Delft University of Technology
Parallel and Distributed Systems Report Series

Bandwidth Allocation in BitTorrent-like VoD Systems under Flashcrowds

Lucia D'Acunto, Tamás Vinkó, Henk Sips
l.dacunto@tudelft.nl

Completed June 2011.

Report number PDS-2011-003

ISSN 1387-2109

Published and produced by:
Parallel and Distributed Systems Section
Faculty of Information Technology and Systems Department of Technical Mathematics and Informatics
Delft University of Technology
Zuidplantsoen 4
2628 BZ Delft
The Netherlands

Information about Parallel and Distributed Systems Report Series:
`reports@pds.twi.tudelft.nl`

Information about Parallel and Distributed Systems Section:
`http://www.pds.twi.tudelft.nl/`

Abstract

The efficiency of BitTorrent in content distribution has inspired a number of peer-to-peer (P2P) protocols for on-demand video (VoD) streaming systems (henceforth BitTorrent-like VoD systems). However, the fundamental quality-of-service (QoS) requirements of VoD (i.e. providing peers with a smooth playback continuity and a short startup delay) make the design of these systems more challenging than normal file-sharing systems. In particular, the bandwidth allocation strategy is an important aspect in the design of BitTorrent-like VoD systems, which becomes even more crucial in a scenario where a large number of peers joins in a short period of time, a phenomenon known as *flashcrowd*. In fact, the new joining peers all demand for content while having few or no pieces of content to offer in return yet. An unwise allocation of the limited bandwidth actually available during this phase may cause peers to experience poor QoS.

In this work, we analyze the effects of a flashcrowd on the scalability of a BitTorrent-like VoD system and propose a number of mechanisms to make the bandwidth allocation in this phase more effective. In particular, we derive an upper bound for the number of peers that can be admitted in the system over time and we find that there is a trade-off between having the seeders 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 extensive simulations.

1 Introduction

In recent years, significant research effort has focused on how to efficiently use a P2P architecture to provide large-scale VoD services. In particular, much has been investigated on how to utilize the design of BitTorrent to create efficient P2P VoD protocols [1, 6, 10, 11, 13]. Adapting BitTorrent's bandwidth allocation strategy to VoD is challenging because, similar to P2P live streaming systems, content has to be delivered by streaming, which imposes some QoS requirements, i.e. providing users with smooth playback continuity and a short startup delay. On the other hand, unlike P2P live streaming systems, in P2P VoD systems different peers can be interested in different parts of the video at a certain moment over time, hence the peer dynamics resemble those of P2P file-sharing systems.

While it has been demonstrated that these systems can attain a high performance once they have reached a steady state [8], it is still unclear how well they deal with a phenomenon known as *flashcrowd*, in which a large number of peers joins within a short period of time. In fact, it is considerably more challenging for a P2P VoD system to accommodate an abrupt surge of new joining peers, while still providing an acceptable service to existing ones. Thus, it is evident that an unwise bandwidth allocation strategy during this phase may delay reaching the steady state and cause peers to experience poor QoS.

Despite the relevance of the problem, to date only a few research efforts have investigated P2P systems under flashcrowds and they mainly address file-sharing and live streaming applications (see [4, 5]). However, the analysis presented in [16] shows that flashcrowds affect P2P VoD systems as well. Motivated by these observations, in this work, we seek to study P2P VoD systems under flashcrowds. More specifically, due to BitTorrent's efficiency and high proliferation of BitTorrent-inspired VoD protocols, in our study we focus on a BitTorrent-like design.

Our analysis aims to answer the following questions: (i) how does a flashcrowd affect a BitTorrent-like VoD system? (ii) how can bandwidth allocation be made more effective in enhancing peer QoS during flashcrowd? With respect to the second research question, we have especially investigated the role of the seeders, as they represent the major bottleneck when bandwidth is scarce [2, 4].

To summarize, we make the following contributions:

- We devise an analytical model that captures the dynamics of peers in a BitTorrent-like VoD system during a flashcrowd.
- Using this model, first we find an upper bound to the number of newcomers that can be admitted in the system over time, and then we show that a trade-off exists between having the seeder minimize the upload of pieces already injected recently and high peer QoS.
- Finally, employing the insights of our analysis, we present and evaluate a class of flashcrowd-handling algorithms to make bandwidth allocation more effective during flashcrowds, thereby improving peer

QoS.

2 Related Work

BitTorrent is a widely popular P2P protocol for content distribution. In BitTorrent, files are split into pieces, allowing peers which are still downloading content to serve the pieces they already have to others. Nodes find each other through a central *tracker*, which provides them with a random subset of peers in the system. 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. More specifically, each peer divides equally its upload capacity into a number of *upload slots*. Peers that are currently assigned an upload slot from a node p are said to be *unchoked* by p; all the others are said to be *choked* by p. The unchoking policy adopted by BitTorrent, and many of its variants, is based on a kind of *tit-for-tat*: peers prefer unchoking nodes that have recently provided data to them at the highest speeds. Each peer maintains its neighborhood informed about the pieces it owns. The information received from its 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, so to increase piece diversity.

Because of its high efficiency, a lot of research has been conducted on adapting BitTorrent to VoD (see [1, 6, 10, 11, 13]). These studies mainly focus on the piece selection policy, exploring the trade-off between the need of sequential download progress and high piece diversity. Also, extensive work has been done on modeling and analyzing BitTorrent-like VoD systems. Parvez et al. [8] study the performance of such systems and conclude that they are scalable in steady state. Lu et al. [14] propose a fluid model to analyze the evolution of peers over time. However, they do not consider the QoS requirements for VoD (to be discussed in Section 3.2) in their analysis nor they focus on the flashcrowd scenario.

With respect to flashcrowds, Liu et al. [5] 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. [4] 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 P2P VoD applications.

3 System Model and Fundamental Principles

In this section, we present a discrete-time model to describe a BitTorrent-like VoD system under flashcrowd. Then, we discuss the fundamental QoS requirements for a VoD system and derive an upper bound for the system scale over time.

3.1 Model

We consider a BitTorrent-like VoD system consisting of an initial *seeder*, 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 streaming 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 1.

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 streaming rate R. This is equivalent to the concept of *substreams* used in commercial P2P streaming systems (e.g. Coolstreaming [12]) and in P2P streaming literature (e.g. [3, 5]), where a video stream is divided into many substreams of equal size and nodes could download different substreams from different peers.

Table 1: Model parameters.

Notation	Definition
F	filesize (Kbits).
n	number of pieces the file is split into.
R	streaming 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).

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

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

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

For simplicity, 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 our 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 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.

$$U(t_k) = M + \mu x(t_k) + \mu y(t_k). \tag{1}$$

3.2 QoS Requirements for VoD

The upload decisions made by peers at the beginning of each timeslot t_k should aim at satisfying the fundamental QoS requirements for streaming. Firstly, peers should be able to play the video as smoothly as possible. This means that those peers whose playback has already started should maintain, on average, a download rate of at least R. For the purpose of the analysis, we assume that all sharers have already started playback and hence we have:

QoS Requirement 1: maximize the number of sharers who maintain a download rate of at least R at each timeslot t_k.

Secondly, it is desirable that joining peers experience low startup delays. Therefore, we have:

QoS Requirement 2: maximize the number of newcomers selected for upload at each timeslot t_k.

With regards to these requirements we make the following observation: allowing a peer to start the playback means that the system has committed itself to provide a satisfactory playback continuity to that peer, while no commitment has been established with a newcomer yet. Hence, when the bandwidth is scarce, it is more important to serve those peers that have already started playing, rather than admitting new nodes in the system (i.e. Requirement 1 has priority over Requirement 2). Furthermore, by doing so, we also avoid admitting in the system peers whose playback continuity cannot be guaranteed due to bandwidth scarcity.

3.3 Scalable System

An immediate consequence of QoS Requirement 1 is that, for a BitTorrent-like VoD system to scale with the number of peers, it must hold that $R \leq \mu$. When this is not the case (i.e. when $R > \mu$), the sharers alone are not able to support themselves with a downloading rate of at least R, and an additional amount of bandwidth equal to $R - \mu$ has to be provided to support each new sharer. In the remainder of this paper, we will only focus on *scalable systems* where, by definition, $R \leq \mu$ holds.

3.4 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, until they complete the download of their first piece, newcomers consume bandwidth without providing any bandwidth in return. 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 QoS Requirement 1. 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

From QoS Requirement 1 it follows that the minimum amount of bandwidth $U_x(t_k)$ that needs to be reserved for the sharers at timeslot t_k is

$$U_x(t_k) = Rx(t_k). \tag{2}$$

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-like VoD system with streaming 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*

$$\hat{z}(t_k) \leq \frac{M + \mu y(t_k) + (\mu - R)x(t_k)}{r}. \tag{3}$$

Proof. Taking Eq. (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. (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. When this happens, it is impossible to avoid newcomers experience longer startup delays, as we will show with our experiments in Section 6.

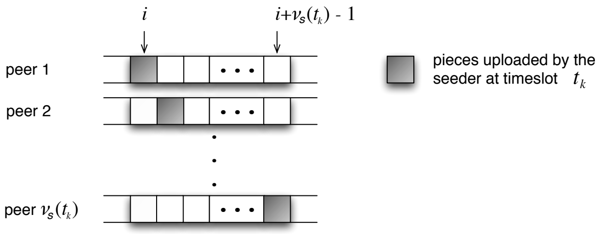


Figure 1: Seeder's piece allocation at timeslot t_k.

4 Seeders' Piece Allocation Analysis

In this section, we will to study the piece allocation strategy of the seeders in a BitTorrent-like VoD system during a flashcrowd. The reason to focus on the seeders is twofold. Firstly, their piece allocation strategy is a crucial aspect during flashcrowd, as seeders are the only interesting peers in that phase (all other peers have few or no pieces yet). Secondly, the complete bandwidth allocation problem in a BitTorrent-like system is NP-hard [4].

We proceed by first defining the features of the BitTorrent-like VoD protocol we consider and introducing a useful concept to understand the flow of data from the seeders to the peers. Then, we analyze in detail the seeders' piece allocation.

4.1 Protocol Features

For the purpose our analysis, we assume that the seeders coordinate their behaviors. Consequently, in the remainder of this section, we assume that there is only one seeder in the system holding the total seeding capacity $\nu_s(t_k)$ available at each timeslot t_k. We assume that the 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 each moment in time.

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

Seeder Piece Allocation

Having denoted with $\nu_s(t_k)$ the total number of upload slots provided by the seeder at timeslot t_k, we assume that

3) each of these slots is allocated to a different peer;

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

5) unless otherwise specified, at each timeslot t_k, the seeder uploads 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 1. We note that this scheme increases the bartering abilities among peers, hence allowing a high peer bandwidth utilization.

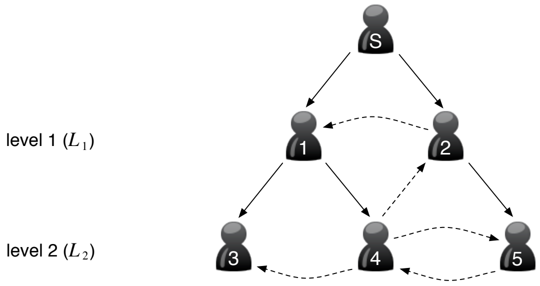


Figure 2: Organized view of an overlay mesh relative to a BitTorrent-like VoD system. Solid arrows and dashed arrows represents diffusion connections and swarming connections, respectively.

Piece Download Scheme

According to the 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, the 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 pieces $[i, i + B - 1]$, where $i = \alpha B$, with $\alpha \in \{0, 1, 2, \ldots, \lfloor \frac{n}{B} \rfloor\}$.

Once all the pieces in the current buffer $\mathcal{B}_\alpha$ are downloaded, a peer defines 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 [9, 10, 13]), a static buffer makes the computation of its filling rate easier, which will be useful in the analysis in Section 4.3.

4.2 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 [7]. In this view, downloaders are grouped into *levels* based on their shortest distance from the seeder through the overlay as shown in Figure 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*.

4.3 Piece Replication at the Seeder

A seeder might decide to upload only 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 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 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-like VoD system with streaming 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*

$$\max F_k = \frac{\nu_s - \frac{R}{r}}{\nu_s}. \tag{4}$$

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 means 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 paper (Section 6), the upper bound for the seeder replication factor is also the value yielding the best playback continuity. In fact, on 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-like VoD system with streaming 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*

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

where

$$K = \nu_p - \frac{R}{r}, \tag{5}$$

$$Z_1(t_k) = \frac{R}{r} + Kx(t_k), \tag{6}$$

$$Z_2(t_k) = \nu_s + Kx(t_k). \tag{7}$$

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

Lemma 2. *Given a BitTorrent-like VoD 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. (5) and (7), respectively.

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

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

$$X_s(t_k) = X_1(t_k) - X_2(t_k) = \nu_s \frac{R}{r} + Kx(t_k) - Z_m(t_k). \tag{8}$$

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

$$F_k = \frac{C(t_k)}{\nu_s} = \max\left\{0, \frac{Z_m(t_k) - \frac{R}{r} - Kx(t_k)}{\nu_s - 1}\right\}. \tag{9}$$

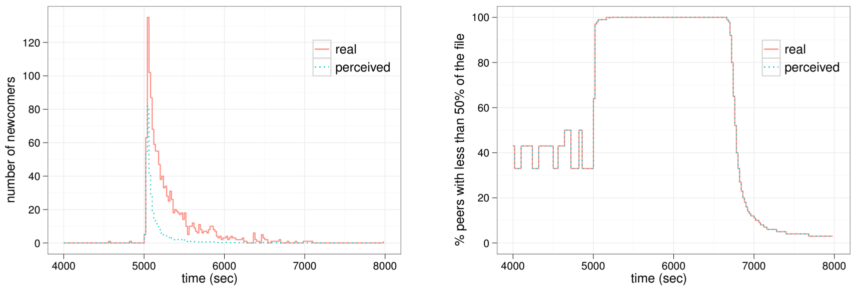
From Eq. (9) we notice that, when $z(t_k) \leq Z_1(t_k) = \frac{R}{r} + Kx(t_k)$, the seeder does not need to perform any replication. Furthermore, we observe that, when $z(t_k) \geq Z_2(t_k) = \nu_s + Kx(t_k)$, the minimum seeder replication factor equals to $\frac{\nu_s - \frac{R}{r}}{\nu_s - 1}$, which completes our proof. □

5 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-like VoD systems under flashcrowds more effective in enhancing the QoS requirements of peers. First, we explore some methods to allow a peer to detect whether the system is under flashcrowd, and then, we describe our algorithms in detail.

5.1 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-like systems has been shown to be NP-hard [4].



(a) Flashcrowd detection based on the number of newcomers.

(b) Flashcrowd detection based on the percentage of peers with less than 50% of the file.

Figure 3: Comparison between the real value and the value perceived by a peer for different flashcrowd detection metrics. 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. All the other parameters are like in Table 2.

Hence, in this paper, we will use an heuristic approach where each peer 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 3(a), obtained running the BitTorrent-like VoD protocol proposed in [13] under the settings described in Section 6.1 of this paper). 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:

(b) **Fraction of neighbors having less than 50% of the file**. Esposito et al. [4] 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.

Our experiments corroborate the findings of Esposito et al. [4]. 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 (Figure 3(b)). These results confirm that (b) represents a good metric for a peer to detect a flashcrowd only based on his 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 needs also 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.

5.2 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) it measures that its 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 streaming 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-like VoD 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 3.1 and 4.3.

Flashcrowd-handling algorithm for the sharers

Recall that, when bandwidth is scarce, the priority of a BitTorrent-like VoD system is to meet the QoS Requirement 1, i.e. maximize the number of sharers that keep a smooth playback continuity (Section 3.2). Hence, 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.

Flashcrowd-handling algorithm for the seeder

As we have observed in Section 4.3, 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 4 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 have the seeder keep the oldest peers always unchoked. Then, we have implemented two different classes of seeding behavior as reported below.

1) **Passive seeding (FH with 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 with 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 4.

For what concerns the coordination of multiple seeders, we make the following observations. Firstly, in a flashcrowd scenario, typically there is only one or a few seeders in the swarm, i.e. the content injectors.

[1] a peer's sequential progress is defined as the rate at which the index of the first missing piece in the file grows [6]

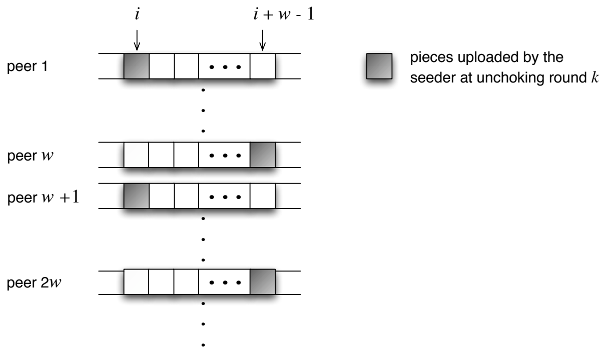


Figure 4: Seeder's piece allocation with $g = \frac{\nu_s}{w}$ groups of peers.

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. 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 this 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 [?]. 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 would 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 Evaluation

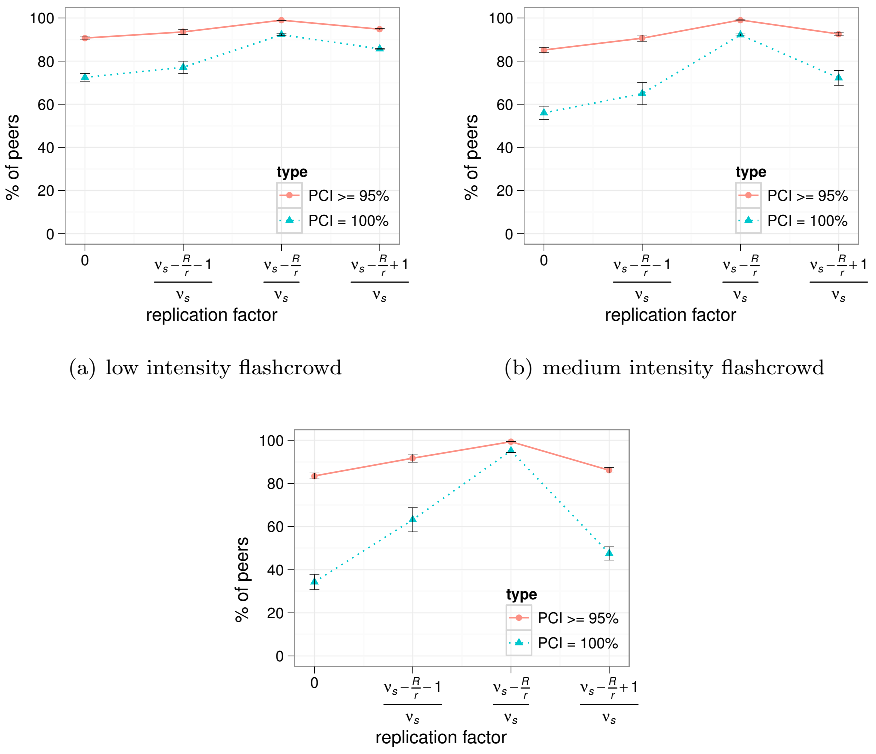
In this section, we evaluate our proposed flashcrowd-handling algorithms by means of simulations. First, we introduce the details of the experimental setup, the evaluation metrics, and we describe the different flashcrowd scenarios used. Then, we present and analyze the simulation results.

6.1 Experimental Setup

We have implemented a default BitTorrent-like VoD algorithm and our flashcrowd-handling algorithms on top of the MSR BitTorrent simulator [2]. This discrete event-based simulator accurately emulates the behavior of BitTorrent at the level of piece transfers and has been widely used, also for simulating BitTorrent-like VoD protocols [13, 15]. In all our experiments we have utilized the algorithm presented in [13] as our default BitTorrent-like VoD protocol, with tit-for-tat as peer selection policy and local rarest-first within the buffer as piece selection policy[2]. We have set the flashcrowd detection threshold value T to 0.5, since our simulations show that, in normal state, the fraction of a peer's neighbors having less than 50 % of the file lies, on average, below 0.5 (see Figure 3(b)). Different threshold values will be explored in future work.

The settings for our experiments are shown in Table 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

[2] The VoD protocol presented in [13] employs an adaptive mechanism to increase a peer's buffer 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 buffer size B" reported in Table 2 represents the default initial size of each peer's buffer.



(a) low intensity flashcrowd

(b) medium intensity flashcrowd

(c) high intensity flashcrowd

Figure 5: Percentage of peers experiencing perfect playback continuity (PCI = 100%) and good playback continuity (PCI ≥ 95%) in a system hit by flashcrowds of different intensities as defined in Section 6.3. The graphs compare the performance of the flashcrowd-handling algorithms with different replication factors. The vertical bars represent the confidence intervals over 10 simulation runs. Note that the scale of the horizontal axis is not linear.

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 [6] 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.

6.2 Evaluation Metrics

To evaluate how well our solutions meet the QoS requirements for VoD, we have utilized the following metrics:

1. *Playback Continuity Index* (PCI): defined as the ratio of pieces received before their playback deadline over the total number of pieces. The higher a peer's PCI, the smoother the playback it experienced. Hence, the PCI measures how well the QoS Requirement 1 is met.

2. *Startup delay*, to measure how well the QoS Requirement 2 is met.

6.3 Scenarios

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

Table 2: Simulation Settings

Parameter	Value
Flashcrowd size N	1500 peers
Video playback rate R	800 Kbits/s
Video length L	1 hour
Initial buffer 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

- *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 $\tau = \frac{N}{\lambda_0} = 150$;
- *high intensity*: N peers joining altogether at time t_0.

6.4 Results

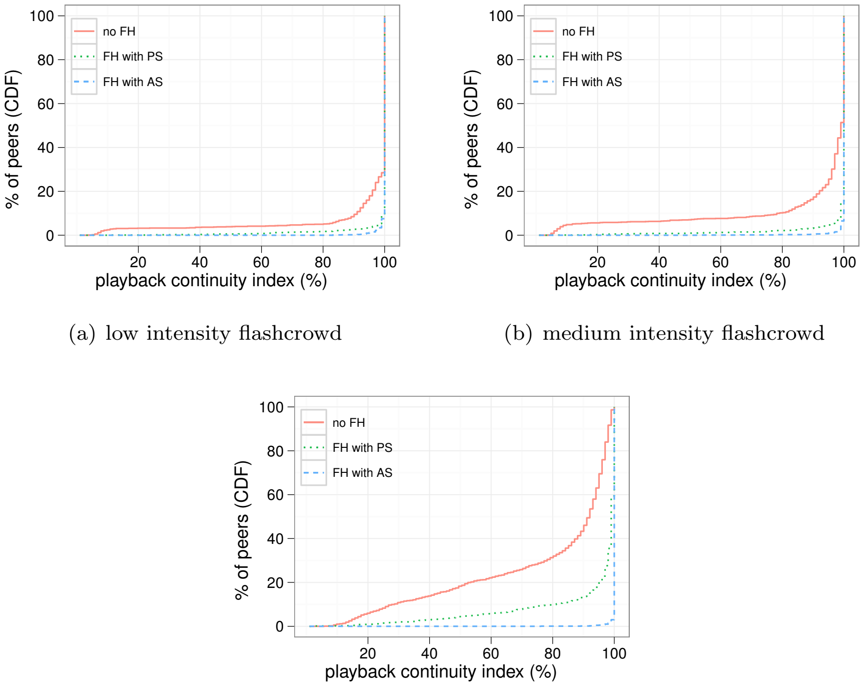
We will first analyze the effect of different replication factors F_k over the performance of our flashcrowd-handling algorithms and then we will compare the default BitTorrent-like VoD algorithm with our flashcrowd-handling algorithms.

The effect of different replication factors

Figure 5 shows the percentage of peers experiencing perfect (PCI = 100%) and good (PCI ≥ 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 4, a higher piece replication at the seeder decreases the chance of peers missing pieces. However, we have also showed that the seeder replication shall 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 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.

Default algorithm vs flashcrowd-handling algorithms

Figure 6 shows the CDF of peer playback continuity index for the default BitTorrent-like VoD algorithm and our flashcrowd-handling algorithms under the three simulated scenarios. The algorithm with active seeding pictured 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 with 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 PCI of 100%. Furthermore, we can notice that, while the performance of the other two algorithms degrades with more intense flashcrowds, that of FH with AS stays constant. Finally, we note that the flashcrowd-handling algorithm with passive seeding (FH with 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



(a) low intensity flashcrowd

(b) medium intensity flashcrowd

(c) high intensity flashcrowd

Figure 6: CDFs of peer's playback continuity index in a system hit by flashcrowds of different intensities as defined in Section 6.3. The graphs compare the performance of the default algorithm without flashcrowd-handling (no FH) with the flashcrowd-handling algorithm with passive seeding (FH with PS) and active seeding (FH with AS), respectively. The active seeding algorithm uses the maximum replication factor according to Eq. (4).

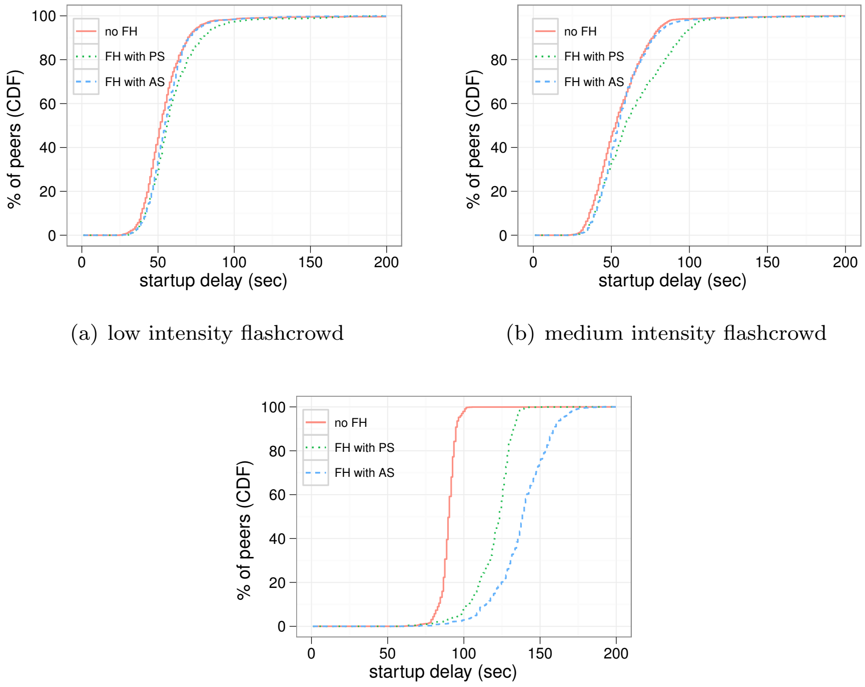
do not coordinate their piece requests among each other. The local rarest-first strategy used by each peer to select a piece to download is supposed to smoothen this effect. However, since its effectiveness builds up once a peer has been in the system for some time, it is less powerful when the system is under a heavy flashcrowd.

For what concerns the startup delay (Figure 7), we can make the following observations. First we note that, for a flashcrowd with low or medium intensity, FH with 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 with AS significantly increases the startup delay of peers in the scenario of heavy flashcrowd. This is an experimental validation of what 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 PCI and startup delay, respectively.

7 Conclusion and Future work

In this work, we have studied the allocation of bandwidth in a BitTorrent-like VoD system under flashcrowd. We have defined what the priorities are when bandwidth is scarce, so to provide a good QoS to as many peers as possible. In doing so, we have shown that there is an upper bound for the number of peers that can be admitted in the system in time. Furthermore, we have demonstrated that a trade-off exists between



(a) low intensity flashcrowd

(b) medium intensity flashcrowd

(c) high intensity flashcrowd

Figure 7: CDFs of peer's startup delays. The notations and the setups as the same as for Figure 6.

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 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-like VoD 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-like (incentivized) systems.

There are several directions for further studies. For example, it would be desirable to consider the effect of early peer departures due to users impatience in getting access to the video content. Furthermore, it might be interesting to dynamically adjust the capacity provisioning of the service provider to adapt to the size of flashcrowd. This will also require a deep investigation of different flashcrowd detection techniques.

References

[1] A. Vlavianos, M. Iliofotou and M. Faloutsos. BiToS: Enhancing BitTorrent for Supporting Streaming Applications. In *IEEE Global Internet Symposium*, 2006. 1, 2

[2] A.R. Bharambe, C. Herley and V.N. Padmanabhan. Analyzing and Improving a BitTorrent Networks Performance Mechanisms. In *IEEE INFOCOM*, April 2006. 1, 11

[3] D. Wu, C. Liang, Y. Liu, and K. Ross. View-Upload Decoupling: A Redesign of Multi-Channel P2P Video Systems. In *IEEE INFOCOM*, 2009. 2

[4] F. Esposito, I. Matta, B. Debajyoti, P. Michiardi. On the Impact of Seed Scheduling in Peer-to-Peer Networks. *Technical Report, Boston University*, 2010. 1, 2, 5, 8, 9

[5] F. Liu, B. Li, L. Zhong, B. Li, D. Niu. How P2P Streaming Systems Scale Over Time Under a Flash Crowd? In *IPTPS*, 2009. 1, 2

[6] N. Carlsson and D. L. Eager. Peer-assisted on-demand Streaming of Stored Media using BitTorrent-like Protocols. In *IFIP NETWORKING*, 2007. 1, 2, 10, 12

[7] N. Magharei, R. Rejaie, Y. Guo. Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches. In *IEEE INFOCOM*, 2007. 6

[8] 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. 1, 2

[9] P. Garbacki, D.H.J. Epema, J.A. Pouwelse and M. van Steen. Offloading Servers with Collaborative Video on Demand. In *IPTPS*, 2008. 6

[10] 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. 1, 2, 6

[11] P. Shah and J. F. Pris. Peer-to-Peer Multimedia Streaming using BitTorrent. In *IEEE IPCCC*, 2007. 1, 2

[12] 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. 2

[13] Y. Borghol, S. Ardon, N. Carlsson and A. Mahanti. Toward Efficient On-Demand Streaming with BitTorrent. In *IFIP NETWORKING*, 2010. 1, 2, 6, 9, 11

[14] Y. Lu and J.J.D. Mol and F. Kuipers and P. van Mieghem. Analytical Model for Mesh-based P2PVoD. In *IEEE ISM*, 2008. 2

[15] Y. Yang, A.L.H. Chow, L. Golubchik and D. Bragg. Improving QoS in BitTorrent-like VoD Systems. In *IEEE INFOCOM*, 2010. 11

[16] Y.Huang, T.Z.J. Fu, D.H. Chiu, J.C.S. Lui and C. Huang. Challenges, design and analysis of a large-scale p2p-vod system. In *ACM SIGCOMM*, 2008. 1

Appendix

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. (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

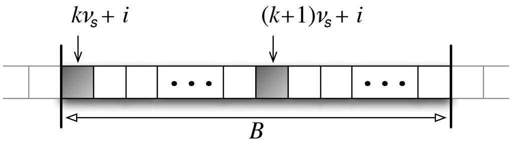


Figure 8: Representation of peer i's current buffer $\mathcal{B}_\alpha$. In this example, $B = 2\nu_s$ and the seeder allocates to peer i the pieces $k\nu_s + i$ (the colored ones in the figure), with k positive integer. Thus, the seeder allocates to peer i a total of $S_B = \frac{B}{\nu_s} = 2$ pieces.

results in the seeder uploading, to each peer at each timeslot, an average number of pieces within the buffer equals to

$$\frac{S_B}{T_B}. \tag{10}$$

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 (see Figure 8), then we have

$$S_B = \frac{B}{\nu_s}. \tag{11}$$

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

$$d_B \approx d_B(t_k) = X_s(t_k) + \frac{S_B}{T_B(t_k)}. \tag{12}$$

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

$$T_B \approx T_B(t_k) = \frac{B - S_B}{X_s(t_k)} = \frac{B(\nu_s - 1)}{\nu_s X_s(t_k)}. \tag{13}$$

Plugging into Eq. (10) the expressions for S_B and T_B as calculated in Eqs. (11) and (13), gives

$$\frac{S_B}{T_B} = \frac{\frac{B}{\nu_s}}{\frac{B(\nu_s-1)}{\nu_s X_s(t_k)}} = \frac{X_s(t_k)}{\nu_s - 1}, \tag{14}$$

which concludes our proof. □