reach the steady state. Therefore, simulations times are longer for lower arrival rates. Furthermore, for the results we have only considered the peers who have joined after the first half of the simulation time and before the last L minutes from the simulation end.

3.3.4 Tuning the sequentiality parameters for the comparison

Recall from Section 3.2.1 that each piece selection policy for VoD is characterized by a sequentiality parameter which allows one to give more emphasis to sequential piece retrieval or to high bartering ability among peers. In order to fairly compare these policies

[2]a peer's *sequential progress* is the speed at which the index of the first piece in the file not yet downloaded grows and it represents the rate at which a continuous stream is received [50]. It should not be confused with the sequentiality parameter (defined in Section 3.2.1) that characterizes each piece selection policy.

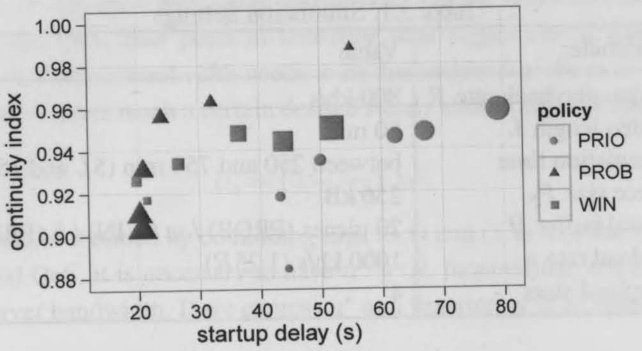


Figure 3.2: A scatter plot of startup delay vs continuity index for different piece selection policy instances. Smaller points correspond to smaller values for the sequentiality parameters of each piece selection policy. Peer arrival rate is 0.005 peer/s and the remaining simulation settings are as in Table 3.1.

against each other, they need to be "tuned" in such a way that they exhibit a similar level of sequentiality. To do so, we have performed several experiments where each policy uses a wide range of values for its sequentiality parameter. For these experiments, the fraction of freeriders was set to 10%, as this represents a typical value in BitTorrent systems [67] and peer arrival rates are as follows: $\boldsymbol{\lambda} = (\lambda_0, \lambda_1, \lambda_2, \lambda_3) = (0.005, 0.01, 0.05, 0.1)$ peers/s. The smaller values (0.005, 0.01) account for the case of not very popular videos, which generate only little load in the system, while the larger values account for the case of very popular video, which thus generate higher load.

To find a common baseline, we use the following approach: we select the sequentiality parameter for each piece selection policy such that the startup delays experienced by honest peers within each of them are similar. We first focus on the direct reciprocity case and then, based on it, we tune the sequentiality parameters for the case of indirect reciprocity.

Direct reciprocity

Given the setup introduced in Section 3.3.3, we have tested the following values for the sequentiality parameters:

- w: from 10 to 60, with step 5 (this set is denoted by W);
- θ: from 1.5 to 4, with step 0.25 (this set is denoted by Θ);
- h: 10, to 60, with step 5 (this set is denoted by H).

As we can observe from Figure 3.2, the instances that produce short startup delays generally determine low continuity indexes as well. This confirms that a trade-off indeed exists between sequentiality and bartering ability among peers (and thus between short startup delay and high continuity index).

To select our policy instances, we start from the window-based one. For this policy, increasing the sequentiality parameter w means longer startup delays but also higher continuity index (Figure 3.2 and Section 3.2.1). We select the smallest sequentiality parameter w for which at most 10% of the honest peers are experiencing a continuity index less than 0.95. This choice is motivated by Habib *et al.* [28]: the user's overall perceived video quality is considered very good when the continuity index of a stream is not less than 0.95. This selection leads to $w^* = 40$.

Then, we calculate the Euclidean distance between the startup delay values of this particular instance of the window-based policy and the startup delay values of the other two policies, and select the istances that exhibit the smallest such a distance. Specifically, having defined with $\boldsymbol{D}^{w^*} = (D_0^{w^*}, D_1^{w^*}, D_2^{w^*}, D_3^{w^*})$ the vector containing the values for the startup delays obtained with arrival rates $\boldsymbol{\lambda}$ for the window-based policy characterized by $w^* = 40$, and with $\boldsymbol{D}^\theta = (D_0^\theta, D_1^\theta, D_2^\theta, D_3^\theta)$ and $\boldsymbol{D}^h = (D_0^h, D_1^h, D_2^h, D_3^h)$ the startup delay vectors for the generic probabilistic and priority-based policies, respectively, we find the vectors $\boldsymbol{D}^{\theta^*}$ and $\boldsymbol{D}^{h^*}$ most similar to $\boldsymbol{D}^{w^*}$ as follows:

$$\boldsymbol{D}^{\theta^*} = \min_{\theta \in \Theta} \sqrt{\sum_{k=0}^{3} (D_k^{w^*} - D_k^\theta)^2}, \quad \text{and} \quad \boldsymbol{D}^{h^*} = \min_{h \in H} \sqrt{\sum_{k=0}^{3} (D_k^{w^*} - D_k^h)^2}.$$

Note that technically this means that for all parameters from the sets Θ and H we take the magnitude of the four-dimensional vectors $\boldsymbol{D}^\theta$ and $\boldsymbol{D}^h$ and choose the one which has the most similar magnitude to that of $\boldsymbol{D}^{w^*}$. Using this method, we obtain that the window-based policy characterized by $w^* = 40$ is mostly similar to the probabilistic and the priority-based policies characterized by $\theta^* = 2$ and $h^* = 25$, respectively. Figure 3.3 shows the startup delay and the continuity index for honest peers when the three policies use these selected values for their sequential parameters.

Indirect reciprocity

In the case of indirect reciprocity we have chosen, for each piece selection policy, a value for the sequentiality parameter such that the startup delay of honest peers is similar to that of the same policy in the case of direct reciprocity. To evaluate similarity, we have again used the Euclidean distance. This approach led to the following values: $w^* = 30$, $\theta^* = 2$, and $h^* = 20$, which, as we can see in Figure 3.3, exhibit equal or slightly lower startup delays

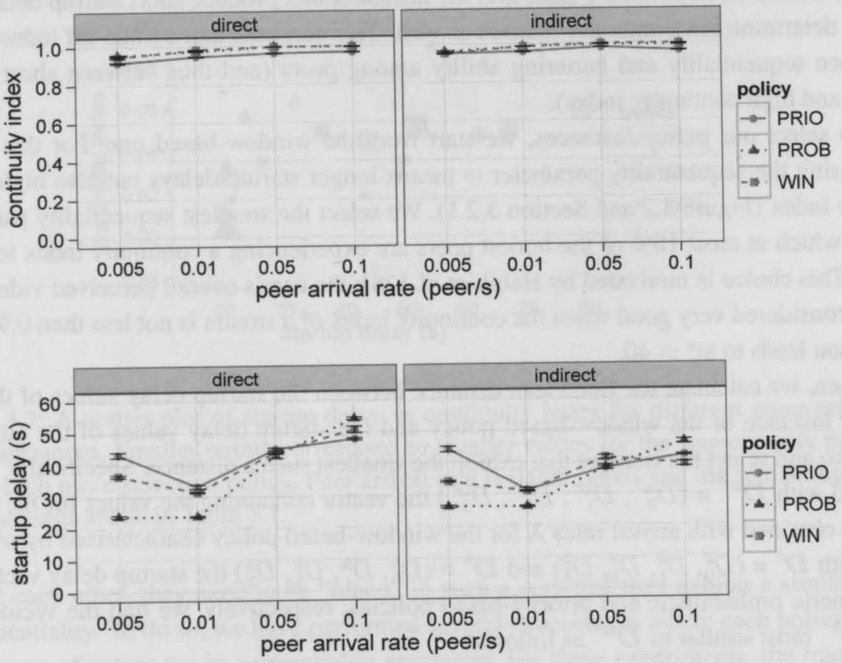


Figure 3.3: Performance of honest peers with piece selection policies using the selected sequentiality parameters (Table 3.2).

as their counderparts in the case of direct reciprocity.

The selected values for the sequentiality parameters for both indirect and direct reciprocity are summarized in Table 3.2. Unless otherwise stated, we will use these values in the remainder of this chapter.

3.4 Freeriding resilience

One of the key factors for the success of BitTorrent is its effective incentive mechanism which induces peers to contribute bandwidth while downloading. This has motivated researchers to apply the same design to the VoD case. However, to the best of our knowledge, the research community still lacks of a clear understanding of whether, once applied to VoD, this mechanism yields to the same degree of freeriding resilience as the original BitTorrent protocol. In order to verify that, in this section, we analyze the performance of each combination of piece and peer selection policy as we vary the fractions of freeriders. We will first analyze the case of direct reciprocity and then the case of indirect reciprocity.

Table 3.2: Setups for the Sequentiality Parameters

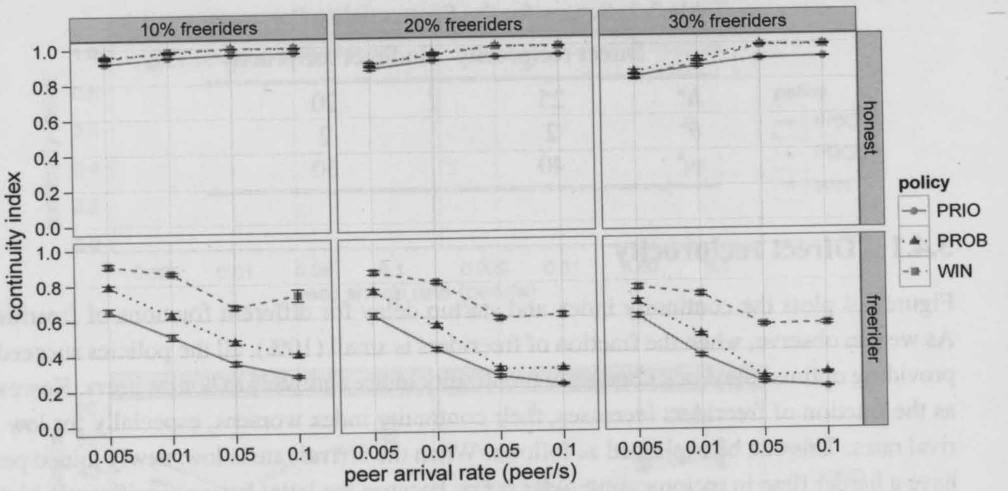
	Direct reciprocity	Indirect reciprocity
h^*	25	20
θ^*	2	2
w^*	40	30

3.4.1 Direct reciprocity

Figure 3.4 plots the continuity index and startup delay for different fractions of freeriders. As we can observe, when the fraction of freeriders is small (10%), all the policies succeed in providing optimal playback continuity (continuity index almost 1) to honest peers. However, as the fraction of freeriders increases, their continuity index worsens, especially for low arrival rates. This can be explained as follows. When the arrival rate is low, newly joined peers have a harder time in reciprocating older peers, because the latter have a significantly higher level of progress. As a consequence, it is difficult for an older peer p to distinguish between a freerider and a newly joined peer, who, although willing to reciprocate, is not doing so because it does not have any interesting piece for p. Intuitively, reducing the sequentiality of the piece selection policy will increase the bartering ability of peers, thus improving freeriding resilience. We have analyzed in more detail the relationship between sequentiality and number of bartering partners for different peer arrival rates by means of mathematical analysis, for which we refer the reader to Appendix A. Figure 3.5 demonstrates our analysis for the window-based piece selection policy; with low arrival rate even a large window size would result in only a few bartering partners.

For what concerns the specific piece selection policies, the probabilistic one performs the best, providing the highest continuity index for honest peers, and a relatively low continuity index for freeriders. The priority-based policy, on the other hand, determines the worst continuity indexes for honest peers. This is due to the fact that, different from the window-based and probabilistic ones, in the priority-based policy, the download point is relative to the current playback position, rather than the first piece not yet downloaded. Hence, there is a "less safe" distance between the download progress and the playback progress, which causes peers to miss pieces more frequently.

Turning our attention to startup delay, we observe that, while for low arrival rates the probabilistic policy provides the shortest startup delays to honest peers, for higher arrival rates the window-based policy performs the best. This is due to the fact that, with the probabilistic approach, peers tend to download also pieces which are farther away. This will increase their bartering ability when peer arrival rate is low but it will slow them down when the arrival rate is high. In fact, in the latter case, downloading pieces farther away is unnecessary, since peers have considerably more chance to meet peers with a similar level



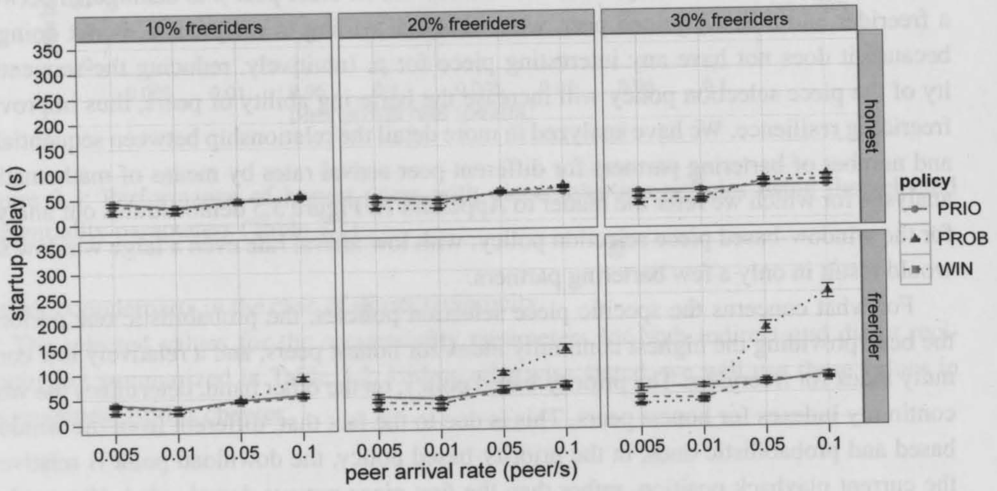


Figure 3.4: Performance of honest and freeriding peers in a system using direct reciprocity with different fractions of freeriders.

of progress. Finally, we observe that in all cases freeriders experience much longer startup delays when the probabilistic policy is used, thus reducing their incentive to freeride even further.

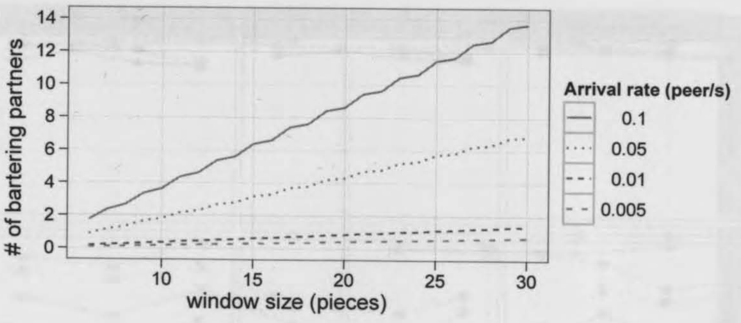


Figure 3.5: Expected number of bartering partners for the window-based piece selection policy with different window size and peer arrival rate. The parameters are taken from Table 3.1.

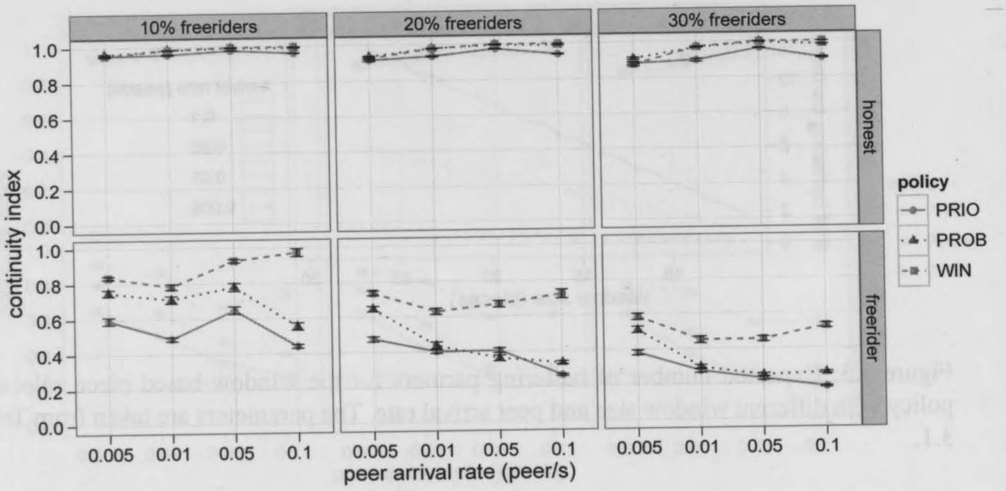
3.4.2 Indirect reciprocity

Figure 3.6 shows the simulation results when using indirect reciprocity. We observe that, in general, this policy provides better QoS to honest peers, both in terms of continuity index and startup delay, than direct reciprocity. At the same time, freeriders experience worse performance. We also observe a lower negative influence of low arrival rates on honest peers. This is due to the fact that peers are rewarded for forwarding, not for bartering. To summarize, we can say that indirect reciprocity works better for VoD than direct reciprocity, for what concerns freeriding resilience.

3.5 Sybil attack to indirect reciprocity

In this section, we focus on a potential vulnerability introduced by the indirect reciprocity-based approach, namely the Sybil attack. Recall that, in a system with a peer selection policy based on indirect reciprocity, a peer uploads to those peers that have recently forwarded data to others at the highest rates (Section 3.2.2). Therefore, a malicious peer b can game the system by first creating Sybils $c_1, ..., c_n$ which act as b's children. Whenever b downloads a piece from another peer a, each of b's children c_i immediately reports to a that b has forwarded pieces to it at a fast rate. From a's perspective, b becomes a fast uploader and hence a keeps uploading to b in the future rounds. As a result, the malicious peer b can keep downloading pieces without any incentive to actually forward or upload any data to other peers; in other words, b is a freerider. Since this is a "dominant strategy" for any peer, it results in a "tragedy of commons" if each peer adopts this strategy.

To illustrate the impact of this attack, we compare the performance for honest and malicious peers under the scenarios of freeriding and the proposed sybil attack. Figure 3.7 plots



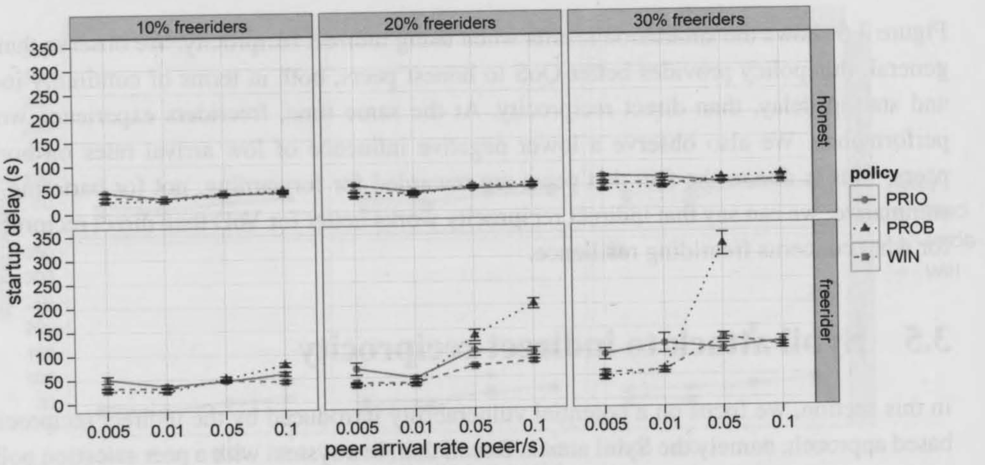


Figure 3.6: Performance of honest and freeriding peers in a system using indirect reciprocity with different fractions of freeriders.

the continuity index and startup delay as we vary the peer arrival rate. We make the following observations. First, the continuity index for malicious peers employing sybil attack strategy is (i) comparable to that of honest peers and (ii) much better than that for freeriders. Second, the startup delay for malicious peers under sybil attack is (i) lower than that for honest peers and (ii) significantly lower compared to freeriders. Hence, even without uploading any data to others, malicious peers have a similar performance to that of honest ones. This result highlights that, albeit more free-riding resilient than direct reciprocity, indirect reciprocity

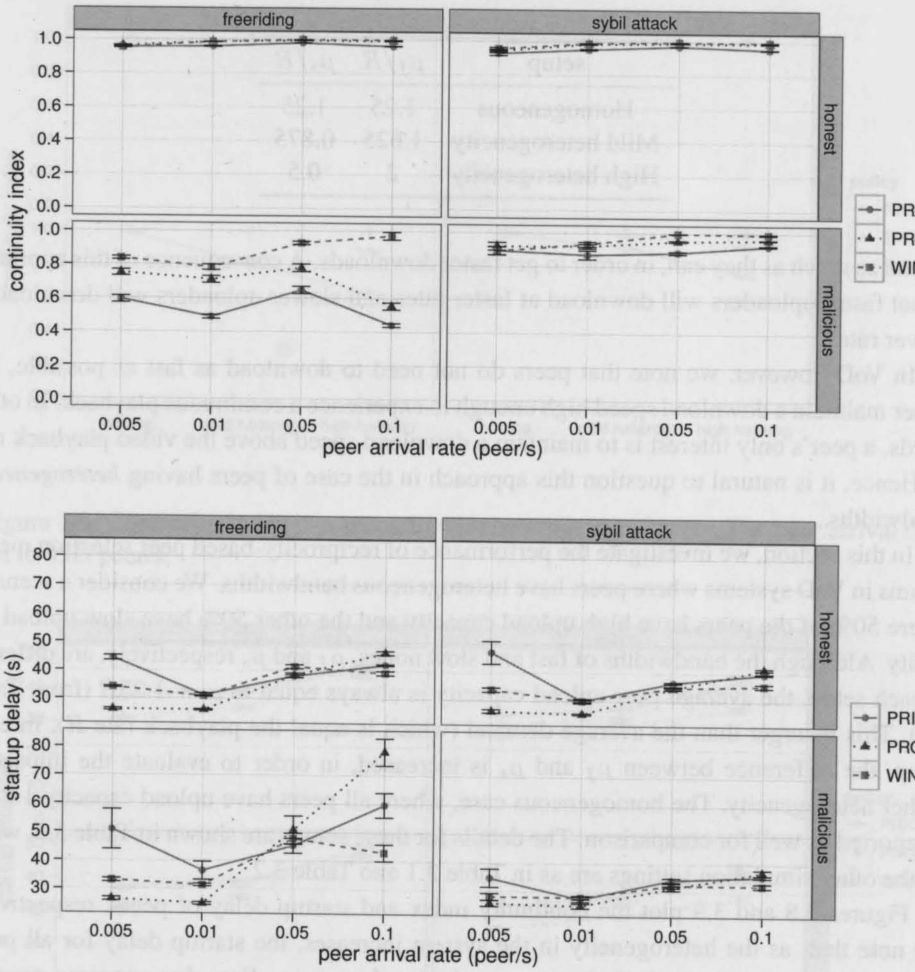


Figure 3.7: Impact of sybil attack. The fraction of malicious nodes is set to 0.1 and the peer arrival rate to 0.05 peer/s.

comes at the price of a severe security drawback.

3.6 Heterogeneous environment

Recall from Section 3.2.2 that the peer selection policies considered in this chapter are both based on bandwidth reciprocity: i.e. a peer receives (roughly) as much bandwidth as it provides to the system. In traditional file transfer, this mechanism incentivizes peers to

Table 3.3: Setups for the heterogeneous scenario.

setup	μ_f/R	μ_s/R
Homogeneous	1.25	1.25
Mild heterogeneity	1.625	0.875
High heterogeneity	2	0.5

upload as much as they can, in order to get faster downloads. A consequence of this approach is that faster uploaders will download at faster rates and slower uploaders will download at slower rates.

In VoD however, we note that peers do not need to download as fast as possible, but rather maintain a download speed high enough to experience a continuous playback. In other words, a peer's only interest is to maintain a download speed above the video playback rate R. Hence, it is natural to question this approach in the case of peers having *heterogeneous* bandwidths.

In this section, we investigate the performance of reciprocity-based peer selection mechanisms in VoD systems where peers have heterogeneous bandwidths. We consider a scenario where 50% of the peers have high upload capacity and the other 50% have slow upload capacity. Although the bandwidths of fast and slow nodes, μ_f and μ_s respectively, are different in each setup, the average peer upload capacity is always equal to $\mu = 1.25R$ (from Table 3.1). This is larger than the average demand (which is equal the playback rate R). In each setup, the difference between μ_f and μ_s is increased, in order to evaluate the impact of higher heterogeneity. The homogeneous case, where all peers have upload capacity $1.25R$, is reported as well for comparison. The details for these setups are shown in Table 3.3, while all the other simulation settings are as in Table 3.1 and Table 3.2.

Figures 3.8 and 3.9 plot the continuity index and startup delay of peers, respectively. We note that, as the heterogeneity in the system increases, the startup delay for all peers increases while continuity index worsens only for slow peers. For what concerns peer selection, indirect reciprocity performs better than direct reciprocity, with generally a higher continuity index for slow peers and a lower startup delay for all peers. Regarding piece selection policies, window-based exhibits the best continuity index and outperforms the other two also for startup delay, at least when used in combination with indirect reciprocity. The more "fair" performance of the window policy is due to the fact that peers only download pieces from a small window, ahead the first piece not yet downloaded. Hence, a fast peer cannot barter with other faster peers having lower or higher progress level, but only with those peers (fast and slow) having similar progress level to itself.

However, the fact that startup delay increases with the heterogeneity is an undesired drawback of these VoD approaches. To understand the reasons for this behavior, we looked at the peers average download rates (Figure 3.10). The average speed of peers (especially of

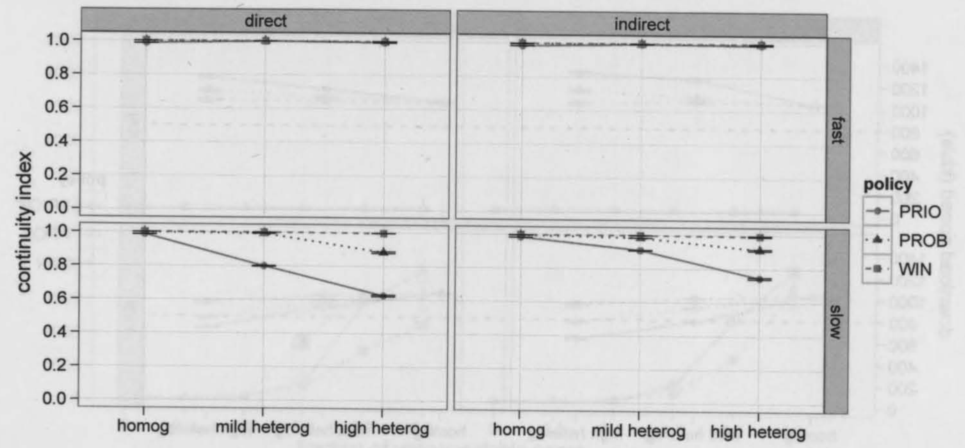


Figure 3.8: Continuity index in a scenario with heterogeneous peers. The peer arrival rate is set to 0.05 peer/s.

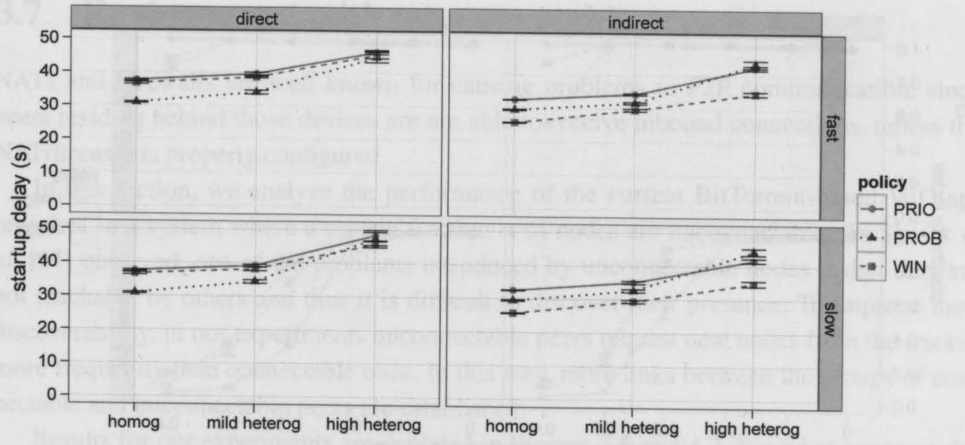


Figure 3.9: Startup delay in a scenario with heterogeneous peers. The peer arrival rate is set to 0.05 peer/s.

the fast ones) is generally far above the playback rate R (which is 800 kb/s). In particular, for fast peers, we observe that the download speed is more or less constant (or in some cases growing with the level of heterogeneity) in all setups. Therefore, the increase of startup delay in the heterogeneous scenarios can be explained with peers downloading at slower rates

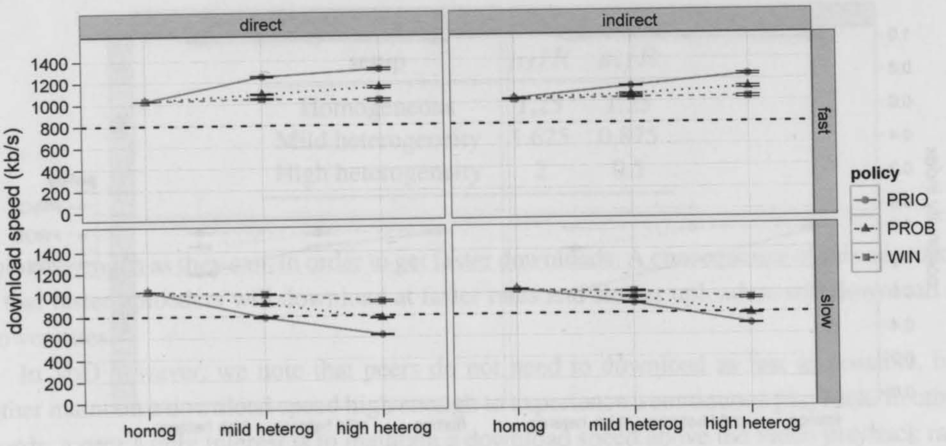


Figure 3.10: Download speed in a scenario with heterogeneous peers. The peer arrival rate is set to 0.05 peer/s.

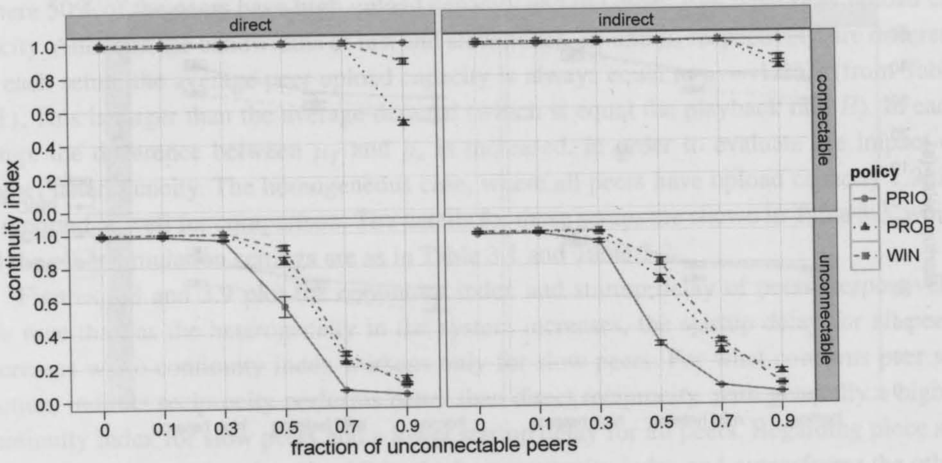


Figure 3.11: Continuity index experienced by peers when some are unconnectable. The peer arrival rate is set to 0.05 peer/s.

when at the beginning of the file, while downloading much faster when coming towards the end.

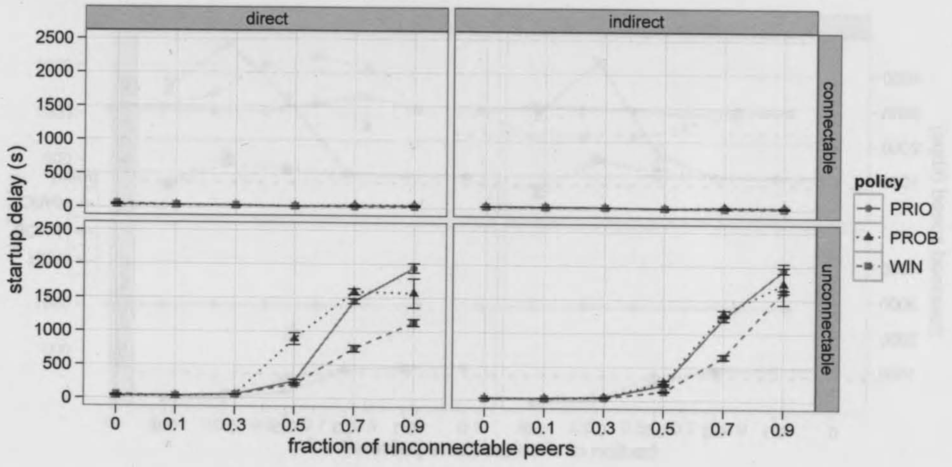


Figure 3.12: Startup delay experienced by peers when some are unconnectable. The peer arrival rate is set to 0.05 peer/s.

3.7 Environment with unconnectable peers

NATs and Firewalls are well known for causing problems to P2P communication, since peers residing behind those devices are not able to receive inbound connections, unless the NAT/firewall is properly configured.

In this section, we analyze the performance of the current BitTorrent-based VoD approaches in a system where a certain fraction α of nodes are unconnectable. As Skevik *et al.* [61] observed, one of the problems introduced by unconnectable nodes is that they are not reachable by others and thus it is difficult to discover their presence. To improve their discoverability, in our experiments unconnectable peers request new nodes from the tracker more frequently than connectable ones. In this way, more links between the groups of connectable and unconnectable peers are established.

Results for our experiments are depicted in Figures 4.6 and 4.7. For what concerns the continuity index, connectable nodes are not affected, until their fraction drops below 10%. The unconnectable nodes, on the other hand, start experiencing a bad continuity index already when their fraction reaches 50%, which has been measured to be a typical value in BitTorrent as well as BitTorrent-based VoD systems [32, 45]. Similarly, their startup delay suffers a steep increase once past the 50% threshold. This phenomenon is due to the fact that unconnectable nodes can only upload to connectable ones, while the latter can potentially upload to any other peer. Hence, when the unconnectable peers are the majority, the connectable ones receive a lot of bandwidth from them while giving only a smaller amount of bandwidth in return. This result is in accordance with the study of general P2P swarming

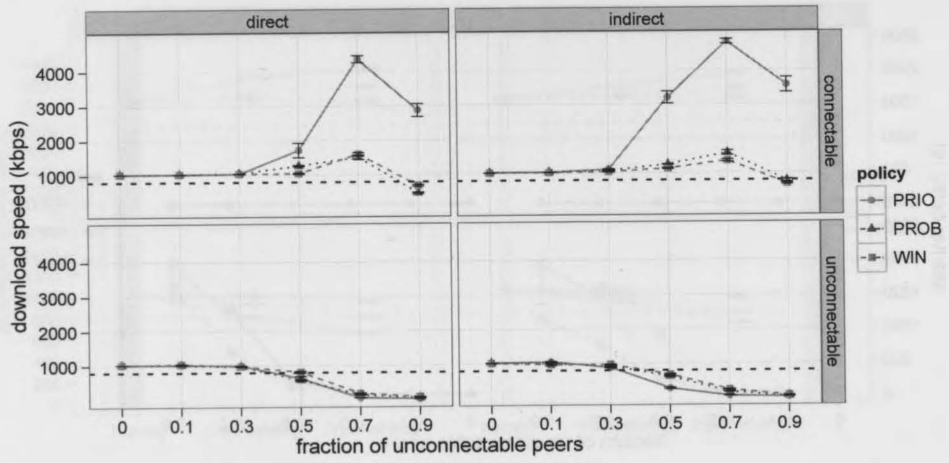


Figure 3.13: Download speed experienced by peers when some are unconnectable. The peer arrival rate is set to 0.05 peer/s.

systems presented in Chapter 2. Consequently, it is clear that the presence of unconnectable nodes severely affects the performance of P2P swarming systems, and VoD systems in particular, and must be kept into account when designing a new P2P system or protocol.

In order to explore avenues for possible improvements, we have looked again at the download speeds of peers (Figure 3.13). While the download rate of unconnectable nodes drops below the playback rate R (800 kb/s) already when their fraction reaches 50%, that of connectable nodes stays above R, and actually increases, until α = 90%. This implies that connectable nodes often get more bandwidth than necessary to maintain a continuous playback. It is intuitive that this bandwidth surplus could be used to "help" unconnectable peers, as already suggested in the previous chapter.

3.8 Coexistence with traditional file transfer

In this section, we analyze the implications of having a mixed environment, where peers doing streaming coexist with those doing file transfer using the standard BitTorrent protocol. In fact, nowadays many BitTorrent clients (e.g. BitTorrentDNA [1], μTorrent [2], and Tribler [4]) already support streaming functionalities and more clients are likely to start doing this in the future.

At a first glance, one would expect that introducing VoD nodes in a BitTorrent system where peers do traditional file transfer will negatively affect piece availability and consequently decrease the download speed of the original (non VoD) nodes. In order to verify

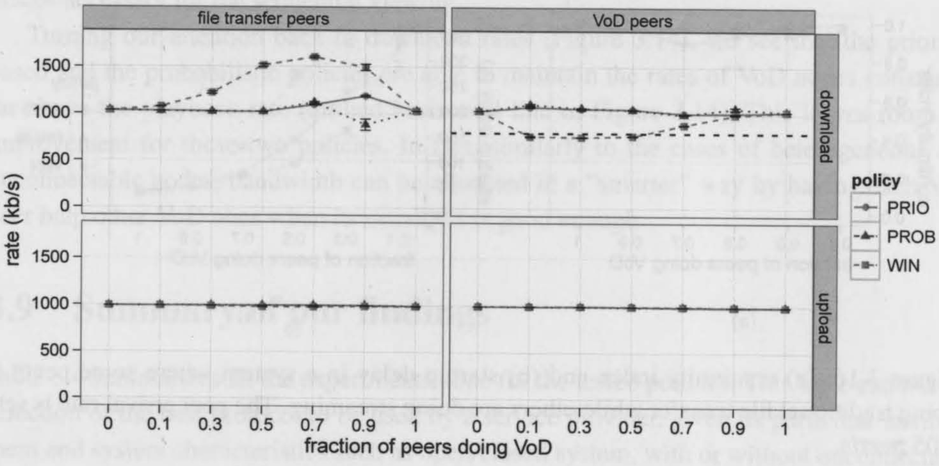


Figure 3.14: Download and upload rates of nodes doing traditional file transfer against nodes doing streaming, for different VoD piece selection policies (standard BitTorrent is used for file transfer in all cases). The dashed horizontal line in the top-right panel represents the video playback rate R. The peer arrival rate in these experiments is set to 0.05 peer/s.

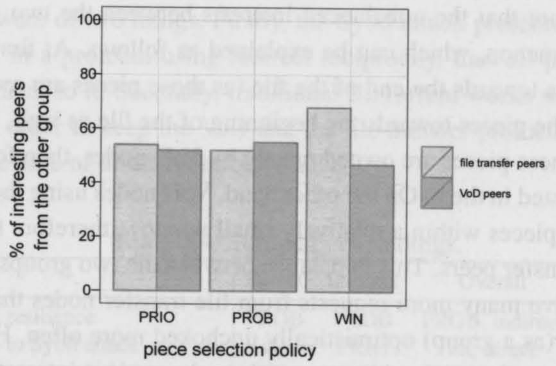


Figure 3.15: Interest in the other group of peers in a system consisting of both streaming and file transfer nodes.

whether this is really the case, we have performed experiments with increasing fractions of VoD peers. Figure 3.14 plots the download rates for both file transfer nodes and VoD nodes. As we can observe, file transfer peers actually *benefit* from the presence of VoD ones: the more the latter, the better for the former. The largest advantage is obtained when the VoD nodes employ the window-based piece selection policy. In order to explain this result, we

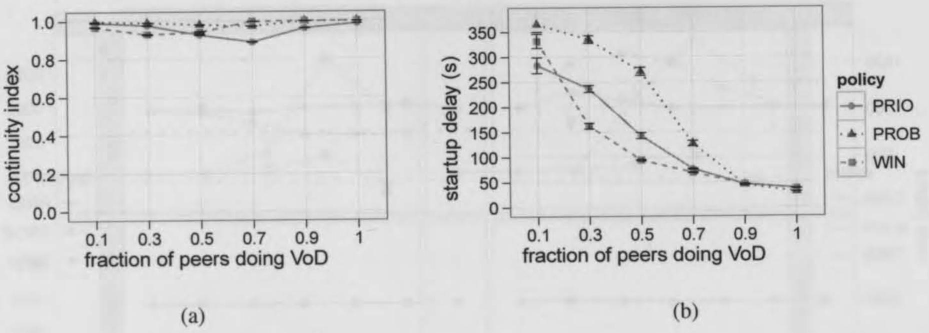


Figure 3.16: (a) continuity index and (b) startup delay in a system where some peers are doing traditional file transfer while others are doing streaming. The peer arrival rate is set to 0.05 peer/s.

have taken a look at the peers interests. Figure 3.15 shows, for each group of peers, the percentage of interesting peers belonging to the other group, in a scenario where VoD peers are 50% of the total. While for the priority-based and the probabilistic policies the interest of each group in the other one is more or less even, we observe that, for the window-based policy, there are more file transfer nodes interested in VoD nodes (68%) than *vice versa* (47%). We conjecture that the unbalanced interests between the two groups are the main cause of the phenomenon, which can be explained as follows. At first, file transfer nodes download the pieces towards the end of the file (as those pieces are more rare). Later, they need to download the pieces towards the beginning of the file as well, in order to complete their downloads. These pieces are owned mostly by VoD nodes, therefore file transfer peers become very interested in them. On the other hand, VoD nodes using the window-based policy only download pieces within a relatively small window, therefore they will not be very interested in file transfer peers. This unbalance between the two groups' interests causes the VoD nodes to receive many more requests from file transfer nodes than other VoD nodes, which then will be (as a group) optimistically unchoked more often. However, file transfer peers will not have much to upload in return, given the myopic interest of VoD nodes using the window-based policy. As a consequence, file transfer nodes download from VoD nodes much more than they upload to them in return.

Furthermore, regardless of the piece selection policy employed, VoD nodes suffer from longer startup delays. According to Figure 3.16, when the fraction of VoD nodes is 0.1, for example, their startup delay is, for all piece selection policies, more than 10 times longer than it would be in a system where all nodes are doing streaming. This is due to the fact that, the more there are file transfer nodes, the lesser the availability of pieces near the beginning of the file (since file transfer nodes download pieces according to the rarest-first rule). Therefore, the longer it takes for the VoD nodes to complete the download of the initial pieces necessary for the sequential viewing.

Turning our attention back to download rates (Figure 3.14), we see that the priority-based and the probabilistic policies are able to maintain the rates of VoD nodes constantly far above the playback rate (dashed horizontal line in Figure 3.14). This leaves room for improvement for these two policies. In fact, similarly to the cases of heterogeneous and unconnectable nodes, bandwidth can be allocated in a “smarter” way by having each VoD peer help other VoD ones when its own QoS is good enough.

3.9 Summary of our findings

Table 3.4 summarizes all the experiments done for the tested policies. This table can aid the selection of the best protocol to be used by a service provider, given its particular environment and system characteristics such as open/closed system, with or without unconnectable nodes, or nodes with heterogeneous bandwidth, etc. For each test case we have selected the best performing piece selection policy for both direct and indirect reciprocity (in the Direct and Indirect columns, respectively). We also show, in the ‘Overall’ column, which policy combination performed the best for each scenario. The last column indicates how well BitTorrent-based VoD performs in the given scenario in absolute terms (full circle means ‘good’, half circle stands for ‘average’, and dash means ‘bad’). When using this table, the reader should be aware of two things. Firstly, the Sybil attack presented in this chapter can only be performed in a protocol using indirect reciprocity, thus all protocols using direct reciprocity are resilient to it. Secondly, traditional BitTorrent works with direct reciprocity only. Therefore, in order to keep the VoD and the file transfer protocols homogeneous, we have focused on the case of direct reciprocity only.

Table 3.4: Summary of findings.

Test case	Direct	Indirect	Overall	Absolute
Freeriding resilience	PROB	PROB	PROB, indirect	●
Resilience to Sybil attack	all	PRIO	all, direct	●
Heterogeneous bandwidths	WIN	WIN	WIN, indirect	●
Unconnectable peers	WIN	WIN	WIN, direct	◖
Coexistence with trad. file transfer	PRIO	n.a.	PRIO, direct	◖

3.10 Conclusion

In this chapter, we have taken the first steps towards developing a deeper understanding of how the different BitTorrent-based P2P approaches to VoD work.

We have shown that in all current approaches a trade-off exists between QoS and freeriding-resilience, and between QoS and security, especially at low peer arrival rates. However, protocols using a probabilistic piece selection generally seem to offer the most advantageous trade-off.

Furthermore, we have shown that the current incentive schemes adopted in BitTorrent-based VoD systems are not suitable to VoD when the system is heterogeneous. This is due to the fact that current incentives are based on a general file transfer goal rather than a VoD goal. In file transfer, the goal is to maximize the total download speed, therefore an incentive based on bandwidth reciprocity induces peers to contribute as much as they can. On the other hand, in VoD the goal is to have as many peers as possible experience a smooth playback. In other words, a peer does not earn any benefit in downloading at much higher rates than the playback rate. Similarly, we have discovered that unconnectable peers generally receive low download speeds, and consequently experience low QoS, even when they are only 30-50% (which is a realistic range in P2P swarming systems, see [45] and references therein) of the total number of nodes in the system and even when methods to increase the reachability of these nodes are in place. Connectable peers, on the other hand, download much faster than needed to maintain a good playback continuity.

Finally, we have evaluated the case where one VoD protocol competes with the original BitTorrent protocol and found out that, generally, this coexistence is disadvantageous for VoD peers, much to the benefit of file-transfer peers.

Chapter 4

Peer selection strategies for improved QoS in heterogeneous BitTorrent-based P2PVoD systems

In the previous chapter, we have observed that the approach to peer selection employed by many BitTorrent-based P2PVoD protocols has limitations in providing a good QoS to users, particularly when they have highly heterogeneous upload capacities or are located behind a NAT/firewall that does not allow incoming connections.

The problem with peers having heterogeneous capacities resides in the fact that peer selection is based on bandwidth reciprocity. In a file-transfer context, this principle incentivizes users to contribute more bandwidth by rewarding them with faster download speeds. In this way, the users obtain the file of interest earlier. In VoD, however, a peer selection mechanism based on bandwidth reciprocity can result in high-capacity peers receiving download rates substantially higher than the video playback rate, and in low-capacity peers experiencing download rates that are too low.

For what concerns system with unconnectable nodes, in Chapter 2 we have illustrated that the problem lies in that connectable nodes provide their bandwidth to both connectable and unconnectable ones, while unconnectable peers can only upload to the former. As a result, connectable nodes achieve, on average, faster download rates than unconnectable ones.

In this chapter, we focus on both these issues, and aim at increasing the number of peers that are able to meet the QoS requirements needed for streaming, while maintaining the incentives for cooperation of the reciprocity-based piece selection mechanism. Our approach is to let nodes act more altruistically if they are receiving a service that is higher than necessary to preserve stream continuity. More specifically, our contributions are as follows:

1. We propose an adaptive strategy for peers to decide, in relation to their current progress, whether relaxing the reciprocity-based peer selection and serve more ran-

dom peers (Section 4.1.2); in this way, when bandwidth is abundant, a larger fraction of peers can achieve a satisfactory QoS, in terms of both startup delay and stream continuity;

2. We extend this adaptive mechanism to have nodes give preference to newly joined peers (Section 4.1.3); by doing so, the average startup delay can be reduced even further, with little or no negative effect on the stream continuity;

3. We further extend the initial adaptive mechanism to have nodes give preference to unconnectable peers, in this way increasing their QoS (Section 4.1.4);

Simulation results show that, in heterogeneous systems where bandwidth is abundant, our adaptive peer selection policy significantly increases the number of low-capacity peers receiving a satisfactory QoS (up to 4 times), without affecting that of high-capacity peers. When extended to give preference to newcomers, this strategy can reduce startup delays (up to 48% in the scenarios considered) with no significant impact on the stream continuity. When the preference is given to unconnectable peers, their QoS is increased without negative effect for those that are connectable. Finally, we show that, if bandwidth is scarce, higher-contributors are prioritized, proving that the incentive for cooperation is retained (Section 4.3).

4.1 Proposed Peer Selection Strategies

In this section, we present our proposed strategies to improve the QoS of peers in a heterogeneous environment.

4.1.1 Proportional slot number

The first step towards our adaptive strategies consists of adjusting the number of upload slots each peer opens. In the original BitTorrent protocol, the number of upload slots is generally set to a constant (normally 4) [9, 21, 42]. Of these upload slots, usually one is reserved for optimistic unchoke, while the others are used for regular unchokes.

However, this strategy may lead to a scenario where a high-capacity peer will provide each of its (few) children with a very high bandwidth, while some other nodes in the system are experiencing insufficient speeds to achieve reasonable QoS. Since peers do not earn more utility in downloading at rates substantially higher than the video playback rate and balancing the service each peer receives is desirable, it makes more sense that a node limits the bandwidth provided to each child and opens a number of upload slots according to its total upload bandwidth, as well as the video playback rate. We propose the following rule to calculate how many upload slots s a peer p opens:

$$s = \max\left(s_0, \left\lceil s_0 \frac{\mu}{R} \right\rceil\right), \tag{4.1}$$

where μ is p's upload capacity, R is the video playback rate and s_0 is a protocol parameter which represents the minimum number of upload slots each peer opens. With this rule, a high-capacity peer ($\mu > R$) will try to provide each of its children with an upload capacity of $\sim \frac{R}{s_0}$. In this way, no peer will receive from another peer pieces at a rate higher than R (unless the uploading peer has some more capacity left, i.e. if the other nodes it is uploading to hold download bottlenecks) and the bandwidth of high-capacity nodes can be shared among more peers.

Given a number of slots, it is necessary also to determine how many of them are reserved for optimistic unchoke. Leaving only one slot for this purpose means that, the larger a peer's bandwidth, the smaller the fraction of it dedicated to optimistic unchoke. Jia *et al.* [12] show that doing so leads to a larger service difference among heterogeneous peers. To avoid that, nodes should reserve always the same fraction of their bandwidth to the optimistic unchoke. Hence, we propose that each peer is initialized with o_{min} optimistic unchoke slots where o_{min} is:

$$o_{min} = round\left(\frac{s}{s_0}\right), \tag{4.2}$$

in this way, each peer will assign $\sim \frac{1}{s_0}$ of its upload capacity to optimistic unchokes.

In the strategies that follow, we assume that a peer's upload slot number is always initialized according to this rule, which we call the *proportional slot number rule* (PSN).

4.1.2 Adaptive optimistic unchoke slot number

The core of this strategy is to have peers dynamically adjust the number of their optimistic unchoke slots to their current QoS. To measure the QoS a peer is currently experiencing, we use the peer's current sequential progress. Recall, from Chapter 3, that a peer's sequential progress is defined as the speed at which the index of the first missing piece in the file grows. Therefore, a peer's sequential progress is an indicator for the preservation of the continuity of the stream: a sequential progress faster than the video playback rate implies a continuous stream. At the same time, this metric is agnostic with respect to the underlying piece selection policy, making a strategy based on it directly portable to other P2PVoD protocols.

Since a peer p does not earn more utility in having a sequential progress much higher that the video playback rate R, we propose that, when this is the case, p will decide to "help" other peers by increasing the number of its optimistic unchoke slots. On the other hand, it will reduce its optimistic unchoke slots if its sequential progress has decreased past a limit.

Algorithm 1 Adapting optimistic unchoke slot number to sequential progress

```
1:  for each optimistic unchoke round do
2:      $R$ := video playback rate
3:      $t$ := time now
4:      $w$ := window length
5:      $T_{up}$ := upper threshold for the sequential progress
6:      $T_{lw}$ := lower threshold for the sequential progress
7:      $s$ := total number of upload slots
8:      $o$ := number of optimistic unchoke slots
9:      if sequential progress($t$,$w$) $> T_{up}$ then
10:         $o \Leftarrow \min(s, o+1)$
11:     if sequential progress($t$,$w$) $< T_{lw}$ then
12:         $o \Leftarrow \max(o_{min}, o-1)$
13:     if sequential progress($t$,$w$) $< R$ then
14:         $o \Leftarrow o_{min}$
```

To avoid switching too often between increasing and decreasing the number of optimistic unchoke slots, we use the following approach. First, we define an interval (T_{lw}, T_{up}) for the equilibrium value of the sequential progress. A peer p will then increase the number of its optimistic unchoke slots only when its sequential progress is above the upper threshold T_{up} and decrease it when its sequential progress goes below the lower threshold T_{lw}. Second, at every optimistic unchoke round, the current sequential progress is calculated over the last w seconds (Algorithm 1). By opportunely tuning the thresholds T_{lw} and T_{up} and the length of the window w, the performance of other peers can improve, while that of peer p will stay constant. In particular, in order to guarantee that peer p does not get worse off from being too altruistic, it is advisable for T_{lw} to have a value larger than R. Similarly, the size of the window w should be large enough to capture the trend of a peer's sequential progress, besides eventual instantaneous oscillations, but small enough to be sensitive to a change of trend. Finally, in order to protect peers from a very rapid change in their bandwidth supply, the number of a peer's optimistic unchoke slots will be immediately set to the minimum value o_{min}, when its sequential progress becomes lower than the video playback rate R.

4.1.3 Priority to newcomers in optimistic unchoke slots

When a new peer joins the network, it needs to wait for a certain period of time to be optimistically unchoked by other peers. Once a node has obtained the first piece, it can forward it to other peers and get a chance of being selected by its parents in their regular unchoke slots. We denote this time in which peers are waiting for the first piece as the "bootstrap time". It has to be noticed that peers are subjected to a bootstrap time also in a regular BitTorrent system. However, compared to the total download time that nodes have to wait before they can access the file, the bootstrap time becomes insignificant. On the contrary,

Algorithm 2 Unchoke algorithm with priority to newcomers (PNC)

```
 1: for each unchoke round do
 2:   $s$ := total number of upload slots
 3:   $o$ := number of optimistic unchoke slots
 4:   $c \Leftarrow$ number of newcomers that can be preferentially unchoked ($c \leq o$)
 5:   $P \Leftarrow$ list of all choked neighbors having no pieces yet
 6:   $Q \Leftarrow$ list of all neighbors
 7:   sort($Q$) /* sort list according to contribution (from high to low) */
 8:   $c' \Leftarrow \min(c, |P|)$ /* $|P|$ is the number of peers in $P$ */
 9:   for $i = 1$ to $\min(s - o)$ do /* regular unchoke */
10:     unchoke($Q[i]$)
11:   for $i = 1$ to $c'$ do /* optimistic unchoke: first, unchoke the newcomers */
12:     unchoke($P[i]$)
13:   $N \Leftarrow$ all choked interested neighbors
14:   if $(o - c') > 0$ then /* if there are some slots left, then */
15:     for $i = 1$ to $(o - c')$ do /* unchoke some other peers at random */
16:       unchoke($N[i]$)
```

in a VoD system, where playback starts well before the file is completely downloaded, the bootstrap time can represent an important fraction of the startup delay [63].

Hence, the startup delay can be reduced by decreasing the bootstrap time. In order to do so, we propose to extend the previous strategy with a mechanism where nodes give *priority to newcomers* when performing optimistic unchoke. In this way, newcomers will not need to wait too long before being unchoked for the first time. Algorithm 2 illustrates the pseudo-code for this strategy. When deciding how many newcomers will be given priority to, there are two things to consider: the new strategy should not (i) disrupt one of the original purpose of optimistic unchoke slots, i.e. finding new, potentially better, neighbors, nor (ii) neutralize the benefits of the adaptive strategy introduced above. In order to do so, we propose a *priority to newcomers strategy* (PNC) where the number c of newcomers that will be given priority is calculated as follows:

$$c = \left\lfloor \frac{o - o_{min}}{2} \right\rfloor, \tag{4.3}$$

where o is the current number of optimistic unchoke slots opened by a peer p.

Hence, when p has only o_{min} optimistic unchoke slots, it will assign them to randomly selected peers, in this way we eliminate the risk that a node gets worse off from helping newcomers. A node p will favor newcomers only when it has at least $o_{min} + 2$ optimistic unchoke slots (meaning that its current QoS is high) and will use at most half of them, so that there are enough left for helping regular peers as well.

To prevent nodes from lying about being newcomers, a peer will only upload to newcomers pieces from an initial buffer (i.e. any piece from the first h pieces of the file).

Algorithm 3 Unchoke algorithm with priority to unconnectable nodes (PUN)

```
1: for each optimistic unchoke round do
2:     $R$ := video playback rate
3:     $t$ := time now
4:     $w$ := window length
5:     $T_{up}$ := upper threshold for the sequential progress
6:     $T_{lw}$ := lower threshold for the sequential progress
7:     $s$ := total number of upload slots
8:     $o$ := number of optimistic unchoke slots
9:     $\delta \Leftarrow min(1, \frac{\alpha}{1-\alpha})$
10:    if sequential progress($t$,$w$) $> T_{up}$ then
11:        $n^u \Leftarrow \delta(s - o)$
12:    if sequential progress($t$,$w$) $< T_{lw}$ then
13:        $n^u \Leftarrow max(0, n^u - 1)$
14:    if sequential progress($t$,$w$) $< R$ then
15:        $n^u \Leftarrow 0$
16: for each unchoke round do
17:    $U \Leftarrow$ list of all unconnectable neighbors
18:    sort($U$) /* sort list according to contribution (from high to low) */
19:    $C \Leftarrow$ list of all connectable neighbors
20:    sort($C$) /* sort list according to contribution (from high to low) */
21:    for $i = 1$ to $n^u$ do /* unchoke the unconnectable nodes */
22:        unchoke($U[i]$)
23:    for $i = 1$ to $s - o - n^u$ do /* unchoke the connectable nodes */
24:        unchoke($C[i]$)
25:    $N \Leftarrow$ all choked interested neighbors
26:    for $i = 1$ to $o$ do /* optimistic unchoke */
27:        unchoke($N[i]$)
28:        put $N[i]$ at the bottom of the list of connections /* so that at the next optimistic unchoke
           round another peer is chosen */
```

4.1.4 Priority to unconnectable nodes in regular unchoke slots

We can apply the same adaptive principle to have connectable nodes give priority to unconnectable ones when they are experiencing a good enough service. In Chapter 2 we have suggested that connectable peers allocate a fraction δ of their bandwidth to unconnectable peers, where δ depends on the fraction α of unconnectable peers in the system. For simplicity, here we assume that peers have homogeneous bandwidths, and therefore we have $\delta = min(1, \frac{\alpha}{1-\alpha})$.

Furthermore, we note that, different from slow peers, unconnectable peers have the potential of being good bartering partners, as their upload capacity is as high as that of connectable ones. They just need to be given a "larger chance" to prove it. Therefore, in this section, we propose an adaptive strategy where connectable nodes that are already experiencing

Table 4.1: Simulation Settings

Parameter	Value
Video playback rate R	800 kb/s
Video length L	50 min
Simulation time	500 min ($10L$)
Peer arrival rate	0.45 peers/s
Min slot number s_0	4
Piece size P_S	256 kB
Piece selection	PRIO
Peer selection	DIRECT
Initial buffer h	25 pieces
Upload rate of the service provider U_s	8000 kb/s ($10R$)

a high QoS give *priority to unconnectable nodes* in their regular unchoke slots. Algorithm 3 illustrates the pseudo-code for this strategy. The number of unconnectable peers to be give priority to is weighted by the factior δ, which guarantees that such unconnectable peers are not given more bandwidth than their fair share (see Chapter 2). Hence, according to this *priority to unconnectable nodes strategy* (PUN), the number n^u of unconnectable peers that will be given priority is calculated as follows:

$$n^u = \delta(s - o), \tag{4.4}$$

where s and o are the current numbers of total and optimistic unchoke slots, respectively, opened by a peer p. The mechanism that regulates the level of altruism of a peer is similar to the previous cases, so to avoid that a connectable nodes gets a bad performance from helping unconnectable ones too much.

4.2 Methodology

We evaluate the effects of the proposed strategies by means of simulations.

4.2.1 Experimental setup

We have extended our version of the MSR BitTorrent simulator presented in the previous chapter to support all the strategies introduced in Section 4.1.

The settings for our simulations are shown in Table 4.1. We consider the system to be in steady state. For the results, we only take onto account the peers who have joined after the first half of the simulation time and before the last L minutes from the simulation end. For the adaptive strategies, we have used $T_{lw} = 1.2R$, $T_{up} = 1.4R$ and $w = 2min$, as in our experiments these values have shown to provide a good trade-off between maximizing altruism and retaining a good QoS for the altruistic peers.

Furthermore, in our simulations we assume that (i) all peers have a download capacity five times higher than their upload capacity, (ii) peers leave as soon as their download is complete, (iii) there is one peer holding a complete copy of the file, set up by the provider, which never leaves the system.

Each simulation run is executed 15 times, and then the average values and the confidence intervals (with confidence level of 95%) for each of the following metrics are computed.

4.2.2 Performance metrics

Habib *et al.* [28] show that when the continuity index of a stream is 95%, a user's overall perceived video quality is very good. Similarly, they show that a CI $\leq$ 75% leads to a poor video quality. Hence, in this chapter, we evaluate the ability of the P2P VoD system to maximize the number of peers receiving a high stream continuity by characterizing the fraction of peers receiving a good service (i.e. whose CI is above 95%) and poor service (i.e. whose CI index is below 75%).

To analyze how well the startup delay minimization requirement is met, we measure the average and the 95th percentile of the startup delay.

4.3 Experiments and analysis

In this sectiom, we evaluate the performance of our strategies. We treat the cases of heterogeneous peers and unconnectable peers separatedly.

4.3.1 Heterogeneous environment

We consider three types of scenarios as described below:

Overprovision scenario: the average peer upload capacity is larger than the video playback rate R;

Contention scenario: the average peer upload capacity is smaller than the video playback rate R;

Dynamic scenario: the average peer upload capacity is initially larger than the video playback rate R but suddenly becomes smaller in the middle of the simulation.

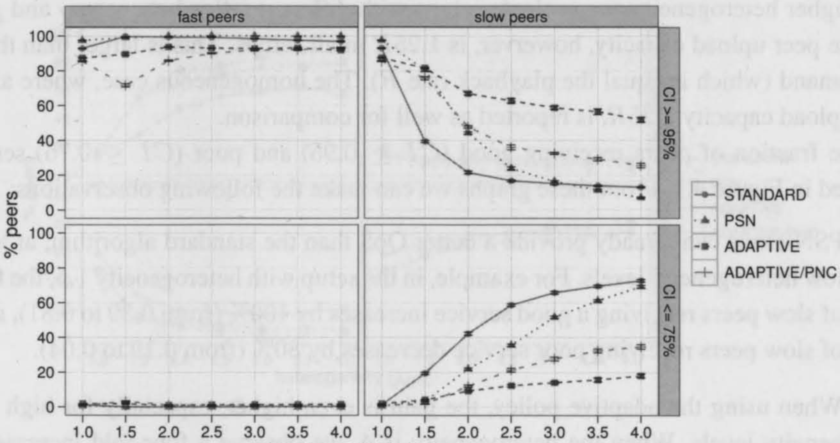


Figure 4.1: Overprovision scenario: fraction of peers receiving a good service ($CI \geq 0.95$) or a poor service ($CI \leq 0.75$).

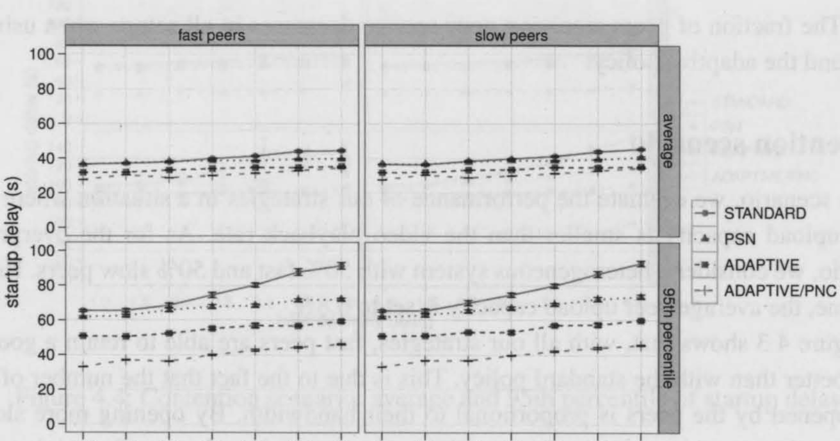


Figure 4.2: Overprovision scenario: average and 95th percentile of startup delay.

Overprovision scenario

In this scenario, we consider a system where 50% of the peers have a high upload capacity and the other 50% have a slow upload capacity, and we denote with μ_f and μ_s the bandwidths of fast and slow nodes, respectively. In order to evaluate how the strategies perform with higher heterogeneity, we evaluate setups with different ratios between μ_f and μ_s. The average peer upload capacity, howerver, is $1.25R$ in all setups. This is larger than the average demand (which is equal the playback rate R). The homogeneous case, where all peers have upload capacity $1.25R$, is reported as well for comparison.

The fraction of peers receiving good ($CI \geq 0.95$) and poor ($CI \leq 0.75$) service is depicted in Figure 4.1. From these graphs we can make the following observations:

- PSN alone can already provide a better QoS than the standard algorithm, at least for low heterogeneity levels. For example, in the setup with heterogeneity 1.5, the fraction of slow peers receiving a good service increases by 100% (from 0.39 to 0.81), and that of slow peers receiving poor service decreases by 80% (from 0.19 to 0.04).

- When using the adaptive policy, the gain is even higher, especially for high heterogeneity levels. When the heterogeneity is 4, we observe a four-fold increase in the fraction of slow peers receiving good service (from 0.13 to 0.52).

- In all setups, both PSN and ADAPTIVE do not significantly affect the service received by fast peers.

- The fraction of peers receiving poor service decreases in all setups when using PSN and the adaptive policy.

Contention scenario

In this scenario, we evaluate the performance of our strategies in a situation where the average upload capacity is smaller than the video playback rate. As for the overprovision scenario, we consider a heterogeneous system with 50% fast and 50% slow peers. However, this time, the average peer upload capacity is set to $0.8R$.

Figure 4.3 shows that, with all our strategies, fast peers are able to retain a good QoS, often better than with the standard policy. This is due to the fact that the number of upload slots opened by the peers is proportional to their bandwidth. By opening more slots, fast peers can barter with more partners and hence have a higher chance of receiving enough bandwidth in return. Slow peers, on the contrary, suffer from a low QoS. This shows that, when bandwidth is scarce, our adaptive strategies do retain the incentives for cooperation of the original BitTorrent protocol. Furthermore, observing Figure 4.4, we can conclude that our proposed strategies are able to reduce the overall startup delays, despite the contention of resources in the system, and with only little effect on the CIs experienced by the peers.
Figure 4.2 reports the mean and the 95th percentile of the startup delay. We can notice that:

- In the homogeneous setup, ADAPTIVE/PNC can substantially reduce the startup delays, (25% for the average and 50% for the 95th percentile), with only little impact on the respective CIs (see Figure 4.1).

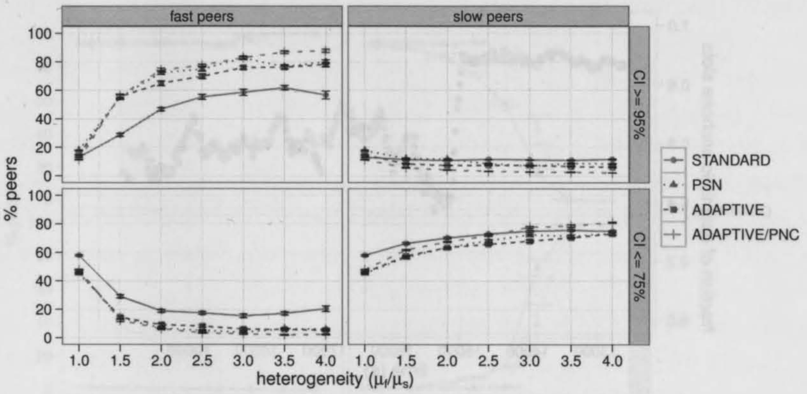


Figure 4.3: Contention scenario: fraction of peers receiving good service ($CI \geq 95\%$) and poor service ($CI \leq 75\%$).

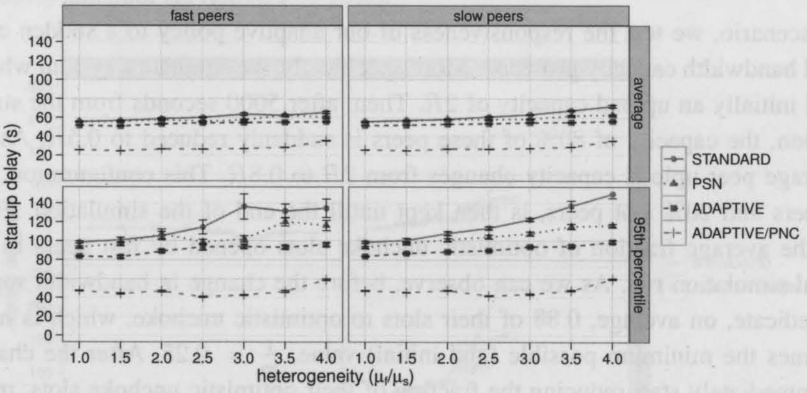


Figure 4.4: Contention scenario: average and 95th percentile of startup delay.

- In the heterogeneous setups, using PSN or ADAPTIVE policy already results in decreased startup delays for all peers. In fact, by opening more upload slots (PSN) or using them altruistically (ADAPTIVE), a higher fraction of peers will be chosen for optimistic unchoke, including newcomers. This phenomenon becomes more and more beneficial as the level of heterogeneity increases.

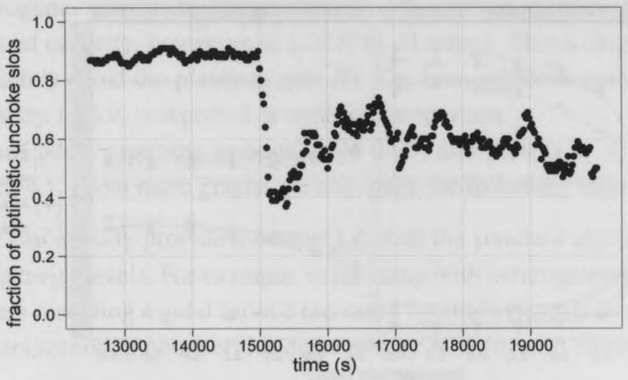


Figure 4.5: Dynamic scenario: fraction of slots fast peers dedicate to optimistic unchoke.

Dynamic scenario

In this scenario, we test the responsiveness of our adaptive policy to a sudden change in the total bandwidth capacity provided. More specifically, we simulate a system where peers have all initially an upload capacity of $2R$. Then, after 5000 seconds from the start of the simulation, the capacity of 80% of these peers is suddenly reduced to $0.5R$. As a result, the average peer upload capacity changes from $2R$ to $0.8R$. This configuration, i.e. 80% slow peers and 20% fast peers, is then kept untill the end of the simulation. Figure 4.5 shows the average fraction of optimistic unchoke slots opened by fast peers in time for a typical simulation run. As we can observe, before the change in bandwidth supply, fast peers dedicate, on average, 0.88 of their slots to optimistic unchoke, which is more than three times the minimum possible (and initial) value, $\frac{1}{s_0} = 0.25$. After the change, fast peers immediately start reducing the fraction of their optimistic unchoke slots, reaching a minimum of 0.37 in about 6 minutes. This value then gradually approaches 0.6.

4.3.2 Scenario with unconnectable peers

In this scenario, all peers have an upload capacity $\mu = 1.25R$, but a fraction α of them are unconnectable. Figures 4.6 and 4.7 show the continuity index and the startup delay, respectively, as we increase the fraction of unconnectable nodes. As we can see, for low values of α, ADAPTIVE/PUN can significantly improve the QoS of unconnectable peers, with no impact on the connectable ones. For example, when $\alpha = 0.5$, the fraction of unconnectable peers receiving a good QoS when using ADAPTIVE/PUN is double than when using the standard policy; similarly, the startup delay is reduced by 30%. Finally, we note that, as expected from our analysis in Section 2, when the fraction of unconnectable peers has gone

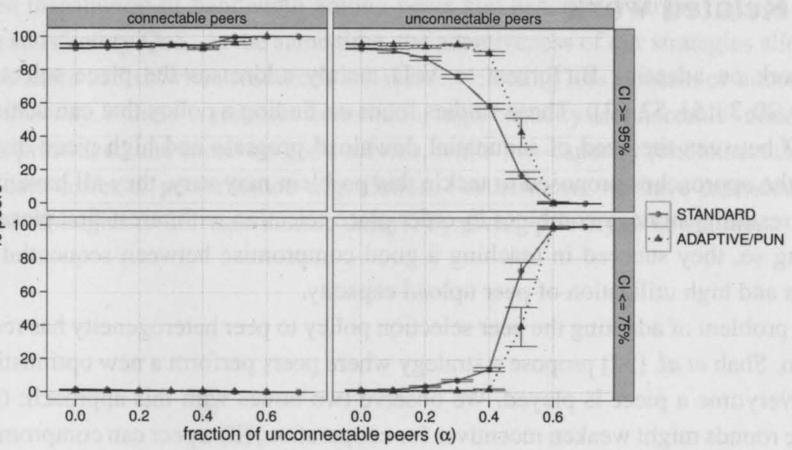


Figure 4.6: Scenario with unconnectable peers: fraction of peers receiving good service ($CI \geq 95\%$) and poor service ($CI \leq 75\%$).

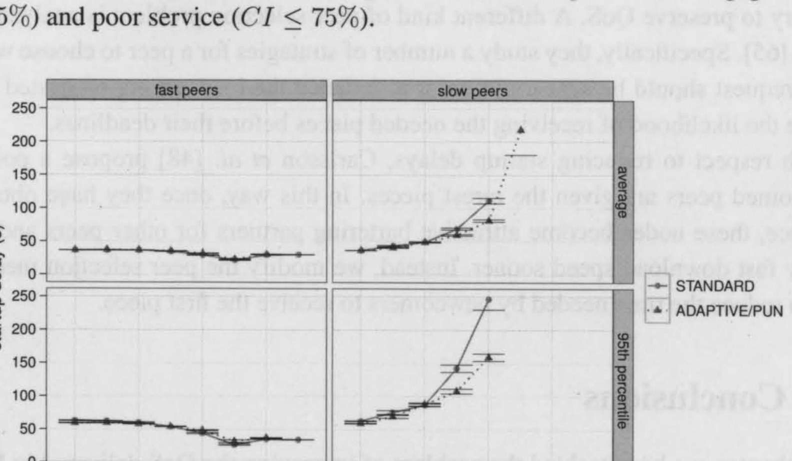


Figure 4.7: Contention scenario: startup delay.

past 50%, their QoS drops significantly, even when using ADAPTIVE/PUN. This is due to the fact that, no matter how bandwidth is allocated in the system, a minority of connectable peers can never sustain the bandwidth needs of a majority of unconnectable peers.

4.4 Related work

Prior work on adapting BitTorrent to VoD mainly addresses the piece selection policy (see [10, 20, 33, 51, 52, 63]). These studies focus on finding a policy that can achieve a good trade-off between the need of sequential download progress and high piece diversity. Although the approaches proposed to tackle this problem may vary, they all have in common that the resulting strategy combines in-order piece selection with rarest-first piece selection. By doing so, they succeed in reaching a good compromise between sequential download progress and high utilization of peer upload capacity.

The problem of adapting the peer selection policy to peer heterogeneity has received less attention. Shah *et al.* [52] propose a strategy where peers perform a new optimistic unchoke round everytime a piece is played. We observe two issues with this approach: (i) frequent unchoke rounds might weaken incentives for cooperation, (ii) a peer can compromise its own QoS by being too altruistic. In contrast, with our proposed strategies, a peer will behave more altruistically only when it is receiving a service that is substantially better than the necessary to preserve QoS. A different kind of peer selection problem is analyzed by Yang *et al.* in [65]. Specifically, they study a number of strategies for a peer to choose which other node a request should be sent to, in order to balance the load among requested nodes and increase the likelihood of receiving the needed pieces before their deadlines.

With respect to reducing startup delays, Carlsson *et al.* [48] propose a policy where newly joined peers are given the rarest pieces. In this way, once they have obtained their first piece, these nodes become attractive bartering partners for other peers and are likely to enjoy fast download speed sooner. Instead, we modify the peer selection mechanism in order to reduce the time needed by newcomers to receive the first piece.

4.5 Conclusions

In this chapter, we have tackled the problem of improving the QoS delivered in BitTorrent-based VoD systems with heterogeneous and unconnectable peers through better peer selection policies. We have shown that, when the resources in the system are abundant and peers have heterogeneous bandwidths, rewarding them proportionally to their upload capacities is neither (i) system-wide efficient, since less peers are granted a satisfactory QoS, nor (ii) individually profitable, since peers do not earn utility in downloading at rates much faster than the video playback rate. Furthermore, we have also demonstrated that unconnectable peers also receive poor service, due to the fact that they have to compete with connectable peers for the upload capacity of the latter ones.

We have proposed a number of peer selection policies to overcome these effects, where nodes that are already receiving a high enough QoS adaptively increase the fraction of their bandwidth allocated to random peers or unconnectable ones. These strategies result in a more even distribution of bandwidth among peers and hence in a larger fraction of nodes receiving satisfactory QoS. At the same time, the adaptiveness of our strategies allows peers to become more selfish when resources are scarce, selecting less random or unconnectable peers. In this way, if there is resource contention, high-capacity (connectable) nodes are still able to be prioritized and receive a good service, while low-capacity (unconnectable) nodes or free-riders suffer of performance degradation. Furthermore, we have extended the idea behind our strategies to decrease startup delays, by giving priority to newcomers.

[illegible] that the resulting [illegible] selection. By doing so, they succeed in reaching a good compromise between sequential download progress and high utilization of peer upload capacity.

The problem of adapting the peer selection policy to peer heterogeneity has received less attention. Shah *et al.* [52] propose a strategy where peers perform a new optimistic unchoke round everytime a piece is played. We observe two issues with this approach: (i) frequent unchoke rounds might weaken incentives for cooperation, (ii) a peer can compromise its own QoS by being too altruistic. In contrast, with our proposed strategies, a peer will behave more altruistically only when it is receiving a service that is substantially better than the necessary to preserve QoS. A different kind of peer selection problem is analyzed by Yang *et al.* in [57]. Specifically, they study a number of strategies for a peer to choose which other node a request should be sent to, in order to balance the load among seed nodes and increase the likelihood of receiving the needed pieces before their deadlines.

With respect to reducing startup delays, Carlsson *et al.* [48] propose a policy where newly joined peers are given the rarest pieces. In this way, once they have obtained their first piece, these nodes become attractive bartering partners for other peers and are likely to enjoy fast download speed sooner. Instead, we modify the peer selection mechanism in order to reduce the time needed by newcomers to receive the first piece.

4.5 Conclusions

In this chapter, we have tackled the problem of improving the QoS delivered in BitTorrent-based VoD systems with heterogeneous and unconnectable peers through better peer selection policies. We have shown that, when the resources in the system are abundant and peers have heterogeneous bandwidths, rewarding them proportionally to their upload capacities is neither (i) system-wide efficient, since less peers are granted a satisfactory QoS, nor (ii) individually profitable, since peers do not earn utility in downloading at rates much faster than the video playback rate. Furthermore, we have also demonstrated that unconnectable peers also receive poor service, due to the fact that they have to compete with connectable peers for the upload capacity of the latter ones.

We have proposed a number of peer selection policies to overcome these effects, where nodes that are already receiving a high enough QoS adaptively increase the fraction of their bandwidth allocated to random peers or unconnectable ones. These strategies result in a

Chapter 5

Analyzing the scalability of BitTorrent-based P2PVoD systems during flashcrowds

While many studies have been conducted on the scalability of BitTorrent-based P2P live video streaming systems, both during steady-state and during flashcrowds [24,59], the analysis of the BitTorrent-based approach for VoD has mostly focused on the steady-state [50,64]. Since flashcrowds do occur in VoD systems as well [30], it is important for service providers that use P2P technology to know how well such systems scale in these circumstances. In this way, the appropriate measures can be taken to keep providing their users with an acceptable level of service (e.g. by adding extra capacity or denying access to newcomers when the system load is to high).

In this chapter, we take a first step towards analyzing this problem. We analytically study the scalability of a BitTorrent-based P2PVoD system during flashcrowds and characterize its relationship with the underlying P2P protocol, the initial service capacity, and the creation of new seeders. More specifically, the contributions of this chapter are the following:

1. We propose a simple model to study the effects of a flashcrowd on the startup delay experienced by the users (Section 5.1);
2. Based on this model, we show that the scale of a BitTorrent-based P2PVoD system during flashcrowd is constrained by the efficiency of piece exchange of the underlying P2P protocol as well as the initial service capacity (Section 5.2).
3. We illustrate the impact of peers turning into seeders on the system scale (Section 5.2).

5.1 Flashcrowd model for a P2PVoD system

In this section, we will derive a model of a BitTorrent-based P2PVoD system during a flashcrowd. We distinguish two phases in a flashcrowd: an initial phase, with no seeders, and a second phase where seeders have started appearing. For each phase, we present a set of differential equations that describe the evolution of a typical BitTorrent-based P2PVoD system. We then analyze the average download speed of peers, and derive the average startup delay from it. Finally, we provide explicit equations that characterize the dependancy of the startup delay on different parameters, such as the peer arrival rate, the initial service capacity, and the efficiency of the underlying P2P protocol.

5.1.1 General approach

We base our analysis on a fluid approximation of a Markov model which describes the arrival and departure rates of the peers in the system. In this approach, the discrete quantities of the number of leechers and seeders in a swarm are modeled as a Markov chain. We follow a similar modeling procedure as the classical fluid model for BitTorrent [53], as well as the P2PVoD models in [50, 64].

5.1.2 Model description and assumptions

We consider a system where a group of peers joins to watch a specific video file with playback rate R and duration L. We assume that there is an *initial seeder* or *source* (consisting of one or more servers) which contributes an aggregate upload capacity C_s. Peers have an average upload capacity μ and join the system at rate $\lambda(t)$ and do not leave before they have become seeders. Seeders, on the other hand, depart from the system at a rate γ (hence, we can consider $\frac{1}{\gamma}$ to be the average seeding time). We also assume that each peer tries to utilize its upload bandwidth as much as it can, a behavior which is enforced by most of BitTorrent-based P2PVoD systems by means of contribution incentives.

5.1.3 Flashcrowd scenario

When a flashcrowd occurs, the system enters a critical phase where the available bandwidth is scarce. After some time, the appearance of seeders will alleviate the negative performance impact, even if peers continue joining at a high rate. Therefore, similar to Lu *et al.* [64], in our analysis we distinguish two phases in a flashcrowd, depending on whether seeders have started appearing or not.

1. ***Starting phase*** $(0 < t < t_s)$

   Excluding the initial source, there is no seeder in the system until the first leechers complete the download of the entire video file (which happens at time t_s), thus becoming the first seeders. Hence, at the very beginning of a flashcrowd, the evolution of leechers $x(t)$ and seeders $y(t)$ in the system is characterized by the following equations

$$\begin{cases} \frac{dx(t)}{dt} = \lambda(t), \\ \frac{dy(t)}{dt} = 0, \\ x(0) = \lambda(0), y(0) = 0, \end{cases} \tag{5.1}$$

and the average download speed can be computed as

$$u(t) = \frac{C_s + \mu\eta x(t)}{x(t)} = \frac{C_s}{x(t)} + \mu\eta. \tag{5.2}$$

The parameter η indicates the *efficiency of piece exchange*. When $\eta = 0$, the leechers do not get any data from each other, meaning that their upload bandwidth is completely unutilized.

From Eq. (5.2) it is also obvious that $u(t)$ is a non-increasing function in the time interval $(1, t_s)$.

The time t_s, at which the first leechers complete their downloads, can be calculated as the time needed to download LR bytes of data (LR represents the size of the video file)

$$t_s = \frac{LR}{u(t_s)} = \frac{LR}{\frac{C_s}{x(t_s)} + \mu\eta} = \frac{LR}{\frac{C_s}{\int_0^{t_s} \lambda(t)dt} + \mu\eta} \tag{5.3}$$

Eq. (5.3) can be solved with numerical analysis (e.g. by using the fixed point iteration method).

2. ***Seeders appearance phase ($t > t_s$)***

In this phase, the number of leechers that turn into seeders is determined by the total upload capacity $U(t) = C_s + \mu(\eta x(t) + y(t))$ of the system divided by the file size LR of the video file. Hence, we can describe the evolution of $x(t)$ and $y(t)$ as follows

$$\begin{cases} \frac{dx(t)}{dt} = \lambda(t) - \frac{U(t)}{LR}, \\ \frac{dy(t)}{dt} = \frac{U(t)}{LR} - \gamma y(t), \\ x(t_s) = \int_0^{t_s} \lambda(t)dt, y(t_s) = 0, \end{cases} \tag{5.4}$$

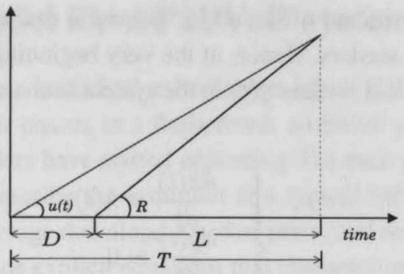


Figure 5.1: Average startup delay when $u(t) < R$. The download speed and the video playback rate are represented, respectively, in terms of pieces downloaded and played in time.

Consequently, the average download speed at time t is:

$$u(t) = \frac{C_s + \mu(\eta x(t) + y(t))}{x(t)}. \tag{5.5}$$

Startup delay and scalability

In principle, the system could scale indefinitely, if we let the average expected startup delay $D(t)$ for a peer that joins at time t be large enough. However in practice, as mentioned in the introduction of this thesis, the startup delay should be as small as possible, to ensure the playback to start nearly in real-time. On the other hand, the startup delay needs to be large enough to allow the user to watch the video without subsequent stall times, in order to meet the first QoS requirement for VoD.

Let $T(t) = LR/u(t)$ be the expected total download time at time t. Then the minimum startup delay that meets the QoS requirements is

$$D(t) = \max\left(0, \left(\frac{R}{u(t)} - 1\right) L\right). \tag{5.6}$$

That is, when $u(t) \geq R$ the startup delay is 0, otherwise it can be calculated as the waiting time necessary to have the expected download time be no longer than the duration of the video (Figure 5.1). This guarantees that, if current download speeds are maintained, the playback position is never greater than the download position, which in turn means that the user will perceive a good stream quality.

Note that, in this analysis, we have assumed that a newcomer starts downloading immediately at the system's average download speed. However, the initial speed of a newly joined peer is likely to be lower than that, due to the peer selection mechanisms adopted in real BitTorrent-based P2PVoD protocols (which require that a peer with no pieces needs to be optimistically unchoked by another peer first). Once a peer has collected a certain number of pieces, it will be able to barter with other peers and achieve a comparable download speed. Therefore, we can consider the value for $D(t)$ presented in Eq. (5.6) as a lower bound for the actual startup delay experienced by peers joining at time t.

From Eqs. (5.2) and (5.5) we can notice that the average download speed in the system, and therefore peers' startup delay, is a function of two sets of parameters. On one side, we have some parameters which depend on the peers behavior and cannot be controlled by the service providers, like μ, $\lambda(t)$, and γ. On the other side, $u(t)$ is related to some parameters which are specific to the system design, namely the efficiency of piece exchange η and the capacity C_s of the initial seeder. We will discuss them separatedly in the following paragraphs.

5.1.4 System design parameters

System design parameters are those parameters that can be tuned by the service providers to increase system scalability. For example, in order to make the best use of a downloader's bandwidth, the system might adopt an underlying P2P protocol that realizes an efficient piece selection strategy. In BitTorrent, this is achieved by means of a rarest-first strategy, which allows leechers to utilize, on average, 90% of their upload bandwidth [14]. However, to meet the QoS requirements for VoD, BitTorrent-based P2PVoD systems use a variant of this strategy, which is normally less efficient [13] (in any case, the piece exchange efficiency cannot exceed 1). On the other hand, the service provider can directly control the value of the initial seeder. Based on the average peer bandwidth, peer behavior, and efficiency of the P2P protocol in use, the service provider can set the appropriate value for C_s to support a certain number of peers at a given playback rate.

5.1.5 Peer behavior

Peer behavior can have a profound impact on the scalability of a P2P system. We will study this aspect by considering different peer arrival and departure patterns in our BitTorrent-based P2PVoD model.

More specifically, we assume that peers join the system at an exponentially decaying rate, as proposed by Guo *et al.* [40]: $\lambda(t) = \lambda_0 e^{-\frac{t}{\tau}}$, where λ_0 is the initial arrival rate at time 0 and τ is the decaying factor. The total number of peers joining the system is $\tau\lambda_0$. By varying the parameter τ, flashcrowds with different intensities can be simulated.

Analogously, the impact of different seeding behaviors can be studied by varying the value of seeder departure rate, γ, between ∞ (no seeding at all) and 0 (seeding for ever).

5.2 Numerical results

Based on our model, in this section we will analyze the scalability of a BitTorrent-based P2PVoD system during a flashcrowd.

5.2.1 The influence of system design parameters

We assume that, for the QoS requirements to be satisfied, the startup delay should be no larger than a certain value $D_{\max}$. If we express $D_{\max}$ as a fraction d of the duration L of the video, then we have the following QoS constraint

$$D(t) \leq D_{\max} = dL. \tag{5.7}$$

The average startup delay experienced by peers that join during the starting phase of a flashcrowd is

$$D(t) = \max\left(0, \left(\frac{Rx(t)}{\mu\eta x(t) + C_s} - 1\right) L\right) \quad (t \in (0, t_s)). \tag{5.8}$$

It is easy to see that the right most term in Eq. (5.8) is a strictly increasing function, since its derivative is always positive for $t \in (0, t_s)$. This means that the average startup delay is a monotonically increasing function in the starting phase, with upper bound in t_s. In the second phase of a flashcrowd, with peers becoming seeders, the average startup delay is likely to decrease (especially if seeders stay in the system long enough). Hence, we can consider the startup delay $D(t_s)$ experienced by peers joining at time t_s to be the worst case. Then, in order to ensure that Eq. (5.7) is satisfied, it would be sufficient to enforce the following condition

$$D(t_s) = \max\left(0, \left(\frac{Rx(t_s)}{\mu\eta x(t_s) + C_s} - 1\right) L\right) \leq dL,$$

which can alternatively be expressed as

$$x(t_s) \leq N = \begin{cases} \dfrac{(d+1)C_s}{R - (d+1)\mu\eta} & \text{if } R > (d+1)\mu\eta, \\ \infty & \text{if } R \leq (d+1)\mu\eta. \end{cases} \tag{5.9}$$

Eq. (5.9) illustrates the intrinsic relation between the fundamental system design parameters C_s and η and the maximum number of concurrent users N that can be supported at a playback rate R during a flashcrowd.

Figure 5.2 plots the sustainable flashcrowd size in relation to the capacity of the initial source and the efficiency of piece exchange for a scenario where $\mu = 1Mbps$, $R = 0.8\mu$,

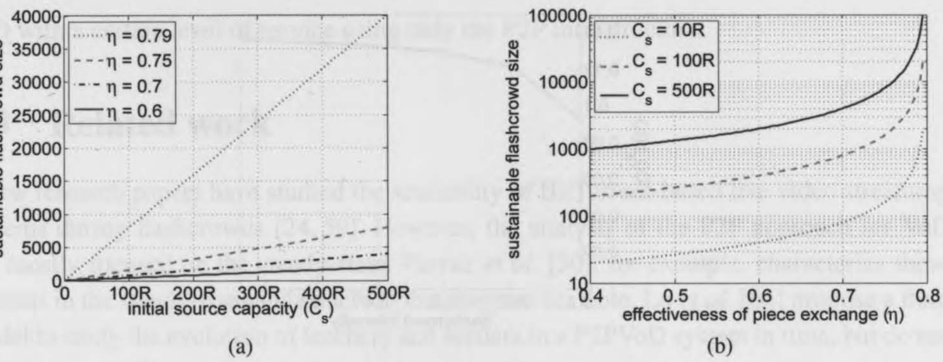


Figure 5.2: Sustainable flashcrowd size as a function of (a) the capacity of the initial seeder or source and (b) the efficiency of piece exchange η.

$d = 0$. We can observe that, when $R > (d+1)\mu\eta$, N grows linearly as C_s increases and hyperbolically as $R - (d+1)\mu\eta$ approaches 0 (which, in this case, happens when η moves towards 0.8). Hence, a natural solution to cheaply increase the system scale during a flashcrowd would be to have the playback rate R be nearly $(d+1)\mu\eta$. However, considering that η cannot be in any case larger than 1, this approach would either lead to excessively sacrificing the quality of the video (R small) or sacrificing the second QoS requirement for VoD (d high). From this, it is then evident that, for the system to be able to sustain a reasonable scale during flashcrowds, a significant amount of service capacity has to be initially provided by the source.

The importance of a high initial service capacity is also recognized by many BitTorrent communities, which seek for volunteers to seed newly released content (e.g. `http://eztv.it`). The content is normally uploaded to these peers before its official release date, which is per se an incentive to volunteer.

5.2.2 The influence of peer behavior

In this section, we will analyze the influence of peer behavior on the startup delay. Figure 5.3 illustrates the impact of flashcrowd intensity on the value of $D(t_s)$ for a scenario where $L = 3600s$, $\mu = 1Mbps$, $\eta = 0.7$, $R = 0.8\mu$, $C_s = 50R$, and the total number of peers joining is 15000. The flashcrowd intensity is defined as the fraction of peers joining the system within the first L seconds from the first peer arrival, out of all the peers that will join the system. For example, an intensity of 0.1 means that 10% of the arrivals occurs within the first L seconds from the first peer arrival.

For the same scenario, figure 5.4 shows the average startup delay of peers for different values of seeding times during a flashcrowd of intensity 0.5. As we can observe, if no peer stays in the system to seed, the average startup delay remains high for a long time. Then it

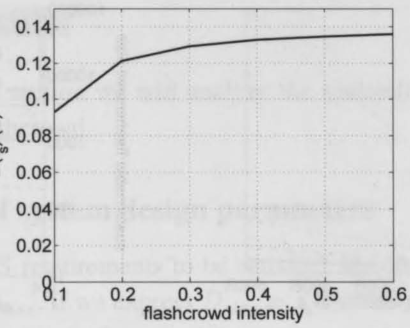


Figure 5.3: Evolution of $D(t_s)$ for different flashcrowd intensities.

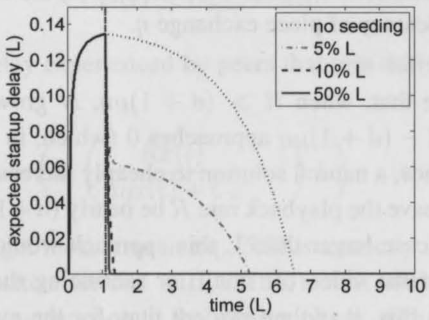


Figure 5.4: Evolution of startup delay for different seeding behaviors. The vertical dotted line represents the time t_s when the first seeders appear in the system.

starts to slowly decrease, only due to the fact that there are less and less new peers joining. However, when seeders contribute their bandwidth, even for as short a time as 5% of the length L of the video, the startup delay rapidly decreases after t_s. The situation gets better as peers start seeding for longer times, although with diminishing returns. In particular, we notice that a seeding time of 50% of the duration of the video L, is already sufficient to have the startup delay of newcomers drop to 0 almost immediately after t_s.

The above findings suggest that, once the system scale has reached a reasonable level, peer resources would be sufficient for the system to be self sustainable and scale even further. If we define $u_p(t)$ as the amount of $u(t)$ which only depends on peer contribution, then it is evident that when $u_p(t) \geq R$ the P2P system has become self-sustainable and the capacity provided by the initial seeder is not necessary anymore,

$$u_p(t) = \frac{\mu\eta x(t) + \mu y(t)}{x(t)} \geq R \Longrightarrow \frac{y(t)}{x(t)} \geq \frac{R - \mu\eta}{\mu}.$$

That is, a reasonable seeders-to-leechers ratio has to be reached, if we want to provide

VoD with a certain level of service using only the P2P infrastructure.

5.3 Related work

A few research papers have studied the scalability of BitTorrent-based live video streaming systems during flashcrowds [24, 59]. However, the analysis of the P2P approach for VoD has mostly focused on the steady-state. Parvez *et al.* [50], for example, characterize these systems in the steady-state and conclude that they are scalable. Lu *et al.* [64] propose a fluid model to study the evolution of leechers and seeders in a P2PVoD system in time, but do not offer any scalability analysis.

5.4 Conclusions

In this chapter, we have analyzed the scalability of BitTorrent-based P2PVoD systems during flashcrowds and found out that, in the initial phase of such phenomena, scalability is intrinsically related to the efficiency of the underlying P2P protocol and the initial service capacity. In particular, although it is important to make the P2P protocol as efficient as possible, we have showed that a large initial service capacity is necessary, in order to support an abrupt surge of joining peers. Our preliminary results also show that, once a sufficient seeders-to-leechers ratio is reached, the system enters a new phase in which it has become self-sustainable and the initial service capacity is no longer needed.

VoD with a certain level of service using only the P2P infrastructure.

5.3 Related work

A few research papers have studied the scalability of BitTorrent-based live video streaming systems during flashcrowds [24, 59]. However, the analysis of the P2P approach for VoD has mostly focused on the steady-state. Parvez *et al.* [50], for example, characterize these systems in the steady-state and conclude that they are scalable. Lu *et al.* [64] propose a fluid model to study the evolution of leechers and seeders in a P2P VoD system in time, but do not offer any scalability analysis.

5.4 Conclusions

In this chapter, we have analysed the scalability of BitTorrent-based P2P VoD systems during flashcrowds and found out that, in the initial phase of such phenomena, scalability is intrinsically related to the efficiency of the underlying P2P protocol and the initial service capacity. In particular, although it is important to make the P2P protocol as efficient as possible, we have showed that a large initial service capacity is necessary, in order to support an abrupt surge of joining peers. Our preliminary results also show that, once a sufficient seeders-to-leechers ratio is reached, the system enters a new phase in which it has become self-sustainable and the initial service capacity is no longer needed.

Chapter 6

Bandwidth allocation in BitTorrent-based P2PVoD systems during flashcrowds

In Chapter 5, we have studied how a flashcrowd affects the scalability of BitTorrent-based P2PVoD systems. From that analysis we have learned that the initial phase of a flashcrowd is the most critical one, due to the sharp increase of the demand and the shortage of seeders. To face this challenge, service providers need to supply enough initial service capacity, as well as deploy an underlying P2P protocol that can effectively use the limited bandwidth available, while at the same time enhance the QoS requirements necessary for VoD.

However, we have already observed several times throughout this thesis that the original BitTorrent protocol has not been designed designed for streaming, and most of its adaptations to VoD still lack of a number of elements necessary to enhance the QoS requirements of this application. For example, these proposals generally do not support any admission control mechanism, with the consequence that newcomers will continue being allowed to join the system, even when older peers struggle to maintaing a decent QoS. Similarly, not much work has been done to make seeding more effective. While acknowledging that Carlsson *et al.* [48] have proposed a new seeding policy in which seeders give priority to newcomers in order to reduce their startup delay, we believe that this is not a good choice in a flashcrowd scenario. In fact, while seeders favor newcomers, there is a big chance that a large fraction of older peers would still be struggling to maintain the necessary download speed to sustain the video playback rate. We argue that, under this circumstance, a system should try *first* to provide a good service to those users that are already in the system, and only *then* serve newcomers.

Therefore, in this chapter, we will investigate how bandwidth allocation during a flashcrowd can be made more effective in enhancing users' QoS. In particular, we focus on the role of the initial seeder, which represent the major bottleneck in this phase [14, 23].

Our contributions in this chapter are the following:

1. We devise an analytical model that describes how bandwidth should be allocated in a BitTorrent-based VoD system during a flashcrowd (Section 6.1).
2. Using this model, first we derive an upper bound to the number of newcomers that can be admitted in the system over time (Section 6.1). Subsequently, we show that a trade-off exists between having the initial seeder minimize the upload of pieces already injected recently and a high peer QoS (Section 6.2).
3. Finally, employing the insights of our analysis, we present and evaluate a class of distributed *flashcrowd-handling* algorithms to make bandwidth allocation during flashcrowds more effective as well as more QoS-oriented (Section 6.3).

6.1 Bandwidth allocation model for a BitTorrent-based P2PVoD system

In this section, we present a discrete-time model to describe the bandwidth allocation in a BitTorrent-based P2PVoD system during flashcrowd. Then, based on the fundamental QoS requirements for a VoD system, we derive an upper bound for the system scale over time.

6.1.1 Approach

In the previous chapter, we have used a fluid approximation of a Markov model to analyze the bandwidth distribution in a generic BitTorrent-based system at a high level of abstraction, in order to draw general conclusions on the factors that affect its scalability. In this chapter, instead, we seek to analyze how the bandwidth of each peer at each moment in time should be allocated, in order to satisfy the fundamental requirements for VoD. Since in BitTorrent-based systems peers usually re-evaluate their decisions every unchoke interval (see Chapter 1), we utilize a discrete-time approach to model their bandwidth allocation and, for simplicity, we assume that peers are synchronized.

6.1.2 Model description

We consider a BitTorrent-based P2PVoD system consisting of an *initial seeder* or *source*, i.e. a peer with a complete copy of the file, with upload capacity M, and a group of peers, with upload capacity μ, joining the system at a rate $\lambda(t)$. The video file shared in this system has playback rate R (Kbits/s), size F (Kbits) and is split into n pieces of equal size, allowing peers who are still in the process of downloading to serve the pieces they already have to others. The notation we use is shown in Table 6.1.

Table 6.1: Model parameters.

Notation	Definition
F	filesize (Kbits).
n	number of pieces the file is split into.
R	playback rate (Kbits/s).
N_0	number of sharers present in the system at the beginning of timeslot t_0.
M	initial seeders upload capacity (Kbits/s).
μ	peer upload capacity (Kbits/s).
r	per-slot capacity (Kbits/s).
$\nu_s = \lfloor M/r \rfloor$	number of upload slots opened by the initial seeder.
$\nu_p = \lfloor \mu/r \rfloor$	number of upload slots opened by each peer.
$\lambda(t)$	arrival rate of peers in the system.
$z(t_k)$	number of newcomers at timeslot t_k.
$\hat{z}(t_k)$	number of newcomers admitted in the system at the end of timeslot t_k.
$x(t_k)$	number of sharers at timeslot t_k.
$y(t_k)$	number of seeders at timeslot t_k.
$U(t_k)$	total upload capacity available at timeslot t_k (Kbits/s).

In the analysis, we assume that all peers utilize upload slots of identical size, i.e. the total number of upload slots ν_s offered by the initial seeder and the number of upload slots ν_p offered by a peer are defined as follows

$$\nu_s = \left\lfloor \frac{M}{r} \right\rfloor, \quad \nu_p = \left\lfloor \frac{\mu}{r} \right\rfloor,$$

where r is the *per-slot capacity*, which, without loss of generality, we assume to be a submultiple of the playback rate R. This is equivalent to the concept of *substreams* used in commercial P2P streaming systems (e.g. Coolstreaming [62]) and in P2P streaming literature (e.g. [22, 24]), where a video stream is divided into many substreams of equal size and nodes can download different substreams from different peers.

If each uploader has at least as many unchoked peers as upload slots, the minimum time needed to upload a piece is equal to

$$\tau_p = \frac{F}{nr},$$

with F/n being the size (in Kbits) of a piece.

We assume that time is discrete, with the size of each timeslot t_k being τ_p, i.e. $t_k = k\tau_p$ and $k \in \{0, 1, 2, \ldots i, \ldots\}$, and that the upload decisions are made at the beginning of each timeslot. Consequently, in each timeslot, a peer will upload to another peer exactly one piece.

In this analysis, we distinguish between two types of downloaders: *newcomers*, having no piece yet, and *sharers*, having at least one piece. We denote with $z(t_k)$, $x(t_k)$ and

$y(t_k)$, the number of newcomers, sharers, and seeders during timeslot t_k, respectively. In this notation, $y(t_k)$ excludes the initial seeder supplied by the video provider. Furthermore, we assume that, at timeslot t_0, there are already N_0 initial sharers in the system and that no peer leaves the system before its download is complete. Given this notation, the evolution of peers in the system can be described by means of the following set of discrete-time equations

$$\begin{cases} z(t_k) = z(t_{k-1}) - \hat{z}(t_{k-1}) + \lambda(t_{k-1}), \\ x(t_k) = x(t_{k-1}) + \hat{z}(t_{k-1}) - \hat{x}(t_{k-1}), \\ y(t_k) = y(t_{k-1}) + \hat{x}(t_{k-1}) - \gamma(t_{k-1}), \\ z(t_0) = 0, x(t_0) = N_0, y(t_0) = 0, \end{cases}$$

where $\lambda(t_{k-1})$ is the number of peers who joined within timeslot t_{k-1}, $\hat{z}(t_{k-1})$ is the number of newcomers that turned into sharers at the end of timeslot t_{k-1} (i.e. they were *admitted* in the system), $\hat{x}(t_{k-1})$ is the number of sharers that turned into seeders at the end of timeslot t_{k-1}, and $\gamma(t_{k-1})$ is the number of seeders who have left at the end of timeslot t_{k-1}.

The total bandwidth available during a timeslot t_k is given by the sum of the contributions of all the sharing peers (seeders and sharers) available at the beginning of timeslot t_k, i.e.

$$U(t_k) = M + \mu x(t_k) + \mu y(t_k). \tag{6.1}$$

6.1.3 Upper bound for the system scale in time

Even for a scalable system, only a limited number of newcomers can be admitted at each timeslot. This is due to the fact that newcomers consume bandwidth without providing any bandwidth in return, until they complete the download of their first piece. In this section, we will derive an upper bound for the number of newcomers that can be admitted in the system at each timeslot, assuming that all the bandwidth $U(t_k)$ available at a certain timeslot t_k is fully utilized. We proceed by first reserving the necessary bandwidth for the sharers to satisfy the first QoS requirement for VoD. Then, based on the remaining bandwidth, we calculate the number of newcomers that can be admitted in timeslot t_k.

Reserving the necessary bandwidth for the sharers

Recall from the introduction of this thesis that, when bandwidth is scarce, the primary goal of a VoD system is to maximize the number of viewers that maintain a smooth playback continuity. For the sake of simplicity, here we assume that all sharers have started playback. It follows that a certain amount of bandwidth $U_x(t_k)$ needs to be reserved for the sharers at timeslot t_k. This bandwidth should be such that each sharer can maintain a download speed of at least R. Hence we have

$$U_x(t_k) = Rx(t_k). \tag{6.2}$$

Admitting the newcomers and upper bound

After having reserved the necessary bandwidth for the sharers, the remaining bandwidth (if any), can be used to admit newcomers in the system. To this end, we find the following upper bound for the number of newcomers that can be admitted during timeslot t_k.

Lemma 1. *For a BitTorrent-based P2PVoD system with playback rate R and average peer upload capacity $\mu \geq R$, the number of newcomers $\hat{z}(t_k)$ that can be admitted during timeslot t_k has the following upper bound*

$$\hat{z}(t_k) \leq \frac{M + \mu y(t_k) + (\mu - R)x(t_k)}{r}. \tag{6.3}$$

Proof. Taking Eq. (6.2) into account, the bandwidth available for newcomers at timeslot t_k is at most $U(t_k) - U_x(t_k) = U(t_k) - Rx(t_k) = M + \mu y(t_k) + (\mu - R)x(t_k)$. Since the capacity of a peer upload slot is r, Eq. (6.3) follows. □

From Lemma 1, it is clear that, at the beginning of a huge flashcrowd, when there are only few or no seeders (besides those supplied by the service provider) and few sharers who can only provide a limited fraction of bandwidth to newcomers, the system can only admit a small amount of newcomers per timeslot. In this case, it is impossible to avoid newcomers experience longer startup delays, as we will show with our experiments in Section 6.4.

6.2 Initial seeder's piece allocation analysis

In this section, we will study the piece allocation strategy of the initial seeder in a BitTorrent-based P2PVoD system during a flashcrowd. We focus on the initial seeder because its piece allocation strategy is a crucial aspect during flashcrowd, being the seeder the only interesting peer in that phase (all other peers have few or no pieces yet). Furthermore, studying the piece allocation of each single peer in the system is unpractical, since the complete bandwidth allocation problem in a BitTorrent-based system has been proven to be NP-hard [23].

We proceed by first defining the relevant features of the BitTorrent-based P2PVoD protocol we consider and introduce a useful concept to understand the flow of data from the seeder to the peers. Then, we analyze in detail the seeder's piece allocation.

6.2.1 Protocol features

For the purpose our analysis, we assume that, if there is more than one initial seeder, they coordinate their behaviors so that we can consider them as one identity. The total capacity

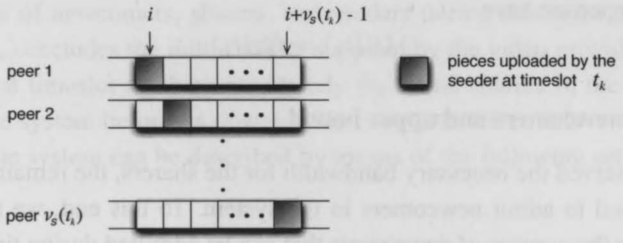


Figure 6.1: Seeder's piece allocation at timeslot t_k.

provided by the initial seeder at each timeslot t_k is denoted by $\nu_s(t_k)$. We assume that the initial seeder knows

1. the arrival rate $\lambda(t_k)$, the leaving rate $\gamma(t_k)$; and

2. the last piece it has sent to the sharers.

Given our first assumption, the seeder always knows the exact number of peers in the system at any moment in time.

In the following, we describe the piece allocation and the piece download schemes adopted by this seeder and by the downloaders, respectively.

Seeder's piece allocation

Having denoted $\nu_s(t_k)$ as the total number of upload slots provided by the seeder at timeslot t_k, we assume the seeder to adopt the following behavior

3) allocate each of these slots to a different peer;

4) unchoke, at each timeslot, the oldest $\nu_s(t_k)$ peers in the system; and

5) unless otherwise specified, at each timeslot t_k, upload pieces from i to $i + \nu_s(t_k) - 1$, where $i - 1$ is the piece with highest index uploaded at the previous timeslot t_{k-1}.

Strategy 3) reflects the idea of serving as many peers as possible. Strategy 4) is justified by the fact that younger peers, having a lower level of progress than older peers, can download their needed pieces from older peers, while the oldest peers can obtain the pieces they need only from the seeder. As a consequence of our strategy 5), each of the $\nu_s(t_k)$ peers unchoked by the seeder will receive a different piece, as illustrated in Figure 6.1. We note that this scheme increases the bartering abilities among peers, hence allowing a high peer bandwidth utilization.

Piece download scheme

According to QoS requirement 1 for VoD, a sharer should keep an average download rate of at least R, in order to maintain a good stream continuity. However, pieces needed by peers cannot always be downloaded in a strict sequential order, otherwise the bartering abilities among nodes are hampered. To avoid this scenario, we assume that

6) each peer defines a download buffer $\mathcal{B}_\alpha$ of size B which includes the pieces $[i, i+B-1]$, where $i = \alpha B$, with $\alpha \in \{0, 1, 2, \dots, \lfloor \frac{n}{B} \rfloor\}$.

Once all the pieces in the current buffer $\mathcal{B}_\alpha$ are downloaded, a peer determines the next buffer $\mathcal{B}_{\alpha+1} = \mathcal{B}_\alpha + B = [i+B, i+2B-1]$. Although pieces from outside the buffer can be downloaded, it is necessary to enforce the buffer filling rate to be at least R, in order to satisfy QoS requirement 1. Even if the schemes used in practice are more practically convenient (with the buffer being implemented as a sliding window following the playback position or the first missing piece of the file [26, 51, 63]), a static buffer makes the computation of its filling rate easier, which we will use in the analysis in Section 6.2.3.

6.2.2 Organized view of an overlay mesh

To understand the flow of data from the seeder to the downloading peers, we use the concept of *organized view of an overlay mesh*, originally proposed for P2P live streaming systems [49]. In this view, downloaders are grouped into *levels* based on their shortest distance from the seeder through an overlay, as shown in Figure 6.2. The set of peers on level i is denoted by L_i. L_1 peers are directly served by the seeder, L_2 peers are served by L_1 peers, and so on. The connections from L_i peers to L_{i+1} peers are called *diffusion connections*, since they are used for diffusing new pieces through the overlay. On the other hand, the connections from L_i peers to L_j peers, where $j \leq i$, are used to exchange missing content through longer paths in the overlay (i.e. swarming). We call these connections *swarming connections*.

6.2.3 Piece replication at the seeder

A seeder might decide to only upload pieces not yet present in the overlay or upload again some pieces already injected recently (a behavior which we term *piece replication*).

As observed earlier, a system where the seeder adopts the first strategy allows a higher peer bandwidth utilization.

On the other hand, a higher piece replication at the seeder, when properly implemented, allows a faster diffusion of pieces in the system and increases the system scale. In fact, if the seeder serves to the peers the pieces they need in the immediate future (rather than new, far-away ones), then these peers have a lower chance of missing a piece before its playback

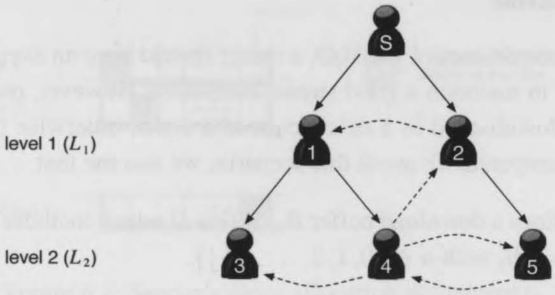


Figure 6.2: Organized view of an overlay mesh relative to a BitTorrent-based P2PVoD system. Solid arrows and dashed arrows represents diffusion connections and swarming connections, respectively.

deadline. Furthermore, since these nodes obtain some of the needed pieces directly from the seeders, they need to obtain fewer pieces from their neighbors, which in turn can then utilize a higher fraction of their bandwidth to serve newcomers, thereby reducing startup delays and increasing the system scale.

However, even if the seeder decides to upload again some pieces already present in the system, a certain minimum number of new pieces has to be injected at each timeslot, to allow older peers maintain a download speed of at least R.

Hence, a balance is necessary between injecting enough new pieces in the system and serving pieces needed right away. We study this issue using the concept of *seeder replication factor* F_k at timeslot t_k, which we define as the fraction of replicated pieces over the total number of pieces that a seeder allocates in that timeslot. Thus, a seeder replication factor of a/b, for a seeder with b upload slots, means that a of the allocated pieces will be a replica while the other $b-a$ will be pieces not yet present in the system. In the following, we show how to determine an upper and a lower bound for the seeder replication factor F_k.

Theorem 1. *Let a BitTorrent-based P2PVoD system with playback rate R consist, at the beginning of timeslot t_k, of a seeder with upload capacity $r\nu_s \geq R$ and at least $x(t_k) \geq \nu_s$ sharers with upload capacity $r\nu_p \geq R$. Then, the maximum value of the seeder replication factor F_k guaranteeing that, independently from previous upload allocations, the sharers keep a buffer filling rate of R at timeslot t_{k+1}, is*

$$\max F_k = \frac{\nu_s - \frac{R}{r}}{\nu_s}. \tag{6.4}$$

Proof. Let us assume that $F_k > \frac{\nu_s - \frac{R}{r}}{\nu_s}$. This means that the number of replicated pieces uploaded by the seeder at timeslot t_k is $C(t_k) \geq \nu_s - \frac{R}{r}$, which in turn means that the seeder has injected at most $D(t_k) < \frac{R}{r}$ new pieces. Now, let us assume that previous upload allocations are such that, by the end of timeslot t_k, all L_1 peers complete the download of all pieces until (and including) piece i, where i is the piece with highest index uploaded by the seeder at timeslot t_{k-1}. Consequently, at timeslot t_{k+1}, the L_1 sharers can complete, at most, the download of the $D(t_k) < \frac{R}{r}$ new pieces injected by the seeder at timeslot t_k, which means that their average download rate can be at most $D(t_k)r < R$. Hence, we have demonstrated that there exist at least one scenario in which the sharers will not be able to maintain a piece buffer filling rate of at least R when $F_k > \frac{\nu_s - \frac{R}{r}}{\nu_s}$. On the other hand, when $F_k \leq \frac{\nu_s - \frac{R}{r}}{\nu_s}$, then $D(t_k) \geq \frac{R}{r}$, which implies that the sharers can potentially reach an average download rate of $D(t_k)r \geq R$. □

As we will see later on in this chapter (Section 6.4), the upper bound for the seeder replication factor is also the value yielding the best playback continuity. In fact, on the one hand this value allows enough replication to limit the number of pieces peers miss, and on the other hand it guarantees that the oldest peers have enough new pieces to keep an average download rate as high as the playback rate.

Theorem 2. *Let a BitTorrent-based P2PVoD system with playback rate R consist, at the beginning of timeslot t_k, of a seeder with upload capacity $r\nu_s \geq R$, $x(t_k) \geq \nu_s$ sharers with upload capacity $r\nu_p \geq R$ and $z(t_k)$ newcomers. Then, the minimum value of the seeder replication factor F_k at timeslot t_k necessary to maximize the number of newcomers to be admitted, while still guaranteeing the sharers a buffer filling rate of R, is*

$$\min F_k = \begin{cases} 0 & \text{if } z(t_k) \leq Z_1(t_k), \\ \frac{z(t_k) - \frac{R}{r} - Kx(t_k)}{\nu_s - 1} & \text{if } Z_1(t_k) < z(t_k) < Z_2(t_k), \\ \dfrac{\nu_s - \frac{R}{r}}{\nu_s - 1} & \text{if } z(t_k) \geq Z_2(t_k), \end{cases}$$

where

$$K = \nu_p - \frac{R}{r}, \tag{6.5}$$

$$Z_1(t_k) = \frac{R}{r} + Kx(t_k), \tag{6.6}$$

$$Z_2(t_k) = \nu_s + Kx(t_k). \tag{6.7}$$

In order to prove Theorem 2, we need to introduce the following lemma

Lemma 2. *Given a BitTorrent-based P2PVoD system under the same conditions as in Theorem 2, if the seeder does not replicate, then its average contribution of pieces within the buffer of each L_1 sharer is*

$$\frac{\nu_s \frac{R}{r} + Kx(t_k) - \min\{Z_2(t_k), z(t_k)\}}{\nu_s - 1}$$

pieces per timeslot, where K and $Z_2(t_k)$ are defined in Eq. (6.5) and (6.7), respectively.

For the proof of Lemma 2 we refer the reader to Appendix B.

Proof of Theorem 2. When the sharers are able to serve all the newcomers (with at least one piece each), as well as complete the download of the $\frac{R}{r}$ pieces within their respective current buffers necessary to maintain a good stream continuity (QoS requirement 1), utilizing only their aggregate bandwidth, then the seeder does not need to replicate and can inject new pieces into the system.

Specifically, if the sharers serve the newcomers, they will be having a total of $X_1(t_k) = \nu_p x(t_k) - Z_m(t_k)$ slots left, being $\nu_p x(t_k)$ the total number of slots offered by the sharers, $Z_m(t_k) := \min\{Z_2(t_k), z(t_k)\}$, and $Z_2(t_k)$ the maximum number of newcomers that can be served at this timeslot (as derived from Lemma 1 applied to this case). Hence, it holds that $X_1(t_k) \geq \nu_p x(t_k) - Z_2(t_k) = \frac{R}{r}x(t_k) - \nu_s$. Of these slots, $X_2(t_k) = (x(t_k) - \nu_s)\frac{R}{r}$ can be used to provide the $\frac{R}{r}$ needed pieces to the L_j sharers ($j > 1$), which are $x(t_k) - \nu_s$ in total. Consequently, the number of slots from the sharers available for the L_1 peers are

$$X_s(t_k) = X_1(t_k) - X_2(t_k) = \nu_s \frac{R}{r} + Kx(t_k) - Z_m(t_k). \tag{6.8}$$

Alternatively, $X_s(t_k)$ can be considered as the maximum number of pieces that L_1 peers can receive through swarming. Now, the piece replication at the seeder should be such to allow each of these peers complete the download of the $\frac{R}{r}$ pieces within their current buffers at the end of timeslot t_k. This makes a total of $R\nu_s/r$ needed pieces for all the L_1 peers. Of these pieces, $X_s(t_k)$ can be obtained from swarming, and, by Lemma 2, at most other

$$\frac{X_s(t_k)}{\nu_s - 1}$$

pieces are provided by the seeder (when not taking replication into account). Hence, the total amount of needed pieces minus those provided through swarming and by the non-replicating activity of the seeder corresponds to the minimum number of pieces that the seeder needs to replicate at timeslot t_k

$$\begin{aligned} C(t_k) &= \max\left\{0, \frac{R}{r}\nu_s - X_s(t_k) - \frac{X_s(t_k)}{\nu_s - 1}\right\} = \\ &= \max\left\{0, \frac{\nu_s}{\nu_s - 1}\left(Z_m(t_k) - \frac{R}{r} - Kx(t_k)\right)\right\}. \end{aligned}$$

Hence, the minimum replication factor F_k is

$$F_k = \frac{C(t_k)}{\nu_s} = \max\left\{0, \frac{Z_m(t_k) - \frac{R}{r} - Kx(t_k)}{\nu_s - 1}\right\}. \tag{6.9}$$

From Eq. (6.9) we notice that, when $z(t_k) \leq Z_1(t_k) = \frac{R}{r} + Kx(t_k)$, the seeder does not need to perform any replication. Furthermore, we observe that, when $z(t_k) \geq Z_2(t_k) = \nu_s + Kx(t_k)$, the minimum seeder replication factor equals to $\frac{\nu_s - \frac{R}{r}}{\nu_s - 1}$, which completes our proof. $\square$

6.3 Algorithms for flashcrowds

In this section, we present a class of *flashcrowd-handling* algorithms that use the insights gained by our analysis to make the bandwidth allocation in BitTorrent-based P2PVoD systems during flashcrowds more effective in enhancing the QoS of peers. First, we explore some methods to allow a peer to detect whether the system is under a flashcrowd. Subsequently, we describe our algorithms in detail.

6.3.1 Flashcrowd detection

Ideally, the bandwidth allocation of each peer at every moment in time should rely on some global knowledge of the state of the system at that time (e.g. total number of peers, number of newcomers, current download progress of all peers, etc.). However, providing all the nodes in the system with this kind of information is not feasible in practice. Furthermore, the bandwidth allocation problem in BitTorrent-based systems has been shown to be NP-hard [23].

Hence, in this chapter, we will use a heuristic approach where each peer individually considers the system to be either in "normal state" or "under flashcrowd". Depending on which state the peer assumes the system is in, it will utilize a different bandwidth allocation algorithm. To implement this mechanism, peers need some way to detect the occurrence of a flashcrowd. Based on a peer's local knowledge, a natural choice to identify a flashcrowd would be to measure the following:

(a) **Increase in the perceived number of newcomers**. A peer can track the number of newcomers that connect to it by checking the pieces owned by its neighbors.

However, when the peerlist provided by the tracker contains a constant number of nodes, this is not a good metric for detecting a flashcrowd, as its accuracy decreases with the size of the system (see Figure 6.3(a), obtained running the BitTorrent-based P2PVoD protocol proposed in [63] under the settings described in Section 6.4.1 of this chapter). On the other hand, since the tracker provides each peer with a random subset of the nodes, we can assume that each peer encounters a random and therefore representative selection of other peers and we can measure the following:

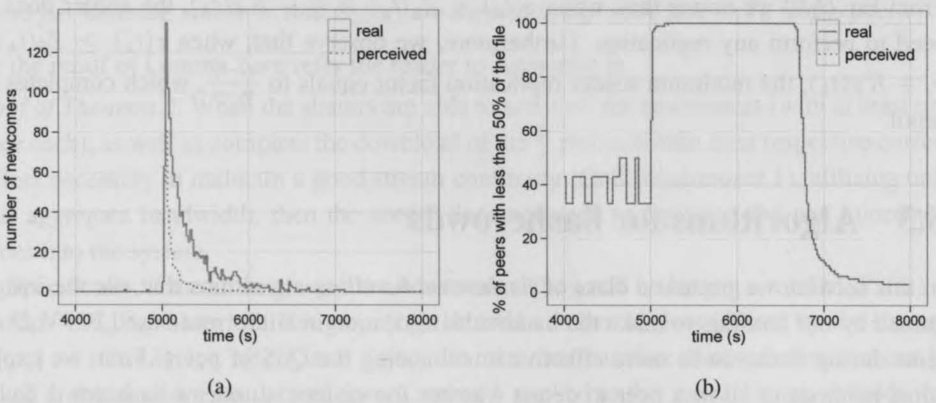


Figure 6.3: Comparison between the real value and the value perceived by a peer for a flashcrowd detection based on (a) the number of newcomers, and (b) the percentage of peers with less than 50% of the file.

(b) **Fraction of neighbors having less than 50% of the file**. Esposito *et al.* [23] observed that in the BitTorrent file-sharing system, the average file completion level of peers during a flashcrowd is *biased towards less than half of the file*, i.e. there are many more peers with few pieces than peers with many pieces.

Figure 6.3(b) illustrates the performance of both methods for a scenario where peers join at rate $\lambda(t) = \lambda_0 e^{-\frac{t}{\tau}}$ with $\lambda_0 = 5$ and $\tau = 1500$ from time $t_0 = 5000$s in a system with a seeder and $N_0 = 7$ initial peers. As we can see, our experiments corroborate the findings of Esposito *et al.* [23]. Furthermore, our experiments also show that the difference between the real value of peers having less than 50% of the file and the value perceived by a peer (i.e. based on the nodes in its neighborhood) is barely visible. These results confirm that (b) represents a good metric for a peer to detect a flashcrowd solely based on its local information. Furthermore, using this method, peers can estimate the end of a flashcrowd as well, by checking when the fraction of neighbors with more than half of the file becomes higher than that of peers with less than half of the file. Hence, in our experiments, we will use this method to detect a flashcrowd.

Once detected a flashcrowd, a peer also needs to know whether the flashcrowd is negatively affecting the system performance. In fact, the same flashcrowd might have a different impact on the system performance depending on how many peers are already there when the flashcrowd hits. Therefore, each sharer periodically measures its download performance and checks whether it is enough to meet the QoS requirement 1 for VoD. On the other hand, a seeder does not download data nor it can trust information received by other peers (as they might lie). Therefore, a seeder will only use the flashcrowd detection method to activate its flashcrowd-handling algorithm.

6.3.2 Flashcrowd-handling algorithms

In our proposal, a peer runs a certain default algorithm until it detects both a flashcrowd and (in the case of a sharer) that its own performance is low. When this happens, it will switch to a flashcrowd-handling algorithm. More specifically, a peer will assume the system to be under flashcrowd once the number of its neighbors having less than 50% of the file is gone above a certain threshold T. If the peer is a seeder, this is enough for it to activate its flashcowd-handling algorithm. If it is a sharer, it will only activate its flashcrowd-handling algorithm if its *sequential progress*[1] is below the playback rate R. The sequential progress is a good metric for a real-time check of the preservation of a peer's stream continuity. Furthermore, it has the advantage of being agnostic with respect to the piece selection policy adopted by the underlying BitTorrent-based P2PVoD protocol.

In the following we present our flashcrowd-handling algorithms for the sharers and the seeder respectively, which are derived from the insights gained from our analysis in Sections 6.1 and 6.2.3.

For the sharers

Since the priority of a BitTorrent-based P2PVoD system is to maximize the number of sharers that keep a smooth playback continuity, newcomers should only be allowed in the system if there is enough bandwidth available for them, after the necessary bandwidth for all the current sharers has been reserved (Lemma 1). Peers, however, do not have (nor it is reasonable for them to have) global knowledge of what is happening in the system at a certain instant in time (how many sharers and newcomers there are, how many newcomers have been already unchoked, etc.). Therefore, we propose that, when a sharer is running the flashcrowd-handling algorithm, it will choke *all* the newcomers and keep them choked until it switches back to the default algorithm. Newcomers might still be unchoked by peers who are not running the flashcrowd-handling algorithms, if any. This strategy avoids wasting bandwidth to admit newcomers, when existing peers struggle to keep a smooth playback continuity.

For the seeder

As we have observed in Section 6.2.3, the seeder's behavior is crucial during a flashcrowd. Similarly to sharers, seeders choke all newcomers when they are running the flashcrowd-handling algorithm. Furthermore, based on the observation from our analysis in Section 6.2 that older peers can only get their pieces from the seeder and given that the competition for the seeder is higher during flashcrowd, we designed our flashcrowd-handling algorithm to

[1]a peer's sequential progress is defined as the rate at which the index of the first missing piece in the file grows [20]

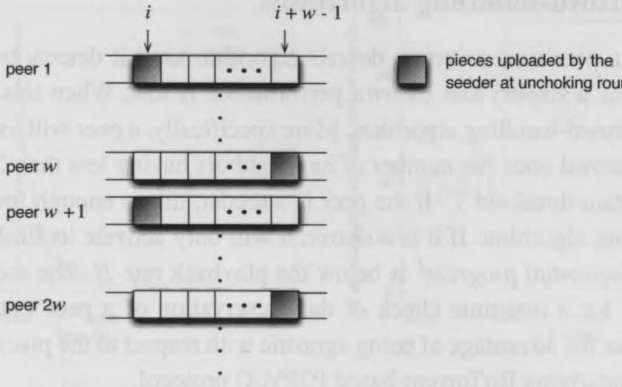


Figure 6.4: Seeder's piece allocation with $g = \frac{\nu_s}{w}$ groups of peers.

have the seeder keep the oldest peers always unchoked. Then, we have implemented two different classes of seeding behavior as described below.

1) **Passive seeding (FH/PS)**: the seeder does not directly decide which pieces it will upload and the decision is left to the requesting peers.

With this strategy we will evaluate the effectiveness during flashcrowd of the piece selection strategy employed by peers.

2) **Active seeding (FH/AS)**: the seeder decides which piece to send to each requesting peer.

This second strategy allows us to evaluate the impact of different replication factors. For what concerns the pieces to replicate, we have chosen a proportional approach, in order to reduce the skewness of piece rarity: all pieces are replicated the same number of times. More specifically, given a replication factor F_k at seeder's unchoking round k, the number of new pieces the seeder injects in the system is $w = (1 - F_k)\nu_s$, ν_s being the number of upload slots of the seeder. Then, the number of peers directly unchoked by the seeder is divided in $g = \frac{\nu_s}{w}$ groups of size w each and peers within each group are assigned pieces from i to $i + w - 1$, where $i - 1$ is the piece with highest index uploaded by the seeder in the previous round. For an illustration see Figure 6.4.

For what concerns the coordination of multiple seeders, we make the following observations. Firstly, in a flashcrowd scenario, typically there are only one or a few seeders in the system, i.e. the content injectors. In the case of only one seeder, no coordination is needed, while in the case of few seeders, the coordination overhead is not very high. In fact, since the seeders do not unchoke new nodes until some of the currently unchoked peers leave, and since the behavior of the seeders is deterministic, they need to coordinate only at the beginning, when getting their first connections, and every time an unchoked peer leaves.

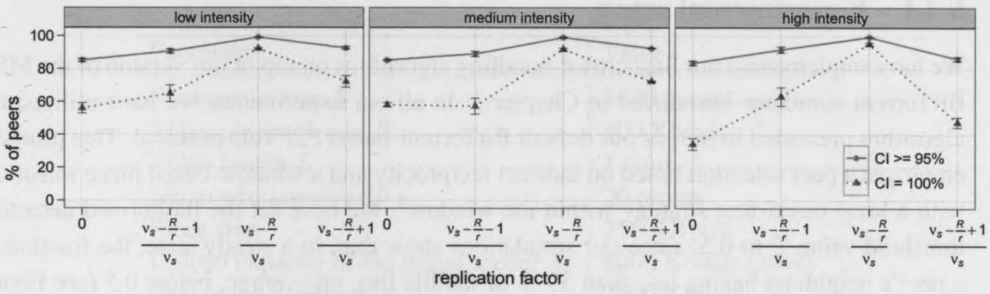


Figure 6.5: Percentage of peers experiencing perfect playback continuity (CI = 100%) and good playback continuity (CI $\geq$ 95%) for different replication factors and flashcrowd intensities.

Secondly, the creation of new seeders at a later stage, as a consequence of peers completing their downloads and remaining in the system to seed, indicates per se that more and more bandwidth becomes available in the system. At that stage, the system would likely be already able to deliver a reasonably good service even for short seeding times and no flashcrowd-handling mechanism in place [39]. Thus, the coordination between these newly created seeders and the initial seeder(s) can be avoided.

Finally, we note that, even if a seeder activates its flashcrowd-handling algorithm in a flashcrowd that would not affect the system very seriously, peer QoS will not degrade. In fact, although the seeder does not unchoke any newcomers, they will still be unchoked by many other sharers in the system. Hence the impact on newcomers' startup delay will be minimal. Regarding the fact that older peers always remain unchoked, we believe that this is not a problem either. In fact, as pointed out earlier, older peers can only obtain their pieces from the seeders and, if they do not need to compete with other peers for the seeder's slots, they are likely to experience better QoS, and hence able to serve more peers with a lower level of progress.

6.4 Evaluation

In this section, we evaluate our proposed flashcrowd-handling algorithms by means of simulations. First, we introduce the details of the experimental setup, the evaluation metrics, and we describe the different flashcrowd scenarios used. Then, we present and analyze the simulation results. In the results, we also included a case with heterogeneous peers, in which a flashcrowd handling algorithm is compared with a strategy favoring fast peers.

6.4.1 Experimental setup

We have implemented our flashcrowd-handling algorithms on top of our version of the MSR BitTorrent simulator introduced in Chapter 3. In all our experiments, we have utilized the algorithm presented in [63] as our default BitTorrent-based P2PVoD protocol. This protocol employes a peer selection based on indirect reciprocity and a window-based piece selection, with a local rarest-first strategy within the window[2]. We have set the flashcrowd detection threshold value T to 0.5, since our simulations show that, in a steady state, the fraction of a peer's neighbors having less than 50 % of the file lies, on average, below 0.5 (see Figure 6.3(b)).

The settings for our experiments are shown in Table 6.2. The system is initially empty, until a flashcrowd of N peers starts joining. In our simulations, we have utilized both an exponentially decreasing arrival rate $\lambda(t) = \lambda_0 e^{-\frac{t}{\tau}}$, and an arrival rate with N peers joining altogether at time $t_0 = 0$. The simulation stops after the last peer completes its download. In our experiments, we have assumed the worst case scenario of peers leaving immediately after their download is complete. On the other hand, the initial seeder never leaves the system.

Finally, to decide when playback can safely commence, the method introduced in [20] is used. Specifically, a peer will start playback only when it has obtained all the pieces in the initial buffer and its current sequential progress is such that, if maintained, the download of the file will be completed before playback ends.

Each simulation run is executed 10 times and then average values and confidence intervals (with confidence level of 95%) for the metrics introduced below are computed.

6.4.2 Evaluation metrics

To evaluate how well our solutions meet the QoS requirements for VoD, we have utilized again the continuity index (CI) and the startup delay, defined in Chapter 3.

6.4.3 Scenarios

In our simulations, we have considered three scenarios characterized by three different flashcrowd intensities:

- *low intensity*: exponentially decreasing arrival rate $\lambda(t) = \lambda_0 e^{-\frac{t}{\tau}}$, with $\lambda_0 = 5$ and $\tau = \frac{N}{\lambda_0} = 300$;
- *medium intensity*: exponentially decreasing arrival rate $\lambda(t) = \lambda_0 e^{-\frac{t}{\tau}}$, with $\lambda_0 = 10$ and $\tau = \frac{N}{\lambda_0} = 150$;

[2]The VoD protocol presented in [63] employes an adaptive mechanism to increase a peer's window size if that peer is experiencing a good QoS. In this way, peers bartering abilities are increased when the conditions are favorable. The parameter "initial window size B" reported in Table 6.2 represents the default initial size of each peer's window.

Table 6.2: Simulation Settings

Parameter	Value
Flashcrowd size N	1500 peers
Video playback rate R	800 Kbits/s
Video length L	1 hour
Initial window size B	20 pieces
Piece size	256 KBytes
Upload capacity of the initial seeder M	8000 Kbits/s ($10R$)
Peer upload capacity μ	1000 Kbits/s
Per-slot capacity r	200 Kbits/s
Flashcrowd detection threshold T	0.5

- *high intensity*: N peers joining altogether at time t_0.

6.4.4 Results

We will first analyze the effect of different replication factors F_k over the performance of our flashcrowd-handling algorithms and then we will evaluate the performance of our flashcrowd-handling algorithms.

The effect of different replication factors

Figure 6.5 shows the percentage of peers experiencing perfect (CI $=100\%$) and good (CI $\geq 95\%$) playback continuity for flashcrowd-handling algorithms with active seeding having different replication factors under the three simulated scenarios. As we can see, no replication (i.e. $F_k = 0$) is not an optimal strategy, as it always causes a considerable amount of peers experience poor stream continuity (in the case of flashcrowd of high intensity, for example, only 36% of peers experience perfect playback continuity). On the other hand, when the seeder performs replication, the playback continuity index of peers increases. In fact, as we observed in Section 6.2, a higher piece replication at the seeder decreases the chance of peers missing pieces. However, we have also showed that the seeder replication must not be too high: the seeder needs to inject new pieces at a rate of at least R (which means a replication factor $F_k \leq \frac{\nu_s - \frac{R}{r}}{\nu_s}$), in order to make sure that its unchoked peers keep a download rate of at least R necessary to meet the first QoS requirement for VoD. Indeed, from Figure 6.5 we can observe that, in all scenarios, the playback continuity index improves as the replication factor grows until it reaches the limit $F_k = \frac{\nu_s - \frac{R}{r}}{\nu_s}$. When $F_k > \frac{\nu_s - \frac{R}{r}}{\nu_s}$, the playback continuity index starts degrading again.

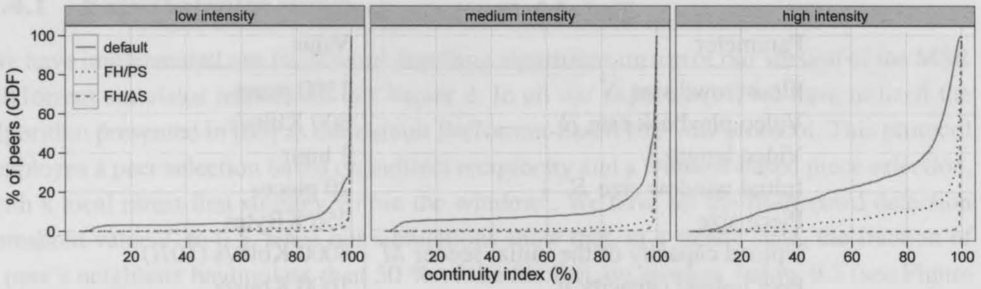


Figure 6.6: CDFs of peer's playback continuity index in a system hit by flashcrowds of different intensities. The active seeding algorithm uses the maximum replication factor according to Eq. (6.4).

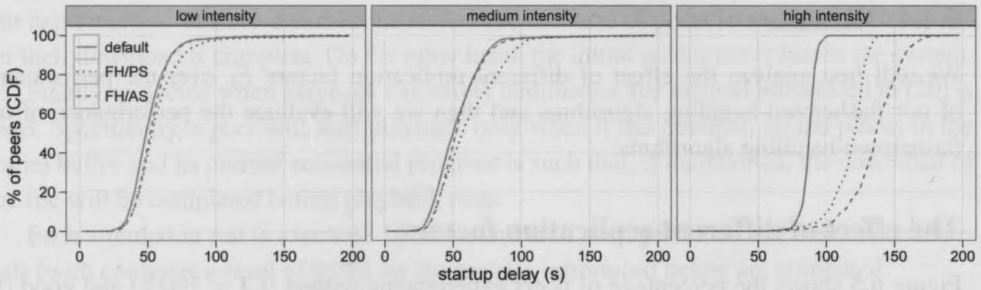


Figure 6.7: CDFs of peer's startup delays. The notations and the setups as the same as for Figure 6.6.

Default algorithm vs flashcrowd-handling algorithms

Figure 6.6 shows the CDF of peer playback continuity index for the default BitTorrent-based P2PVoD algorithm and our flashcrowd-handling algorithms under the three simulated scenarios. The algorithm with active seeding has replication factor $F_k = \frac{\nu_s - \frac{R}{r}}{\nu_s}$, which, as shown by the previously presented results, is the one that maximizes QoS requirement 1. As we can observe, the flashcrowd-handling algorithm with active seeding (FH/AS) consistently outperforms the other ones, with never more than 10% of the peers receiving a playback continuity index below 100%. By contrast, in the case of flashcrowd with high intensity, the default algorithm is not able to provide *any* peer with a CI of 100%. Furthermore, we can notice that, while the performance of the other two algorithms degrades with more intense flashcrowds, that of FH/AS stays constant. Finally, we note that the flashcrowd-handling algorithm with passive seeding (FH/PS) works relatively well for not too intense flashcrowds, but suffers performance degradation with a very intense flashcrowd. This is due to the fact that the seeder replication factor is controlled by the peers, which do not coordinate their piece requests among each other. The local rarest-first strategy used by each peer to select a piece to download is supposed to smoothen this effect. However, since its effectiveness only builds up once a peer has been in the system for some time, it is less powerful when the system is under a heavy flashcrowd.

Regarding the startup delay (Figure 6.7), we can make the following observations. First we note that, for a flashcrowd with low or medium intensity, FH/AS is able to maintain a relatively low startup delay for all peers (comparable to that of the default algorithm). This is a sign that an adequate replication of pieces at the seeder results in satisfying both QoS requirements, when possible. On the other hand, FH/AS significantly increases the startup delay of peers in the scenario of heavy flashcrowd. This is an experimental validation of what is stated in Lemma 1, the bandwidth available at the beginning of the flashcrowd is not enough to serve all the joining peers, which, consequently, will experience longer startup delays.

We have simulated each of the three flashcrowd scenarios 10 times and found out that the behavior of the different algorithms is very stable, with the standard deviation never exceeding 1.6 and 3.4 of the mean values of CI and startup delay, respectively.

Heterogeneous case: active seeding vs favoring fast peers

In our model and algorithm design we have considered, for the sake of simplicity, the case of an homogeneous system, where all peers have the same upload capacity $\mu = 1000$kbits/s. In order to verify the effectiveness of our approach in a more general setting, in this paragraph we present results for a set of simulations involving heterogeneous peers. Specifically, two groups of peers are considered for this experiment: *fast peers*, having upload capacity $\mu_f = 2R = 1600$ Kbits/s, and *slow peers*, having upload capacity $\mu_f = \frac{R}{2} = 400$ Kbits/s. Every time a new peer joins, it will be a fast peer with probability 0.5. Hence, on average, there will be an equal number of fast and slow peers in the system at any moment in time. Consequently, the average peer upload capacity in the system will be roughly $\frac{\mu_f+\mu_s}{2} = 1000$kbits/s, i.e. the same as for the homogeneous case.

Along with our flashcrowd-handling with active seeding (FH/AS) strategy, we have investigated another strategy where downloaders have the default behavior and the seeder favors fast peers (FF), i.e. it uploads data preferably to fast nodes. The reason for considering this latter strategy is due to the fact that, intuitively, favoring fast peers seems a good idea for spreading pieces faster, and hence reaching the steady-state sooner.

Results for a low intensity flashcrowd are shown in Figure 6.8. As we can observe, Active Seeding has a similar performance as for the homogeneous case. On the other hand, Favoring Fast Peers results in an overall worse performance (even worse than the default algorithm with no flashcrowd-handling) for both metrics considered. This phenomenon can

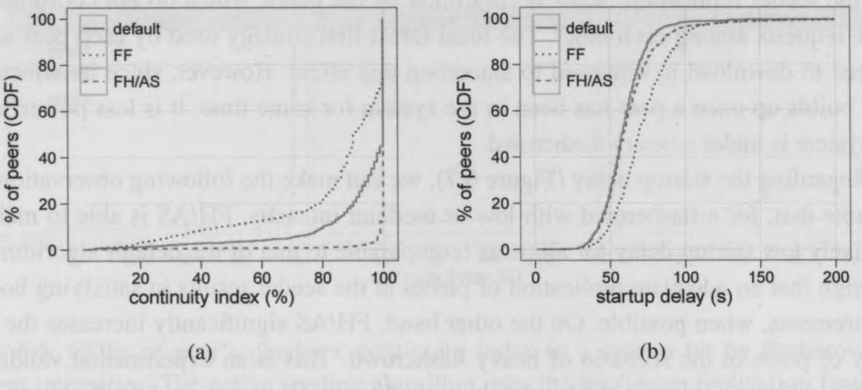


Figure 6.8: CDFs of (a) continuity index and (b) startup delay in an heterogeneous system hit by a low intensity flashcrowd.

be explained as follows. When the seeder favors a fast peer, this peer will choose to download the pieces it needs in the immediate future, rather than pieces further away in the file. As a consequence, the number of replicas for the pieces at the end of the file becomes too low, and many peers whose playback position is towards the end of the file suffer of bad viewing quality. As these peers are left with too little or no pieces to exchange, the overall efficiency of the P2P protocol decreases, which in turn determines longer startup delays.

6.5 Related work

Extensive work has been done on modeling and analyzing BitTorrent-based P2PVoD systems. Parvez *et al.* [50] study the performance of such systems and conclude that they are scalable in steady state. Lu *et al.* [64] propose a fluid model to analyze the evolution of peers over time. However, they do not consider the QoS requirements for VoD in their analysis nor focus on flashcrowd scenarios.

With respect to flashcrowds, Liu *et al.* [24] study the inherent relationship between time and scale in a generic P2P live streaming system and find an upper bound for the system scale over time. Esposito *et al.* [23] recognize the seeders to be the major bottlenecks in BitTorrent systems under flashcrowds and propose a new class of scheduling algorithms at the seeders in order reduce peer download times. However, none of these previous works analyzes the case of P2PVoD applications.

6.6 Conclusion

In this chapter, we have studied the allocation of bandwidth in a BitTorrent-based P2PVoD system during a flashcrowd. We have shown that, in order to satisfy the QoS requirements described in the introduction of this thesis, there is an upper bound for the number of new peers that can be admitted in the system in time. Furthermore, we have demonstrated that a trade-off exists between low piece replication at the seeders and high peer QoS. In particular, we have shown that, the larger a flashcrowd, the more pieces (up to a certain limit) the seeders need to replicate, in order to have peers experience an acceptable QoS. Then, we have used the insights gained from our analysis to design a class of fully decentralized flashcrowd-handling algorithms that improve peer QoS when the system is under a flashcrowd.

On a different note, our study also shows that heavy flashcrowds have a huge impact on BitTorrent-based P2PVoD systems, although peers are incentivized to contribute their bandwidth to the network. We therefore expect that systems which do not incorporate such incentives are (i) either likely to provide lower QoS to their users, since peers are not "forced" to contribute their bandwidth (and might decide not to), or (ii) they need to supply considerably more server bandwidth in order to have their service scale with the flashcrowd size, as compared to BitTorrent-based (incentivized) systems.

Chapter 7

Conclusion

In this thesis, we have investigated the problem of delivering high quality on-demand streaming by means of a highly decentralized P2P architecture. We have first identified the major factors that challenge the design of such systems, and we have quantified their impact. Then, for each of these issues, we have proposed robust and light-weight algorithms that lead to significant improvement in the QoS delivered by P2PVoD systems.

In this chapter, we present our conclusions, which will answer the research questions identified in the introduction of this thesis, and also propose suggestions for future work in this field.

7.1 Conclusions

Based on the research presented in this thesis, we draw the following major conclusions:

1. Bandwidth distribution in P2P swarming systems is highly dependent on the presence, as well as the amount, of unconnectable nodes. Since connectable peers can upload data both to connectable and unconnectable nodes, while the latter can only upload to the former, it follows that connectable nodes receive, on average, more bandwidth than unconnectable ones. Consequently, connectable nodes also experience faster download speeds and better QoS. Also, because the bandwidth of unconnectable peers can only be consumed by connectable ones, a larger fraction of unconnectable nodes determines a larger performance gap between the two types of peers. Furthermore, although bandwidth can be redistributed more evenly by means of connectability-aware policies, it will be in any case impossible for unconnectable peers to experience as good performance as connectable ones once the fraction of unconnectable nodes has gone past 50%.

2. Current BitTorrent-based P2PVoD approaches all share some similarities, such as that, although in different degrees, they all trade a QoS-oriented design with lower

freeriding-resilience or security. Our analysis also shows that current peer selection policies based on bandwidth reciprocity are not suitable to provide high QoS in heterogeneous systems where peers have different upload bandwidths or are unconnectable. Finally, we have observed that, when VoD peers coexist with file-sharing peers, the former experience poor performance while the latter usually benefit from it.

3. We have developed a set of adaptive algorithms to allow peers that are already receiving a high enough QoS to dynamically increase the fraction of their bandwidth allocated to random peers or unconnectable ones. By doing so, when the bandwidth in the system is abundant, it will be distributed more evenly among peers and hence a larger fraction of users will receive satisfactory QoS. At the same time, the adaptiveness of our strategies allows peers to become more selfish when resources are scarce, selecting less random or unconnectable peers. In this way, if there is resource contention, high-capacity (connectable) nodes are still able to receive a good service, while low-capacity (unconnectable) nodes or freeriders suffer of performance degradation. We have also shown that this adaptive mechanism can be used to significantly reduce startup delays, by giving priority to newcomers.
4. The scalability of BitTorrent-based P2PVoD systems during flashcrowds is intrinsically related to the efficiency of the underlying P2P protocol and the initial service capacity, especially in the initial phase of such phenomena. In particular, although it is important to make the P2P protocol as efficient as possible, a large initial service capacity is always necessary to support an abrupt surge of joining peers. However, once a sufficient seeders-to-leechers ratio is reached, the system enters a new phase in which it has become self-sustainable and the bandwidth contribution from the service provider is no longer needed.

5. We have proposed a set of decentralized algorithms to enable peers to quickly adjust their behavior to a flashcrowd scenario. The quick response of our algorithms is due to an efficient flashcrowd-detection mechanism, that allows peers to immediately determine its occurrence. In downloaders, such a flashcrowd-detection mechanims is coupled with a more selfish behavior as their current QoS decreases. In seeders, the flahscrowd-detection mechanism is coupled with a more efficient piece dissemination strategy. Altogether, the algorithms implemented in downloaders and seeders allow the system to meet the QoS requirements for VoD as outlined in the introduction of this thesis.

7.2 Suggestions for future work

There are several directions for further studies relative to the topics presented in this thesis. Here, we list the following:

1. We have shown that the current incentive schemes adopted in BitTorrent-based VoD systems, based on bandwidth reciprocity, are not suitable to VoD. This is due to the fact that they pursue a general file-transfer goal rather than a VoD goal. While in file-transfer the goal is to maximize the total download speed, in VoD the goal is to have as many peers as possible experience a smooth playback. Hence, as opposed to file-transfer, in VoD a peer does not earn any benefit in downloading at much higher rates than the playback rate. In this thesis, we have proposed a way to relax the reciprocity of these mechanisms when bandwidth is abundant, in order to grant more peers with a satisfactory QoS. Based on this line of work, researchers in this field should focus on creating incentives that better fit the VoD case.

2. The performance of unconnectable peers can get really low, even when they are only 30-50% (which is a realistic range in P2P swarming systems, see [45] and references therein) of the total and even when connectable peers use all their bandwidth to serve them. Therefore, in order to improve their QoS, it is important to make these nodes completely open, which calls for the need of NAT traversal. This is probably one of the reasons behind the recent development of UDP-based P2P swarming protocols (NAT traversal is much easier to implement in UDP than TCP [7]), such as Swift [3] and the uTP protocol in the μTorrent client [6].

3. A system where mixed policies are used can drastically change the performance of these policies. We have evaluated the case where one VoD protocol competes with the original BitTorrent protocol. However, an interesting direction for future research is to test other scenarios too, where, for example, different VoD protocols compete against each other. A similar approach has been used by Rahman *et al.* [56] for testing performance and robustness of P2P swarming systems. This kind of analysis will eventually help system designers in creating VoD protocols that better suit the scenarios in which they are going to be deployed.

4. In Chaper 6 we have presented a distributed flashcrowd-detection mechanism that allows peers to individually determine the occurrence of a flashcrowd based on the file completion levels of their neighbors. Although effective, this mechanism assmumes that the file size is known in advance, and therefore it is not applicable to all P2P systems, such as those that deliver live streaming content. Hence, we suggest to research other distributed mechanisms for flashcrowd-detection to adopt in these systems. Another interesting direction for further studies is using flashcrowd-detection as a mecha-

nism for implementing distributed load-balancing algorithms for multi swarm/channel systems. For example, a peer that is active in multiple swarms/channels can allocate its bandwidth to each of them proportionally to their current “flashcrowd level”.

Appendix A

Here we present a mathematical investigation that aims to characterize and understand the bartering ability of peers and its relation to the fundamental properties of BitTorrent-based VoD systems. The bartering ability of a given peer i is expressed by the expected value of the number of peers in the system with which peer i can exchange pieces of the file.

In the following we analyze the window-based piece selection policy. First, we derive the average number of *potential* bartering partners N_b for any peer in the system. In order to do so, we notice that each peer i joins $1/\lambda$ seconds after its predecessor $i-1$ to the system. Assuming that each peer i downloads pieces at a rate at least equal to the playback rate R, then i is, on average, $b_{\min} = R/(\lambda P_s)$ pieces behind its predecessor $i-1$, where P_s is the size (in Kb) of a piece. This implies that a peer has overlapping window with $\lceil \frac{w}{b_{min}} \rceil - 1$ many of its predecessors. The same holds for the successors of peer i. Therefore, the average number of potential bartering partners for any peer i is

$$N_b = 2\left(\left\lceil \frac{w}{b_{min}} \right\rceil - 1\right) = 2\left(\left\lceil \frac{wP_s}{R}\lambda \right\rceil - 1\right).$$

This concept is exemplified in Figure 7.1 for a case with $w = 6$ and $\lambda = \frac{R}{2P_s}$ (which implies that each peer i is 2 pieces behind its predecessor and thus has $N_b = 2(\frac{wP_s}{R}\lambda - 1) = 4$ bartering partners).

This simple analysis shows that low arrival rate can easily lead to $N_b = 0$, thus giving no chance for bartering, if the system parameters R, P_s and w are not set up carefully. Furthermore, it is desired in VoD systems that $N_b \geq \nu$ holds, i.e., the number of potential bartering partners should be more than the number of upload slots of the peers, otherwise peers would start allocating upload slots to randomly chosen neighbors making the system less resilient to freeriding. Note that it is not given that a peer can actually barter with N_b many peers, that needs to be analyzed below. However, we can already see that N_b depends on the arrival rate λ and the sequentiality parameter w, thus it can be adjusted to be high enough (meaning high probability of possible piece exchange between peers in the system), but that can lead to longer startup delays. Hence, we argue that *adopting a reciprocity-based approach, in a VoD system characterized by low peer arrival rates, implies that a trade-off exists between freeriding resilience and peer QoS.*

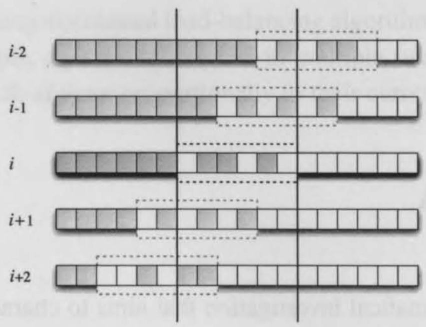


Figure 7.1: Example of overlapping windows among consecutive peers. Peers are numbered according to the time they joined the system.

Knowing the number of potential peers for bartering, we are interested now in the probability that two peers can exchange pieces between each other, assuming they have overlapping windows. For the window-based piece selection policy we can assume that both peers have downloaded half of the pieces of their current window, and that peer i is behind peer j in $d \geq b_{\min}$ pieces. Using the fact that the probability that two peers, i and j, *cannot* exchange any piece is the same as the probability that peer i cannot give pieces to peer j, we obtain

$$P[\text{two peers can exchange}] = \tag{7.1}$$

$$= 1 - P[\text{peer } i \text{ cannot give piece to peer } j] = \tag{7.2}$$

$$= 1 - \sum_{k=\max\{0,d+1-w/2\}}^{\min\{d-1,w/2\}} \binom{d-1}{k}\binom{w-d-1}{w/2-k}\binom{w/2+k-1}{k} B^*, \tag{7.3}$$

where $B^* = \binom{w-1}{w/2}^{-2}$ and we assume that $w > d$ holds. In this sum, the first term counts the number of cases that pieces can be located within the first d slots at peer i, the second term is the number of cases that pieces can be located within the overlapping area of peer i and peer j, the third term is the number of cases that the still missing pieces of peer j can be located, and finally $1/B^*$ counts the total number of possible cases. As this probability depends on w and on d, in the following we are using the notation $\mathcal{P}(w,d)$ for it.

Note that $\mathcal{P}(w,d)$ is monotonically decreasing with the increase of d; it stays very close to 1 up to $d = w/2$ and starts decreasing quickly from there on.

Using the number of potential bartering peers N_b and the probability $\mathcal{P}(w,d)$ that two overlapping peers can barter, we are ready now to calculate the expected value of the number of peers with which peer i can barter, that is

$$E[\text{number of bartering partners of } i] = 2 \sum_{d \in D} \mathcal{P}(w, d),$$

where $D = \{b_{\min}, \ldots, w-1\}$. Here, for the sake of simplicity, we assume that the probabilities that peer i can barter with its potential partners are independent from each other.

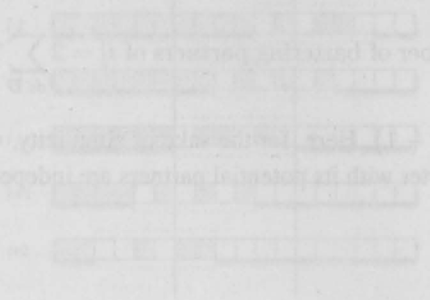


Figure 7.1: Example of overlapping windows among consecutive peers. Peers are numbered according to the time they joined the system.

Knowing the number of potential peers for bartering, we are interested now in the probability that two peers can exchange pieces between each other, assuming they have overlapping windows. For the window-based piece selection policy we can assume that both peers have downloaded half of the pieces of their current window, and that peer i is behind peer j in $d \geq b_{win}$ pieces. Using the fact that the probability that two peers, i and j, cannot exchange any piece is the same as the probability that peer i cannot give pieces to peer j, we obtain

$$P[\text{two peers can exchange}] = \tag{7.1}$$

$$= 1 - P[\text{peer } i \text{ cannot give piece to peer } j] = \tag{7.2}$$

$$= 1 - \sum_{k=\max(0,d+1-w/2)}^{\min(d-1,w/2)} \binom{d-1}{k}\binom{w-d-1}{w/2-k}\binom{w/2+k-1}{k} B^*, \tag{7.3}$$

where $B^* = \binom{w-1}{w/2}^{-2}$ and we assume that $w > d$ holds. In this sum, the first term counts the number of cases that pieces can be located within the first d slots at peer i, the second term is the number of cases that pieces can be located within the overlapping area of peer i and peer j, the third term is the number of cases that the still missing pieces of peer j can be located, and finally $1/B^*$ counts the total number of possible cases. As this probability depends on w and on d, in the following we are using the notation $P(w, d)$ for it.

Note that $P(w, d)$ is monotonically decreasing with the increase of d, it stays very close to 1 up to $d \approx w/2$ and starts decreasing quickly from there on.

Using the number of potential bartering peers N_i and the probability $P(w, d)$ that two overlapping peers can barter, we are ready now to calculate the expected value of the number

Appendix B

In order to give the proof of Lemma 2, first we introduce the following notations. Let T_B be the average number of timeslots needed by a L_1 peer to download a piece buffer of size B and let S_B be the total number of pieces allocated in the buffer by a non replicating seeder within the time interval T_B. Then, the average buffer filling rate d_B of a L_1 peer can be calculated as the sum of the total contribution through swarming and seeder over the number of timeslots T_B needed to complete the download of the buffer, i.e.

$$d_B = \frac{\sum_{t_k=t_j}^{t_j+T_B} X_s(t_k) + S_B}{T_B} = X_s + \frac{S_B}{T_B},$$

where t_j is the timeslot when the current buffer $\mathcal{B}_\alpha$ was defined, $X_s(t_k)$ is given in Eq. (6.8), and finally X_s and $\frac{S_B}{T_B}$ are the average buffer filling rates provided through swarming and by the seeder, respectively, over the time frame T_B.

Proof of Lemma 2. A non replicating seeder will provide consecutive pieces to the downloaders. This means that in timeslot t_k, it will give piece $i + k\nu_s$ to downloader p_j, piece $i + k\nu_s + 1$ to downloader p_{j+1} and so on; then at timeslot t_{k+1} the scheme will be repeated starting from piece $i + (k+1)\nu_s$. This allocation results in the seeder uploading, to each peer at each timeslot, an average number of pieces within the buffer equal to

$$\frac{S_B}{T_B}. \tag{7.4}$$

Since at each timeslot t_k, each peer receives from the seeder a piece with index ν_s higher than the piece received at the previous timeslot (Figure 7.2 shows an example where $B = 2\nu_s$ and the seeder allocates to peer i the pieces $k\nu_s + i$, with k positive integer, meaning that the seeder allocates to peer i a total of $S_B = \frac{B}{\nu_s} = 2$ pieces), then we have

$$S_B = \frac{B}{\nu_s}. \tag{7.5}$$

On the other hand, T_B depends on the buffer size B and on d_B. Though we gave the definition of d_B, its exact value can only be calculated *after* the download of the current buffer $\mathcal{B}_\alpha$ is completed. Since, at timeslot t_k, we do not know the bandwidth allocation for

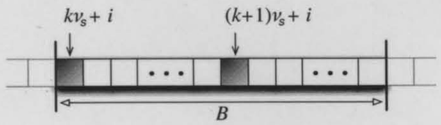


Figure 7.2: Representation of peer i's current buffer $\mathcal{B}_\alpha$.

any timeslot $t_j > t_k$, we approximate d_B to the instantaneous buffer filling rate at timeslot t_k:

$$d_B \approx d_B(t_k) = X_s(t_k) + \frac{S_B}{T_B(t_k)}. \tag{7.6}$$

which is a reasonable approximation for small B, that can be downloaded in a few rounds (i.e. $B < 5\nu_p$).

We can now calculate T_B as follows:

$$T_B \approx T_B(t_k) = \frac{B}{d_B(t_k)} = \frac{B}{X_s(t_k) + \frac{S_B}{T_B(t_k)}},$$

which yields

$$T_B \approx T_B(t_k) = \frac{B - S_B}{X_s(t_k)} = \frac{B(\nu_s - 1)}{\nu_s X_s(t_k)} \tag{7.7}$$

Plugging into Eq. (7.4) the expressions for S_B and T_B as calculated in Eqs. (7.5) and (7.7), gives

$$\frac{S_B}{T_B} = \frac{\frac{B}{\nu_s}}{\frac{B(\nu_s-1)}{\nu_s X_s(t_k)}} = \frac{X_s(t_k)}{\nu_s - 1}, \tag{7.8}$$

which concludes our proof. □

Bibliography

[1] BitTorrentDNA. http://www2.bittorrent.com/dna.

[2] μTorrent. http://www.utorrent.com.

[3] Swift Internet-Draft. http://datatracker.ietf.org/doc/draft-grishchenko-ppsp-swift/.

[4] Tribler. http://www.tribler.org.

[5] UPnP Device Architecture. UPnP Forum. http://www.upnp.org.

[6] uTorrent uTP Documentation. http://www.utorrent.com/help/documentation/utp.

[7] A. Biggadike, D. Ferullo, G. Wilson and A. Perrig. NATBLASTER: Establishing TCP connections between hosts behind NATs. In *ACM SIGCOMM Asia Workshop*, 2005.

[8] A. Legout, G. Urvoy-Keller and P. Michiardi. Rarest First and Choke Algorithms are Enough. In *ACM IMC*, 2006.

[9] A. Legout, N. Liogkas, E. Kohler and L. Zhang. Clustering and Sharing Incentives in BitTorrent Systems. In *ACM SIGMETRICS*, 2007.

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

[11] E. Adar and B.A. Huberman. Free riding on gnutella. *First Monday*, 5(10):pp.2–13, 2000.

[12] A.L. Jia, L. D'Acunto, M. Meulpolder, J.A. Pouwelse and D. Epema. Enhancing Reciprocity or Reducing Inequity: BitTorrent's Dilemma. In *TU Delft PDS, Technical Report*, 2010.

[13] Siddhartha Annapureddy, Saikat Guha, Christos Gkantsidis, Dinan Gunawardena, and Pablo Rodriguez Rodriguez. Is high-quality vod feasible using p2p swarming? In *Proceedings of the 16th international conference on World Wide Web*, pages 903–912. ACM, 2007.

[14] A.R. Bharambe, C. Herley and V.N. Padmanabhan. Analyzing and Improving a BitTorrent Network's Performance Mechanisms. In *IEEE INFOCOM*, April 2006.

[15] B. Cheng, X. Liu, Z. Zhang and H. Jin. A Measurement Study of a Peer-to-Peer Video-on-Demand System. In *IPTPS*, 2007.

[16] B. Ford, P. Srisuresh and D. Kegel. Peer-to-peer communication across network address translators. In *the annual conference on USENIX Annual Technical Conference (ATEC '05)*, 2005.

[17] S. Xie Y. Qu G. Y. Keung C. Lin J. Liu B. Li and X. Zhang. Inside the New Coolstreaming: Principles, Measurements and Performance Implications. In *INFOCOM 2008. The 27th IEEE Conference on Computer Communications. IEEE*, 2008.

[18] A. Bakker, R. Petrocco, M. Dale, J. Gerber, V. Grishchenko, D. Rabaioli, and J. Pouwelse. Online video using BitTorrent and HTML5 applied to Wikipedia. In *IEEE P2P*, pages 1–2, 2010.

[19] C. Huang, J. Li, and K. Ross. Can Internet Video-on-Demand Be Profitable. In *ACM SIGCOMM*, 2007.

[20] Niklas Carlsson and Derek Eager. Peer-assisted on-demand streaming of stored media using bittorrent-like protocols. In *IFIP NETWORKING*, pages 570–581. Springer, 2007.

[21] B. Cohen. Incentives Build Robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer Systems*, Berkeley, USA, May 2003. http://bittorrent.com.

[22] D. Wu, C. Liang, Y. Liu, and K. Ross. View-Upload Decoupling: A Redesign of Multi-Channel P2P Video Systems. In *IEEE INFOCOM*, 2009.

[23] F. Esposito, I. Matta, D. Bera, P. Michiardi. On the Impact of Seed Scheduling in Peer-to-Peer Networks. In *Computer Networks*, 2011.

[24] F. Liu, B. Li, L. Zhong, B. Li, D. Niu. How P2P Streaming Systems Scale Over Time Under a Flash Crowd? In *IPTPS*, 2009.

[25] M. Feldman, K. Lai, I. Stoica, and J. Chuang. Robust incentive techniques for peer-to-peer networks. In *Proc. of the 5th ACM conference on Electronic Commerce (EC'04)*, pages 102–111, May 2004.

[26] P. Garbacki, D.H.J. Epema, J. Pouwelse, and M. van Steen. Offloading servers with collaborative video on demand. In *Proceedings of the 7th International Workshop on Peer-to-peer systems*. USENIX Association, 2008.

[27] Yang Guo, Kyoungwon Suh, Jim Kurose, and Don Towsley. P2cast: peer-to-peer patching scheme for vod service. In *Proceedings of the 12th international conference on World Wide Web*, pages 301–309. ACM, 2003.

[28] A. Habib and J. Chuang. Service Differentiated Peer Selection: An Incentive Mechanism for Peer-to-Peer Media streaming. *IEEE Transactions on Multimedia*, 8:610–621, 2006.

[29] Oliver Heckmann and Axel Bock. The edonkey2000 protocol. Technical report, KOM, 2004.

[30] Y. Huang, T.Z.J. Fu, D.M. Chiu, J. Lui, and C. Huang. Challenges, design and analysis of a large-scale p2p-vod system. *ACM SIGCOMM Computer Communication Review*, 38(4):375–388, 2008.

[31] M. Izal, Guillaume Urvoy-Keller, Ernst Biersack, P. Felber, A. Al Hamra, and L. Garcs-Erice. Dissecting bittorrent: Five months in a torrents lifetime. In *Passive and Active Network Measurement*. Springer Berlin / Heidelberg, 2004.

[32] J.J.D. Mol, A. Bakker, J.A. Pouwelse, D.H.J. Epema, and H.J. Sips. The Design and Deployment of a BitTorrent Live Video Streaming Solution. In *Proceedings of IEEE International Symposium on Multimedia (ISM'09)*, december 2009.

[33] J.J.D. Mol, J. A. Pouwelse, M. Meulpolder, D.H.J. Epema and H.J. Sips. Give-to-Get: Free-riding-resilient Video-on-Demand in P2P Systems. In *SPIE MMCN*, 2008.

[34] D. Magoni H. Chang W. Wang K. Shami and S. Jamin. Impacts of Peer Characteristics on P2PTV Networks Scalability. In *INFOCOM 2009. The 28th IEEE Conference on Computer Communications. IEEE*, April 2009.

[35] V. Goebel K.A. Skevik and T. Plagemann. Analysis of BitTorrent and its use for the Design of a P2P based Streaming Protocol for a Hybrid CDN. Technical report, Department of Informatics, University of Oslo, 2004.

[36] L. D'Acunto, M. Meulpolder, R. Rahman, J.A. Pouwelse and H.J. Sips. Modeling and Analyzing the Effects of Firewalls and NATs in P2P Swarming Systems. In *Proceedings of IEEE IPDPS (HotP2P)*, 2010.

[37] L. D'Acunto, N. Andrade, J.A. Pouwelse and H.J. Sips. Peer Selection Strategies for Improved QoS in Heterogeneous BitTorrent-like VoD Systems. In *IEEE International Symposium on Multimedia (ISM'2010)*, 2010.

[38] L. D'Acunto, T. Vinko and H.J. Sips. Bandwidth Allocation in BitTorrent-like VoD Systems under Flashcrowds. In *IEEE P2P*, 2011.

[39] L. D'Acunto, T. Vinko and J.A. Pouwelse. Do BitTorrent-like VoD Systems Scale under Flashcrowds? In *IEEE P2P*, 2010.

[40] L. Guo and S. Chen and Z. Xiao and E. Tan and X. Ding and X. Zhang. A performance study of BitTorrent-like peer-to-peer systems. *IEEE Journal on Selected Areas in Communications*, January 2007.

[41] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding and X. Zhang. Measurements, analysis, and modeling of bittorrent-like systems. In *IMC '05: Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, Berkeley, CA, USA, 2005. USENIX Association.

[42] Dave Levin, Katrina LaCurts, Neil Spring, and Bobby Bhattacharjee. Bittorrent is an auction: analyzing and improving bittorrent's incentives. In *ACM SIGCOMM*, 2008.

[43] Qiao Lian, Yu Peng, Mao Yang, Zheng Zhang, Yafei Dai, and Xiaoming Li. Robust incentives via multi-level tit-for-tat. In *Proc. 5th International Workshop on Peer-to-Peer systems (IPTPS)*, 2006.

[44] M. Meulpolder, J.A. Pouwelse, D.H.J. Epema and H.J. Sips. Modeling and Analysis of Bandwidth-Inhomogeneous Swarms in BitTorrent. In *Proceeding of the IEEE 9th International Conference on P2P Systems (P2P'09)*, 2009.

[45] M. Meulpolder, L. D'Acunto, M. Capota, M. Wojciechowski, J.A. Pouwelse, D.H.J. Epema and H.J. Sips. Public and private bittorrent communities: A measurement study. In *IPTPS*, 2010.

[46] M. Piatek, A. Krishnamurthy, A. Venkataramani, R. Yang, D. Zhang, A. Jaffe. Contracts: Practical Contribution Incentives for P2P Live Streaming. In *USENIX NSDI*, 2010.

[47] J.J.D. Mol, J.A. Pouwelse, D.H.J. Epema, and H.J. Sips. Free-riding, Fairness, and Firewalls in P2P File-sharing. In *Proceeding of the IEEE 8th International Conference on P2P Computing (P2P'08)*, September 2008.

[48] N. Carlsson, D. L. Eager and A. Mahanti. Peer-assisted on-demand Video Streaming with Selfish Peers. In *IFIP NETWORKING*, 2009.

[49] N. Magharei, R. Rejaie, Y. Guo. Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches. In *IEEE INFOCOM*, 2007.

[50] N. Parvez, and C. Williamson and A. Mahanti and R. Carlsson. Analysis of BitTorrent-like Protocols for On-Demand Stored Media Streaming. In *ACM SIGMETRICS*, 2008.

[51] P. Savolainen, N. Raatikainen and S. Tarkoma. Windowing BitTorrent for Video-on-Demand: Not All is Lost with Tit-for-Tat. In *IEEE GLOBECOM*, 2007.

[52] P. Shah and J. F. Pris. Peer-to-Peer Multimedia Streaming using BitTorrent. In *IEEE IPCCC*, 2007.

[53] D. Qiu and R. Srikant. Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks. In *Proceedings of ACM SIGCOMM*, 2004.

[54] R. Rahman, M. Meulpolder, D. Hales, J.A. Pouwelse and H.J. Sips. Revisiting Social Welfare In P2P. In *TU Delft PDS, Technical Report: PDS-2009-003*, 2009.

[55] R. Rahman, M. Meulpolder, D. Hales, J.A. Pouwelse, D. Epema and H.J. Sips. Improving Efficiency and Fairness in P2P Systems with Effort-Based Incentives. In *IEEE ICC*, 2010.

[56] Rameez Rahman, Tamás Vinkó, David Hales, Johan Pouwelse, and Henk Sips. Design space analysis for modeling incentives in distributed systems. In *ACM SIGCOMM*, pages 182–193, 2011.

[57] R.K. Dash, N.R. Jennings and D.C. Parkes. Computational-mechanism design: A call to arms. In *IEEE Intelligent Systems*, 2003.

[58] S. Guha and P. Francis. Characterization and Measurement of TCP Traversal through NATs and Firewalls. In *Proceeding of the Internet Measurement Conference (IMC'05)*, 2005.

[59] S. Tewari and L. Kleinrock. Analytical model for BitTorrent-based live video streaming. In *Proceedings of IEEE NIME Workshop*, 2007.

[60] S. Xie, G.Y. Keung and B. Li. A Measurement of a large-scale Peer-to-Peer Live Video Streaming System. In *the IEEE International Conference on Parallel Processing Workshops (IEEE ICPPW 2007)*, 2007.

[61] K.A. Skevik, V. Goebel, and T. Plagemann. Evaluation of a comprehensive p2p video-on-demand streaming system. *Computer Networks*, 53(4):434–455, 2009.

[62] X. Zhang, J. Liu, B. Li, and T.S. P. Yum. DONet/CoolStreaming: A Data-driven Overlay Network for Live Media Streaming. In *IEEE INFOCOM*, 2005.

[63] Y. Borghol, S. Ardon, N. Carlsson and A. Mahanti. Toward Efficient On-Demand Streaming with BitTorrent. In *IFIP NETWORKING*, 2010.

[64] Y. Lu and J.J.D. Mol and F. Kuipers and P. Van Mieghem. Analytical Model for Mesh-based P2PVoD. In *IEEE ISM*, 2008.

[65] Y. Yang, A. Chow, L. Golubchik, and D. Bragg. Improving qos in bittorrent-like vod systems. In *Proceedings of IEEE INFOCOM*, 2010.

[66] Z. Ge, D. R. Figueiredo, S. Jaiswal, J. Kurose, and D. Towsley. Modeling peer-peer file sharing systems. In *Proceeding of IEEE INFOCOM*, October 2003.

[67] Manaf Zghaibeh, Kostas G. Anagnostakis, and Fotios C. Harmantzis. The behavior of free riders in bittorrent networks. In *Handbook of Peer-to-Peer Networking*, pages 1207–1230. Springer, 2010.

Summary

Challenges, Design and Analysis of Peer-to-Peer Video-on-Demand Systems

Recently, significant research effort has focused on the effective use of a peer-to-peer (P2P) architecture to provide large-scale video-on-demand (VoD) services. In fact, the natural scalability of a P2P approach could drastically decrease the costs of video service providers. However, delivering on-demand services using P2P is also a challenging task. Similar to live P2P streaming systems, some quality-of-service (QoS) requirements have to be fulfilled, namely providing users with a high playback continuity and a short startup delay. But in VoD, different from live-streaming, the playback positions of nodes that request a video at different times would differ greatly.

Since early P2P systems had been built for file-sharing applications, trying to adapt the designs of these systems to the VoD case came as a natural choice. In particular BitTorrent, making nearly optimal use of peers' upload bandwidth, has inspired many P2P protocols for VoD. BitTorrent achieves a high utilization of peers' upload bandwidth by means of a smart piece retrieval mechanism and strong incentives for cooperation. In this context, previous work on BitTorrent-like P2P VoD systems mainly focuses on adapting BitTorrent's piece retrieval mechanism to make it suitable for streaming.

In this thesis we direct our attention on BitTorrent's incentive mechanism. In fact, this mechanism, based on *bandwidth reciprocity*, has also been clearly designed with a file-sharing goal in mind. Here we show that, in the heterogeneity of today's internet, this approach poses further challenges to VoD applications. When peers have different upload bandwidths, for example, it can happen that nodes with low upload capacities cannot reach a download speed high enough to sustain the video playback rate, while those with higher upload capacities download at rates much higher than actually needed. Likewise, peers residing behind NATs or firewalls blocking unsolicited inbound connections might experience poor performance, much to the benefit of fully connectable nodes. In this thesis, we propose a solution that allows peers to *"relax" their reciprocity-based mechanism when they are already experiencing a good enough service*, consequently serving low-capacity or unconnectable peers. Our solution maintains the robustness against freeriding of the original mechanism, since the nodes who contribute less are only served when there is enough capacity available in the system.

Furthermore, for systems that need to fulfill QoS requirements, flashcrowds represent an

additional issue. In fact, when a sudden surge of new peers joins the system, the average peer download speed likely becomes lower than necessary to maintain a smooth playback continuity, since the new peers all demand for bandwidth while having little or nothing to provide in return. In this scenario, most peers will experience a poor QoS, unless the available bandwidth is properly allocated. In this thesis, we analyze this aspect in detail in the context of VoD systems and we present a set of *distributed flashcrowd-handling algorithms* that considerably improve peer QoS during flashcrowds.

Samenvatting

Uitdagingen, Ontwerp en Analyse van Peer-to-Peer Video-on-Demand Systemen

De afgelopen jaren is er een significante hoeveelheid onderzoek uitgevoerd gericht op het effectief gebruiken van peer-to-peer (P2P) architectuur om grootschalige video-on-demand (VoD) diensten te leveren. Vooral omdat de intrinsieke schaalbaarheid van een P2P opzet de kosten van video service providers drastisch kan verlagen. Het leveren van on-demand diensten met P2P is echter ook een uitdaging. Zoals ook in P2P live-streaming systemen het geval is, moet er aan een aantal servicekwaliteitseisen (QoS) worden voldaan, namelijk een ononderbroken afspeel-continuïteit en een korte wachttijd voordat de video start. Maar VoD heeft als extra uitdaging ten opzicht van live-streaming dat er grote verschillen tussen de afspeelposities van peers bestaan, wanneer die hun video op verschillende momenten hebben opgevraagd.

Omdat de eerste P2P systemen gebouwd waren voor file-sharing programma's, lag het voor de hand om deze ontwerpen te gebruiken als basis voor video-on-demand systemen. Vooral BitTorrent, een systeem dat vrijwel optimaal gebruik maakt van de upload-bandbreedte van peers, is een inspiratiebron voor veel P2P VoD protocollen. BitTorrent kan de upload-bandbreedte van peers efficiënt inzetten door bestand-stukjes slim op te vragen en sterke incentives te geven om elkaar te helpen. In dit kader richtte eerder onderzoek naar P2P VoD systemen op basis van BitTorrent zich vooral op het geschikt maken van BitTorrents opvraag-mechanisme voor het gebruik in video-streaming.

In dit proefschrift gaat onze aandacht uit naar BitTorrents incentive mechanisme. Dit mechanisme, gebaseerd op *reciprociteit van bandbreedte*, is duidelijk ontworpen met file-sharing in gedachte. We tonen aan dat in de verscheidenheid van het hedendaags Internet het ene uitdaging is dit mechanisme toe te passen in VoD programma's. Als peers bijvoorbeeld verschillende upload bandbreedtes hebben, kan het gebeuren dat nodes met lage upload capaciteit niet genoeg download snelheid kunnen verkrijgen om de video ononderbroken te kunnen afspelen, terwijl de peers met hogere upload capaciteit in veel hogere snelheden downloaden dan nodig is. Zo worden ook peers die verbonden zijn achter NATs of firewalls die ongewenste inkomende verbindingen blokkeren negatief benvloed in het voordeel van de peers die algeheel bereikbaar zijn. In dit proefschrift stellen we een oplossing voor die peers toestaat om *hun reciprociteits-mechanisme te versoepelen in het geval dat zij al een goede video kwaliteit ervaren*, en hiermee peers met lage capaciteit of bereikbaarheid te ondersteunen. Onze oplossing behoudt de robuustheid van het originele mechanisme tegen freeriding, omdat de nodes die minder bijdragen alleen worden ondersteund als er genoeg capaciteit beschikbaar is in het systeem.

Voor systemen die aan servicekwaliteitseisen (QoS) dien te voldoen vormen verder flashcrowds een uitdaging. Als tijdens een flashcrowd een plotselinge stroom van nieuwe peers toetreedt tot het systeem, wordt de gemiddelde download snelheid van peers normaliter lager dan nodig is om de video ononderbroken te laten afspelen, omdat de toetredende peers allemaal vragen om bandbreedte terwijl ze weinig tot niets kunnen teruggeven. In dit scenario zal het merendeel van de peers een slechte servicekwaliteit ervaren, tenzij de beschikbare bandbreedte correct wordt toegewezen. In dit proefschrift analyseren we flashcrowds in detail in de context van VoD systemen en presenteren we een verzameling *gedistribueerde flashcrowd-handling algoritmen* die de ervaren servicekwaliteit van peers significant verbeteren tijdens flashcrowds.

Curriculum vitae

Lucia D'Acunto was born in Nocera Inferiore, Italy, on August 5th, 1983. She grew up in Pozzuoli, an amusing and green historical town near Naples, in Italy. Upon completing highschool, she enrolled at the University of Naples Federico II in 2001. There, she received both her BSc and MSc in computer engineering (cum laude) in 2004 and 2007, respectively. Her MSc thesis reseach was carried out at the Facultés Universitaires Notre-Dame de la Paix in Namur, Belgium. There, she developed different resource scheduling algorithms for mobile telephone data transmission. Right after receiving her MSc, she joined the Delft University of Technology to pursue a PhD in computer science.

Lucia currently works as a technical consultant at Collis, advising international banks and mobile phone operators on how to set up mobile payments for their users using the NFC technology. In her free time she enjoys eating good food, traveling and making origami.

Publications:

- L. D'Acunto, Nitin Chiluka, Tamás Vinkó and H.J. Sips. BitTorrent-like P2P Approaches for VoD: A Comparative Study. Submitted to *Computer Networks*.
- L. D'Acunto, Tamás Vinkó and H.J. Sips. Bandwidth Allocation in BitTorrent-like VoD Systems under Flashcrowds. In *Proc. of IEEE P2P*, 2011.
- A.L. Jia, L. D'Acunto, M. Meulpolder, and J.A. Pouwelse. Modeling and analysis of sharing ratio enforcement in private BitTorrent communities. In *Proc. of ICC*, 2011.
- A.L. Jia, L. D'Acunto, M. Meulpolder, J.A. Pouwelse and D.H.J. Epema. BitTorrent's Dilemma: Enhancing Reciprocity or Reducing Inequity. In *Proc. of CCNC*, 2011.
- L. D'Acunto, N. Andrade, J.A. Pouwelse and H.J. Sips. Peer Selection Strategies for Improved QoS in Heterogeneous BitTorrent-like VoD Systems. In *Proc. of IEEE ISM*, 2010.
- L. D'Acunto, Tamás Vinkó and J.A. Pouwelse. Do BitTorrent-like VoD Systems Scale under Flash-Crowds? In *Proc. of IEEE P2P*, 2010.

- M. Meulpolder, L. D'Acunto, M. Capotă, M. Wojciechowski, J.A. Pouwelse, D.H.J. Epema and H.J. Sips. Public and private BitTorrent communities: A measurement study. In *Proc. of IPTPS*, 2010.
- L. D'Acunto, M. Meulpolder, R. Rahman, J.A. Pouwelse and H.J. Sips. Modeling and Analyzing the Effects of Firewalls and NATs in P2P Swarming Systems. In *Proc. of Hot-P2P (in conjunction with IPDPS)*, 2010.
- L. D'Acunto, J.A. Pouwelse and H.J. Sips. A Measurement of NAT & Firewall Characteristics in Peer-to-Peer Systems. In *Proc. of ASCI*, 2009.