Serving Niche Video-on-Demand Content in a Managed P2P Environment

Eli Brosh[1], Chitra Agastya[2], John Morales[2], Vishal Misra[1], Dan Rubenstein[1]
affaddr Department of Computer Science, Columbia University
[1] {elibrosh, misra,danr}@cs.columbia.edu
[2] {csa2111, jm2873}@columbia.edu

ABSTRACT

A limitation of existing P2P VoD services is their inability to support efficient streamed access to niche content that has relatively small demand. This limitation stems from the poor performance of P2P when the number of peers sharing the content is small. In this paper, we propose a new provider-managed P2P VoD framework for efficient delivery of niche content based on two principles: reserving small portions of peers' storage and upload resources, as well as using novel, weighed caching techniques. We demonstrate through analytical analysis, simulations, and experiments on planetlab that our architecture can provide high streaming quality for niche content. In particular, we show that our architecture increases the catalog size by up to 40% compared to standard P2P VoD systems, and that a weighted cache policy can reduce the startup delay for niche content by a factor of more than three.

1. INTRODUCTION

Video-on-demand (VoD) is an attractive service that has already gained popularity in the Internet [10] by allowing users to view a video from a catalog of popular choices at any time. However, current VoD design requires a large amount of costly server resources and significant bandwidth to support its users. The peer-to-peer (P2P) approach has been proven to be an effective solution for scalable content distribution without imposing a significant burden on a centralized infrastructure [5, 15]. In a P2P VoD system, users receive streamed videos from VoD servers as well as from the peers. The ability of peers that are viewing the videos to collaborate with each other reduces the load on serving infrastructure.

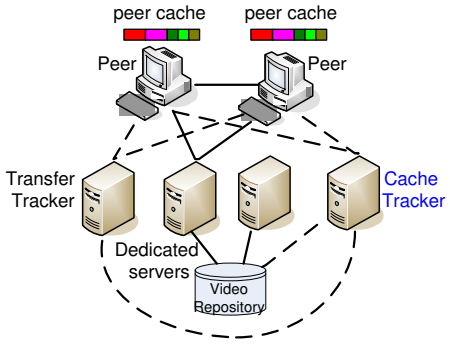
A limitation of existing P2P VoD services is their inability to support efficient streamed access to content throughout the full spectrum of popularity: not just the blockbuster releases but also *niche content* that has relatively small demand: re-runs of TV shows, replays of sporting events, or user-generated content. This limitation stems from the poor performance of P2P when the number of peers sharing the content is small [14] and the insufficient availability of niche content across the system [19]. While the ability to serve a single niche video is insignificant from both a resource-consumption and revenue-producing perspective, there is ample evidence [4] that there is a "long tail" of low-demand items, so that not serving them translates into lost revenues, and serving them using conventional client-server mechanisms is inefficient and costly.

In this paper, we propose a new cross-content P2P VoD framework that supports a large library of stored media. The proposed architecture is TiVo/cable-like, in that the peers remain owned and under the control of the VoD service provider. Our approach uses two key principles specific for the efficient delivery of niche content:

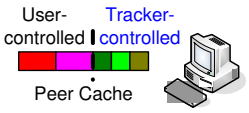
- *Cross-content caching and serving:* Unlike traditional P2P systems in which clients are greedy and participate mainly in transfer of content that is of immediate interest to themselves, our system requires clients to contribute a portion of their resources strictly for the purpose of serving others. The large shared demand for popular content is handled by the greedy component, while the proportionally smaller demand for unpopular content is handled by the altruistic component.

- *Weighted caching strategies:* In VoD, earlier portions of videos must be retrieved shortly after the request for the video, whereas later portions can tolerate larger delivery delays. We reduce the expected delay of retrieving earlier portions by forcing a higher replication rate of the earlier video parts than the later parts in the niche caching component.

We use a combination of mathematical analysis, simulation, and prototyping to study how effective can a P2P VoD system be at supporting files across the full spectrum of popularity. Our analytical models yield the proper cache policy for niche content and an optimized strategy for sharing cache resources among multiple videos. The analysis coupled with simulation allows us to explore a wide range of the design space, while the prototype demonstrates a proof-of-concept.



(a) Overall



(b) Peer

Figure 1: High Level Architectural Views

Our work makes the following contributions:

- We develop a stochastic model that captures the performance of niche content delivery in P2P VoD environments (Section 4.1). We use the model to derive novel, weighted caching strategies. We evaluate our architecture via extensive simulations (Section 4.2) as well as with a prototype implementation (Section 4.3) and show that our caching policy significantly improves the streaming quality of unpopular videos. In particular, we show that weighted caching can reduce the startup delay requirement by more than three compared to uniform caching.

- We demonstrate that the cache distribution plays a less significant role as the popularity of the video increases and that it can lessen the impact of over aggressive piece selection policies (Section 5).

- We demonstrate the ability of content providers to support much larger catalogs for VoD (Section 6). In particular, we show that our architecture increases the catalog size by up to 40% compared to standard P2P VoD systems. In addition, the NaDa project [1] is postulating enormous energy savings in a managed P2P architecture for VoD delivery as compared to a traditional monolithic datacenter or CDN approach.

2. SYSTEM ARCHITECTURE

We now describe the architecture of our system. We consider a VoD system based on client's boxes that cooperate in a peer-to-peer manner. Each client's box implements the VoD functionality and remains owned and under the control of the provider, e.g., like a cable set-top box, a "triple play box" or TiVo box [2]. This control enables the provider to implement optimized replication strategies and manage the content stored at the peers' caches, improving the performance of the overall system, rather than just benefiting the local peer. Since we consider a controlled environment, client's boxes can remain powered and connected to the network, even when not actively watching a video. The peers' connectivity can be taken advantage of to provide storage and forwarding within the P2P setting.

Our system incorporates the successful and widely used BitTorrent protocol [8]. With BitTorrent, peers viewing the same video are organized into a *swarm* and collaborate to distribute the video. A swarm consists of peers which are downloading (viewing) the video, called *downloaders* and of peers which have finished downloading, called *seeds*. Figure 1(a) presents a high-level view of our provider-controlled VoD architecture. Solid arrows indicate exchange of actual video information and dashed arrows represent exchange of control information. A large video repository chops each video into *pieces*, and these pieces are distributed by dedicated servers across peer nodes, who are then able to exchange these videos on demand. The servers also help with the on-demand distribution when necessary. A transfer tracker has functionality similar to what exists in BitTorrent's tracker. It helps peers locate other peers who have cached the content which they seek [16]. An additional *cache tracker* is employed whose job is to monitor the state of the peer's cache and push as well as redistribute the various video pieces across the peers and servers. Peers that were seeded with parts of the video can also act as seeds and upload video pieces.

Figure 1(b) depicts the architecture of a peer, focusing specifically on the peer's cache. The cache is split into two parts:

- A part, called the **personal cache component (PCC)**, that remains under complete control of a user and stores the recently watched videos, like part of a TiVo system. This part is necessary to allow users to instantly access a video rented for some specific time period, or for services such as rewinding.

- A second part, which is kept hidden from the user, called the **niche cache component (NCC)**. This part is seeded with video pieces according to the cache tracker's instructions. The video content can be pushed to the peers' NCC before the video distribution begins, e.g., during off-peak hours. The inclusion of an NCC deviates from traditional VoD architecture.

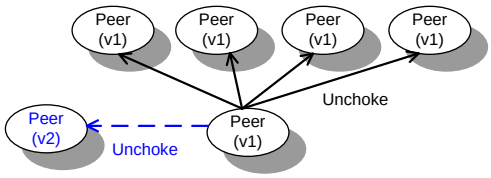


Figure 2: Traditional and modified choke

2.1 A simple BitTorrent extension

We make a simple modification to the BitTorrent protocol so that peers participate not only in exchanging content in their PCC, but also serve users whose requests are serveable from the NCC. Traditionally, a peer who is downloading a file, or video, V_1 directs all of its resources toward the active swarm [5, 9]. It uploads (unchokes) to several other peers. Posing a limit on the number of concurrent peers being served ensures an acceptable startup delay and low protocol overhead [7]. Periodically, a new peer is chosen at random to upload to, as a way to search 'good' new peers.

Figure 2 depicts our extension. We use an existing connection also for an alternate video V_2 that is not the focus of the primary swarm, but is within the (NCC or PCC) cache of the uploading peer. In-line with BitTorent, we periodically select a peer to unchoke. A random selection is made among the interested peers, such that the upload is directed to the active swarm for a fixed portion of the time, and shared among the alternate swarms throughout the remaining time. This behavior is very useful to peers who are interested in niche videos where the number of peers is too small to cooperate efficiently. A shown in Section 6, the intrusion of the additional swarms does not significantly affect the performance of a popular, active swarm, served from the PCC, as such a swarm is often over provisioned.

3. GENERAL MODEL

We begin by presentation a formalization of our system that will enable us to formally state our performance metrics and optimization objectives. We consider a P2P on-demand streaming system with a varying number of active peers and multiple files with different popularity. For ease of exposition, we first focus on the service model for a single video file, and further generalize the service model to account for multiple files of different popularity in Section 6.

The target video file is divided into m fixed-size pieces, and is encoded for playback at rate s. There are multiple peers participating in a distribution swarm. Peer j provides an NCC of size a_j that can be used by the system to store videos, such that the aggregated NCC has size $\sum a_j$. Peer j has an average upload rate of μ_j and download at rate c_j. Unless otherwise specified, we generally assume a homogeneous model with a_j, μ_j and c_j respectively equalling fixed values a, μ and c. Similarly to other P2P studies [20], we assume that the upload capacity of peers, μ_i, is the primary bandwidth bottleneck. The limit on upload capacity is often an artifact of the last-mile technology (e.g., cable, DSL), but can also be imposed by the peer so as to keep bandwidth available for other tasks. We summarize the parameters used in our system model Table 1.

Notation	Definition
r_i	Number of replicas of piece i
p_i	Playback continuity of piece i
t_i	Playback time of piece i
w_i	Playback completion time of piece i
P_a	Upload availability of a peer
N	Number of seeding peers
d	Startup delay before playback commences
r	Number of replicas of a given video
s	Media playback rate
μ	Uploading bandwidth of a peer
c	Downloading bandwidth of a peer
a	Size of NCC provided by a peer
z_i	Cache capacity allocated to video i
λ_i	Arrival rate of requests for video i
l_i	Size of video i
η_i	Swarming effectiveness of video i
γ_i	Seed residence time for video i
T_i	Playback completion time of video i

Table 1: Summary of model notations

3.1 Performance Metrics

Our general objective is to obtain seamless, perfect playback once the video starts playing. Often the user is willing to tolerate a short pre-buffer time, presumably on the order of seconds, which can vastly improve the remaining viewing experience. We let d represent the startup delay, the time since arrival until a peer begins playback. A small d is of course preferred. A piece that arrives later than its playback time leads to a playback glitch. To achieve seamless playback for a media playback rate of s pieces per time unit, piece i must be in the peer's buffer by time $d + i/s$

We focus on two natural performance measures, based on two different forms of video playback:

- *Skip-missing-piece*: playback skips over any piece that is not available by its playback time. We define the playback *continuity* of piece i, denoted by p_i, as the probability that piece i is received in time for playback. We use the fraction of pieces that arrive in-time to measure the performance. A simple objective is then to minimize the loss rate of the highest-lost piece, $\max_i \min p_i$.

- *Waitfor-missing-piece*: when a piece is missing,

playback stops until the piece is available. We define the playback *completion time* of a piece i, denoted by w_i as the playback time of piece i relative to that of piece $i-1$. If t_i is the play time of piece i, then $w_i = t_{i+1} - t_i$, where $t_m = t_{m-1} + 1/s$ and the pieces of the file are enumerated $0, \ldots, m-1$. We use the time taken to complete the entire playback, t_{m-1}, as the performance measure[1]. Our objective is then to minimize the playback completion time of the video, $\min E[t_{m-1}] = \min \sum_{i=0}^{m-1} E[w_i]$.

The waitfor-missing piece is probably the more practical method, as it is generally preferred to pause rather than miss part of a stored video. When presenting results, we mainly focus on the waitfor-missing-piece metric. Nonetheless, the skipfor-missing-piece is more tractable to analysis due to the reduced dependence of piece's playback on previous piece playbacks, and thus can provide additional insight on playback behavior.

3.2 Cache Distrbution

To assist in formalizing a strategy for assigning pieces within peers' NCC we introduce the notation r_i for the number of global replicas of piece i of the video. Clearly, $0 \leq r_i \leq N$, where N is the number of peers in the network. A *cache distribution* refers to the values $\{r_i / \sum_j r_j\}$ over all pieces i. For instance, a uniform distribution would be one in which $r_i = r_j$ for all pieces $i \neq j$ in the video. To ensure high availability of video content, we follow the common practice in managed P2P VoD systems [7, 12], and store r replicas of a video in the NCC. The replication can be done as part of the off-line content injection process.

4. INTRA-VIDEO PIECE DISTRIBUTION

Our main goal is to study the cache tracker's algorithm that facilitates efficient delivery of niche content: where to put pieces of videos, and how many copies of each are needed, based on the playback position of a piece, and its video popularity.

To facilitate exposition of the design and analysis of the caching policy, we go through three typical scenarios. First, we analyze a single client accessing a single video (Section 4.1), the common case when the popularity distribution of videos exhibits a long-tail. Here, we focus on the *intra-video piece distribution*: how to distribute the pieces of a video as a function of cache size, number of peers, and their availability to maximize the viewing objective measures. Then, we analyze multiple clients all accessing a single video (Section 5). Here, we focus on the effect of a swarm's ability to self-serve, which enables more efficient transfer of early movie pieces from other peers rather than NCC, on the piece distribution. Finally, we analyze multiple clients accessing multiple videos and focus on the *inter-video piece distribution* i.e., how to share the NCC among multiple videos of varying popularity (Section 6).

[1]Observe that this time is different from the time taken to complete downloading and cannot be smaller than it.

4.1 Single Client

To derive the desired distribution of pieces for small swarms, we develop a simple homogenous model that captures the playback performance, i.e., the portion of late pieces and the playback completion time, for a single client accessing as single video (see Section 4). The single client assumption greatly simplifies the problem in that: (a) all pieces are retrieved from peers' NCC (b) there is no benefit to using a 'rarest-first' policy [8] to select which piece to request first (i.e., selecting the piece that is rarest in the system), since the downloader has no peers to trade pieces. Hence, we can assume that an 'earliest-first policy' is implemented and that pieces are selected in strict, sequential order.

We perform the analysis within the context of time intervals, or rounds. The duration of a round corresponds to the time it takes to playback a single piece. A peer joins the system with no pieces locally available. It may buffer data for d rounds before commencing playback. The peer departs the system as soon as it completes the video playback.

A key challenge in modeling this system is emulating the behavior that the peers whose NCC contains the video's pieces are also participating in their own swarms of interest, or are busy serving other videos. Our approach is to capture this behavior using a simple availability model in which each round, when a peer is issued a request for upload of a piece in its NCC, it is available to serve that request in that round with a probability $P_a < 1$. Within a round, the peer can determine the availability of every uploading peer. It can then schedule for download the most needed pieces (those closest to playback deadline) for simultaneous download within the round that fit under its download constraint and the uploading peer upload constraint.

4.1.1 Playback Continuity Model

To derive the desired cache distribution, we first model the playback performance of a single downloader in distribution swarm with N seeding peers. The state-space of our model consists of $\{(i,k) | i = 0, \ldots, m-1; k = 0, \ldots, m+d-1\}$ where i is the piece index, k is the current round, d is the startup delay in rounds, and m is the number of pieces of a target video v. The states $\{(.,k) | k = 0, \ldots, d-1\}$ correspond to the rounds a newly-arrived peer buffers data before commencing playback.

We denote by $p(i,k)$ the probability that a downloader at round k has successfully acquired piece i. We assume the method of playback is skip-missing-piece

and follow the definitions in Section 3.1. We denote by $p_i = p(i, i+d)$ the playback continuity of piece i, i.e., the probability piece i has been acquired before its deadline $i+d$. For simplicity of analysis, we assume that a peer can upload one piece per round and download a large number of pieces per round[2].

We adapt a similar methodology as that in [29] to compute $p(i,k)$ by determining its steady-state behavior. We define $q(i,k)$ as the probability that piece i is chosen for download at round k, $W(i,k)$ as the probability that the downloader does not have piece i at round k, $F(i)$ as the probability that at least one of the uploading peers with piece i is available, and $S(i,k)$ as the probability that piece i is the most needed one at round k. We can express $p(i,k)$ as

$$p(i,k) = p(i,k-1) + \begin{cases} q(i,k) & \text{if } k \leq i+d \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

with $q(i,k)$ as the product of three components:

$$q(i,k) = W(i,k)F(i)S(i,k) \quad (2)$$

$$W(i,k) = 1 - p(i,k-1), \qquad F(i) = 1 - (1-P_a)^{r_i}$$

$$S(i,k) = \prod_{j=\max(0,k-d)}^{i-1} 1 - \frac{F(j)}{P_a N}(1 - p(j,k-1))$$

where r_i is the number of seeds that have cached piece i. $S(i,k)$ is determined as the probability that the uploader has no useful piece j with an earlier deadline than i. The probability that a downloader wants piece j is $1 - p(j,k-1)$. The probability that an uploader is selected to serve piece j is the probability that at least one uploader with piece j is available $F(j)$, normalized by the average number of serving peers in the system $P_a N$. With a skip-missing-piece playback, pieces are requested only if they can be received before their scheduled deadline, yielding that $q(i,k) = 0$, $\forall k : k > i+d$. The boundary condition of Eq. (1) is $p(i,k) = 0, \forall i,k : k < 0$.

This formula simplifies to

$$p_i = 1 - \prod_{k=0}^{i+d}(1 - F(i)S(i,k)), \quad (3)$$

where $F(i)$ and $S(i,k)$ are defined in (2). The simplification is given in the Appendix. Intuitively, it can be understood as follows. The playback continuity is the probability a piece is obtained in a sequence of at most $i+d+1$ rounds. The probability of obtaining piece i in a round is $F(i)$, and the probability piece i is the most needed one in the kth round is $S(i,k)$.

[2]We observe that the results provided by our model are in agreement with measurements when a peer's download rate is twice or more its upload rate, a common assumption P2P in studies [15].

4.1.2 Playback Completion Time Model

We generalize the previous model to account for waitfor-missing-piece playback, the more practical metric. We consider a distribution swarm with single downloader and N seeds. The state-space of our model consists of $\{(i,k,s)|i,s = 0,\ldots,m-1; k = 0,\ldots,4m-1\}$ where i is the piece index, k is the time to complete the playback of piece i, s is the peer's playback position, and m is the number of pieces of the video. We assume $k < 4m$, as a conservative bound on the time to complete the video playback. We follow the notations in Section 3.1 and define t_i as the playback time of piece i, and $w_i = t_{i+1} - t_i$ as the playback completion time of piece i. The latter time consists of the time playback is stalled due to a missing piece and a piece's playback time. The playback completion time of the entire video can thus be expressed as $E[t_{m-1}] = d + \sum_{i=0}^{m-1} E[w_i]$. Seamless playback is obtained when $E[t_{m-1}] = m + d$.

We define $c_i(k)$ as the probability that playback is continuous ($k = 0$) or held at position corresponding to piece i for k rounds ($k > 0$), and $p_i(i)$ as the probability that piece i is available when playback advances to position i. We build on the model in Section 4.1.1 to derive the playback completion time of piece i in steady state:

$$E[w_i] = \sum_{k=0}^{2m-1}(k+1)c_i(k) = 1 + \frac{1 - p_i(i)}{F(i)} \quad (4)$$

$$c_i(k) = \begin{cases} p_i(i) & \text{if } k = 0 \\ (1-p_i(i))(1-F(i))^{k-1}F(i) & \text{otherwise} \end{cases}$$

Assuming playback is held at position i ($k > 0$), the playback completion time of piece i is modeled using a geometric distribution with parameter $F(i)$, the probability that at least one peer with piece i is available and is defined by Eq. (2). Computing $p_i(i)$ leads to a rather complex expression, which we present in detail in the Appendix.

We can simplify the expression for $p_i(i)$ by assuming that playback was not held for all the pieces before i. This assumption allows us to approximate $p_i(i)$, the probability of continuous playback of piece i, by p_i, the playback continuity of piece i for the skip-missing-piece method defined in Eq. (3). Applying the simplification, we get

$$E[w_i] = \frac{1 - p_i}{1 - (1-P_a)^{r_i}}. \quad (5)$$

The playback completion time of the video can thus be expressed as

$$E[t_{m-1}] = d + m + \sum_{s=0}^{m-1} \frac{1 - p_i}{1 - (1-P_a)^{r_i}}, \quad (6)$$

where the summation term represents the total amount of time playback is stalled.

4.1.3 Deriving a Good Cache Distribution

Using our derivation of p_i, we can derive closed-form bounds on the continuity and playback completion time metrics. First, we show that when the allocation of pieces to cache is uniform, i.e., $r_i = r_j$ for all i, j, then p_i is an increasing function of the piece position i [3], and $E[w_i]$ is a decreasing function of the piece position. The proof for p_i's monotonicity can be found in the Appendix, and that for $E[w_i]$'s monotonicity follows immediately from Eq. (4).

This results justifies the intuitive reasoning that earlier pieces are more likely to miss their deadlines than later pieces when the cache is uniformly allocated. Next, we can bound continuity using the functions previously defined:

PROPOSITION 4.1. *The playback continuity function p_i is bounded by*

$$(1 - F(i))^{i+d+1} \leq 1 - p_i \leq (1 - F(i)S(i))^{i+d+1} \quad (7)$$

where

$$S(i) = \frac{1}{i+d+1} \sum_{k=0}^{i+d} \prod_{j=\max(0,k-d)}^{i-1} 1 - \frac{F(j)}{p_a N}.$$

Theorem 4.1 indicates that the playback continuity function has an upper bound whose shape is exponential with respect to the piece's position in the video. Furthermore, numerical analysis shows that we can approximate the playback performance using the upper bound in Eq. (7), namely:

$$p_i \simeq 1 - (1 - P_a)^{r_i(i+d+1)}. \quad (8)$$

We use the above approximation to simplify the playback completion time expression in Eq. (6) into

$$E[t_{m-1}] \simeq d + m + \sum_{s=0}^{m-1} \frac{(1 - P_a)^{r_s(s+d+1)}}{1 - (1 - P_a)^{r_s}} \quad (9)$$

Assuming a skip-missing-piece method, an optimal replica distribution function can be obtained by solving $\max_i \min p_i$, as described in Section 3.1. We derive an approximation to the optimal distribution, in which all pieces are equally likely to be retrieved in time, by setting all $p_i = D$ for some constant D where p_i is defined in Eq. (8). Solving for r_i, we get

$$r_i = \frac{\log(1 - D)}{\log(1 - p)(i + g + 1)} \quad (10)$$

Since the size of the aggregate cache reserved for a video is bounded by $R = \sum_{i=0}^{m-1} r_i$, we have

$$r_i = \frac{R}{(i+d+1)\sum_{j=0}^{m-1} \frac{1}{j+d+1}} \approx \frac{R}{(i+d+1)\log m}. \quad (11)$$

[3]Note this result is not trivial, as the priority of serving a piece is inversely proportional to its deadline requirement.

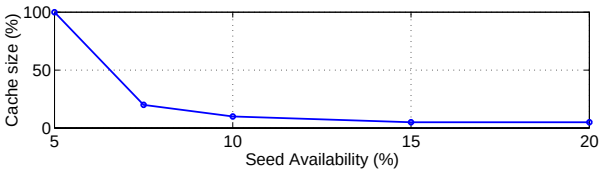


Figure 3: Cache sizing as function of availability.

This result shows that the number of replicas of a piece needs to be inversely proportional to its playback deadline. We can use the same methodology to derive an approximation for the optimal cache distribution under waitfor-missing-piece. Due to the complexity of the completion time expression in Eq. (9) we revert to numerical analysis. As before, the resulting cache distribution is inversely correlated with the piece position.

Observe that the larger we make the cache, the more a video can be globally replicated, the greater its availability within the system. Hence, given we know the right distribution, how large must the cache be for sufficient availability? Since we derived explicit expressions for the playback performance as a function of peer availability, P_a, we simply need to determine a desirable playback performance, and given the availability rate, we can compute the cache size. For example, Figure 3 shows the cache sizing as function of peer's availability for 20 peers, each with cache capacity of a single video, and a target playback completion time that is 1.05 times the video's length.

4.1.4 Two-value Availability Model

So far, we have overlooked the typical scenario where peers that are actively watching a movie are likely to offer less availability to alternate swarms than inactive peers who effectively serve as a seed for all videos stored in their NCC. To be able to compute the right cache distribution for this scenario, we generalize our model to account for two classes of peers: viewing peers who offer availability P_{a_1} and seeding peers who offer availability P_{a_2}, $P_{a_1} < P_{a_2}$. It is easy to see that structure of the expression for the playback completion time in Eq. (4) and that for the playback continuity in Eq. (1) is not altered by the introduction of the two classes of peers. However, the probability of finding an available seed with piece i, $F(i)$, and the probability that i is the most needed piece, $S(i,k)$, is now expressed as:

$$F(i) = 1 - (1 - P_{a_1})^{r_{i_1}} (1 - P_{a_2})^{r_{i_2}}$$

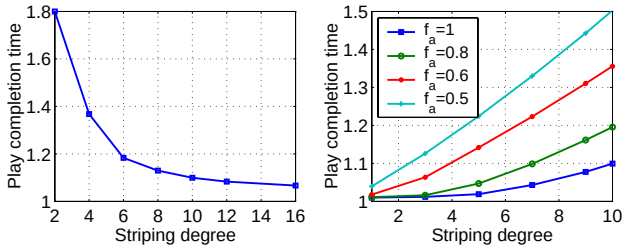
$$S(i,k) = \prod_{j=\max(0,k-d)}^{i-1} 1 - \frac{F(j)}{(P_{a_1} + P_{a_2})N}(1 - p(j, k-1)),$$

where r_{i_1} and r_{i_2} are the number of replicas of piece i across the viewing peers and seeding peers, respectively. We can use numerical analysis to derive a playback-

optimized cache distribution, as described in Section 4.1.3. The resulting cache distribution is inversely correlated with the piece position and is typically more flat than that for the single-value availability model. The flattening happens because the introduction of additional peers increases the availability of early pieces, and thus lowers the need for front weighting the cache distribution.

4.1.5 *Striped vs. Aggregate caching*

An important question we seek to address is how should a niche video be dispersed within a global, distributed cache allocated to it? To this end, we study the affect of a piece grouping strategy, that is, the relative placement of various pieces of a video across peers' NCCs, on the playback performance. We assume there are enough peers so that each peer holds no more than one replica[4]. For example, one strategy is to minimize the number of NCCs used to support a video by grouping all pieces of a replica within a single NCC, and another is to distribute (stripe) pieces among multiple peers.



(a) Fixed Availability (b) Proportional Availability

Figure 4: The playback performance vs. striping degree when peer's availability is (a) fixed; (b) inversely proportional to the number served videos.

For a fixed availability rate P_a, the playback performance increases with the number of peers serving a copy of the video, which we hereafter refer to as the striping degree. Figure 4(a) shows the playback completion time as a function of the striping degree when seed availability is $P_a = 0.1$ and the allocated cache in the NCC is double the size of the video. The wide striping is better because the client can receive copies from peers in parallel, so multiple peers being available to serve simultaneously adds value. However, the availability P_a may depend on how many different videos the peer is responsible for serving. We study this scenario by introducing a simple model for the availability a peer provides for a single video:

$$P_a = f_a \frac{r}{N} \tag{12}$$

where f_a is the probability a peer is available to serve any content from its NCC cache (see Section 2.1), r is the number of replicas of a video in the NCC, and N is the number of peers serving the video. Assuming each peer has an NCC cache capacity of a single video, the number of videos a peer serves is equivalent to the striping degree $S = N/r$. Hence, the availability is proportional to the fraction of upload service allocated to the video. Figure 4(b) shows the playback completion time as a function of the peer's total availability f_a and the striping degree S in a setting where $r = 2$ and $2 \leq N \leq 20$. We see that the playback performance degrades as the striping degree increases. By striping, the amount of NCC cache dedicated to the observed video reduces, and this reduction is significant to cancel the above noticed gains from striping.

Our goal is thus to identify the 'cross-point'. In other words, what functional behavior must P_a have with respect to the fraction of cache allocated to the video whereby the two caching strategies, wide striping and aggregate placement, offer identical performance? Since the biased cache distribution attempts to equalize the continuity of all pieces (see Section 4.1.3), it is sufficient to find the condition for which the continuity of the first piece is independent of the video's cache fraction N/r. To simplify the analysis, we seek to find the condition for which the continuity is independent of the number of serving peers N, assuming the number of replicas of the video r is fixed. We empirically observe that with a biased cache distribution the first piece is typically replicated across all serving peers. Setting the continuity of the first piece to some constant $C \leq 1$, and solving for P_a, where $r_0 = N$ and p_i is given by Eq. (8), yields P_a's functional behavior

$$P_a = 1 - (1 - C)^{\frac{1}{N(d+1)}}. \tag{13}$$

Thus, to achieve a target continuity C, the availability should be inversely correlated with the number of videos a peer serves as in Eq. (13). For example, the availability P_a should be nearly inversely proportional to the number of videos a peer serves to obtain some target continuity when the start delay is zero ($d = 0$). We can derive the cross-point for the waitfor-missing-piece playback by applying the same methodology. For $d = 0$, the functional behavior of P_a is the same as in Eq. (13), where $C = E[w_0]/(1 + E[w_0])$. We validate the cross-point by simulations (see Section 4.2). We consider a setting where $r = 4$ and each seed has NCC cache capacity of a single video. We vary the number of seeds between 4 and 60, so that a peer concurrently serves 1 to 15 videos, and adjust the availability according to Eq. (13). We measure the playback completion

[4]There is no benefit to having two identical replicas in a single peer's cache, since a single copy can be transmitted to multiple peers simultaneously

time and playback continuity of the first 10% and 50% of the video. We observe near constant performance with small deviations from the mean of less than 8%, thus providing empirical evidence to support the cross-point rule.

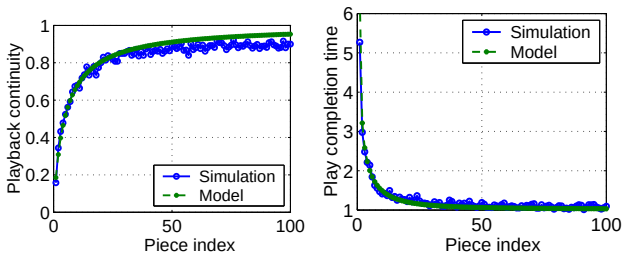
4.2 Simulation Experiments

To explore the parameter space and study the usefulness of the cache distribution proposed in Section 4.1.3, we conduct an extensive performance study using an event-driven simulator that emulates real BitTorrent behavior [6]. We enhance the simulator to support an earliest-first (EF) and rarest-first (RF) hybrid policy to select which piece to download first. With a hybrid policy [9], a peer randomly selects either the piece that is closest to the playback point or the one that is rarest in the system. The hybrid policy has been shown to achieve a good tradeoff between high piece diversity and sequential progress [9, 22, 29]. We use a pure earliest-first policy for the single client experiments, and a hybrid policy with an EF fraction of 0.8 for the multi client experiments in Section 5, as suggested by [17,25].

We further modify the baseline BitTorrent protocol to support cross-content caching and uploading. Cross-content uploading is done over a single connection: a peer selects to upload to (unchoke) the primary swarm with probability $1 - P_a$ and to the additional swarms for which it has content in its cache with probability P_a. We further modify the protocol to allow peers to be seeded with parts of a video file according to a given cache distribution. Unless specified otherwise, we use the following simulation settings to derive the simulation results. We consider a file of 25MB chopped into 100 pieces, each of 256KB. The upload and downland bandwidth of a peer is 1000Kbps and 5000Kbps, respectively, and the video playback rate is 1000kpbs. The unchoke interval is 2 seconds.

We consider a simple setup with two videos, A and B, being concurrently downloaded by $N + 1$ clients. Out of the $N + 1$ clients, N download video B and seed A, and one downloads A. An additional peer is used to seed B. Thus, the peers from two swarms: one for the popular video B with N downloaders and one for the unpopular video A with a single downloader and N peers seeded with parts of A. Each of the N seeding peers offers an availability of P_a for serving A's pieces. The aggregated cache for A across the seeding peers is r times the video size. In the beginning of the simulation, we iteratively distribute video pieces among peers, such that the number of replicas of each piece follows a target cache distribution function. In each iteration, we select a piece replica at random and assign it to the peer with the smallest number of assigned pieces. The process stops when all pieces are assigned.

We focus on the performance of the less popular video



(a) skip-missing-piece (b) waitfor-missing-piece

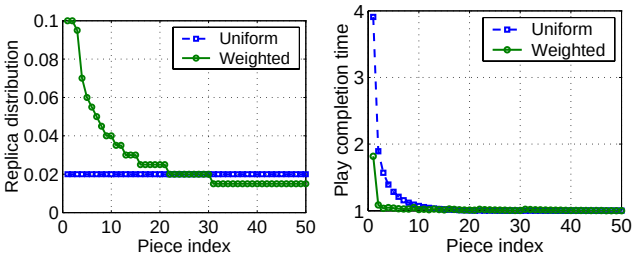
Figure 5: Model validation assuming playback (a) skips a missing piece; (b) waits for a missing piece.

A. For each set of parameters, we repeat the experiment 100 times and report the average results. When presenting results, the playback completion time of a video is expressed relative to the time length of the video. For example, a playback completion time of 1.25 means that the time it takes to playback the video is 1.25 times the video's length. In other words, the total time playback is stalled is 25% of the video length. The piece-level playback completion time, as well as the startup delay, are expressed relative to the playback time of a piece. We repeat each video download experiment 100 times, and typically present the average result in the plots.

4.2.1 Simulation Results

We begin by comparing the results obtained using our models to those obtained using the BitTorrent simulator. We consider a setting with 20 seeds ($N = 20$), seed availability of 0.05 ($P_a = 0.05$), 100-piece video ($m = 100$), zero startup delay ($d = 0$), and an allocated cache in the NCC that is four times the video size ($r = 4$), populated uniformly. Figure 5 shows the predicted vs. measured playback continuity (p_i) and playback completion time ($E[t_{m-1}]$) functions, respectively, as a function of the piece position. As described in Section 3.1, we use the playback continuity metric for skip-missing-piece playback and playback completion time metric for wait-missing-piece playback. We observe a good match with an average error of less than 5% between the predicted and measured playback continuity and playback completion time functions. We repeat the validation when the number of seeds is varied from 10 to 40, the NCC cache size is varied from 2 to 10 times the video size, the availability is varied between 0.05 and 0.2, and using a front-weighted cache distribution, all while keeping the remaining parameters fixed. We observe a similar match between the model and the measurement.

Figure 6 depicts experimental results that show the benefit of a biased distribution. Figure 6(a) depicts how we distribute replicas of pieces within a 100-piece video

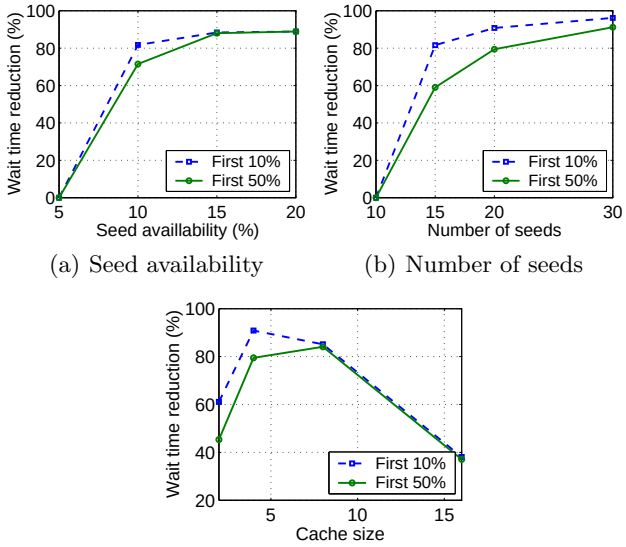


(a) Cache distributions (b) Playback times

Figure 6: Performance comparison between uniform and biased distributions using simulation.

when the allocated cache in the NCC is four times the size of the video (400 pieces), the number of seeds is 20, the availability is $P_a = 0.1$, and $d = 0$. For the uniform policy, there are four copies of each piece. As a second strategy, we implement a *biased* caching strategy according to the results in Section 4.1.1. The biased strategy reduces the number of replicas of pieces 62-100 to two to significantly increase the number of replicas of the first 15 pieces, and slight increases in the next 6. Figure 6(b) depicts the playback completion time of pieces relative to a single piece's playback time as a function of piece position. We see a gain of 30% in playback completion time of the first 10% of the video, i.e., the first 10 pieces, and a gain of 75% in the total time playback is stalled. This gain significantly improves the viewing experience. Intuitively, the benefit stems from the fact that the peers who can provide pieces are only available a fraction of the time, so that forcing a higher replication rate for the earlier pieces with tight retrieval deadlines at the expense of later ones that have relaxed deadlines improves performance.

Figure 7 plots the relative reduction in the time playback is stalled across the first 10% and 50% of the video of a biased cache distribution in comparison to uniform distribution. Figure 7(a) shows the performance improvement as a function the availability of seeds, Figure 7(b) as a function of the the number of seeds, and Figure 7(c) as function of the number of replicas cached in the NCC. We observe significant gains when seed availability ranges from 10% to 20% and when the number of seeds is between 15 and 30. When the piece availability is too scarce, the timely playback of later parts of the video becomes a bottleneck. Hence, shifting replicas from later parts to earlier ones does not benefit the playback completion time, rendering the biased distribution ineffective. We see significant gains even when the cache allocated to the video is minimal (e.g., twice the video size). The results for the skip-missing-piece metric are similar. We repeat the experiments for videos with 300 and 500 pieces and for small swarms with 4 or less clients (see also Section 5) and observe slightly



(a) Seed availability (b) Number of seeds

(c) Cache size

Figure 7: Benefit of a biased distribution in reducing the time playback is stalled when varying (a) availability; (b) the number of seeds; (c) cache size.

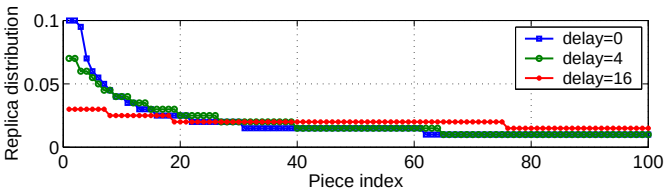


Figure 8: The affect of the startup delay on the optimized cache distribution.

smaller gains.

Increasing the tolerance for the initial startup delay d lessens the demand for the immediate availability of the early pieces, and thus can have a flattening affect on the cache distribution. This behavior is seen in Figure 8 for startup delays of 0, 4, and 16 seconds. We see that the distribution plays a less critical role as the startup delay increases. We also explore the affect of the biased distribution on reducing the startup delay requirement. We measure the lowest possible startup delay a peer needs to choose such that the playback completion time is below some target threshold, and present the reduction in startup delay requirement of the biased distribution in comparison to uniform. We see that the biased distribution can reduce the start delay by 3.5 on average. In summary, the simulation results show that our cache design achieves lower startup delays and significantly improves the streaming quality of small swarms.

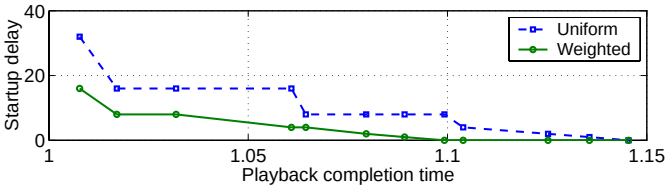


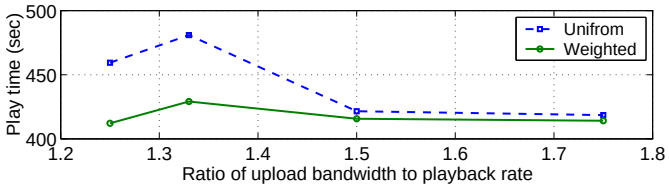
Figure 9: Benefit of a biased distribution in reducing the startup delay.

4.3 PlanetLab Experiments

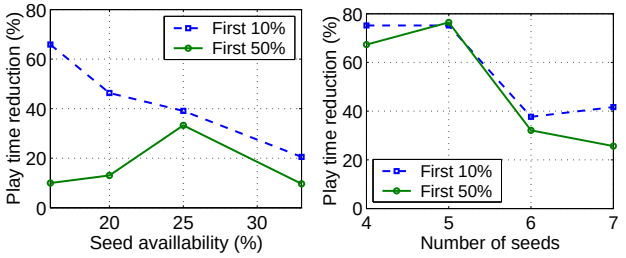
To demonstrate a proof-of-concept and to measure the performance of the biased cache distribution (see Section 4.1.3) in a real network environment, we develop a prototype of a P2P VoD system and conduct experiments on PlanetLab. We base the prototype on the mainline BitTorrent client and modify it to support in-order piece selection, cross-content serving and caching, and seeding of partial files, as in Section 4.2.

We setup two distribution swarms with 10 peers in total, in which one peer is downloading a niche video, another is seeding a popular video, and the remaining peers are downloading the popular video. The peers downloading the popular video are also set to seed the niche movie. We compare the playback performance of the niche video, i.e., playback completion time and continuity, for the biased cache distribution and uniform. In the experiments, we vary the the number of seeds for the niche video swarm from 4 to 7, the upload rate of peers from 768Kbps to 3072Kbps, and the maximum number of unchoked connections from 3 to 6. All except one connection are used for serving the swarm for the popular video. Out of U connections a peer unchokes, $U-1$ are reserved for the popular video's swarm. The other connection is used to serve the niche video's swarm with probability $1/(N-U)$, where N is the number of seeding peers. Thus, the availability offered by the seeding peers can vary from 0.16 to 0.33. The remaining BitTorrent parameters as set as follows. We consider a 25MB video split into 100 pieces, each 256KB in size. The playback rate is 512Kbps, so that the time to playback the entire video without interruptions is 400 seconds. The unchoke interval is 4 seconds.

Figure 10(a) shows the playback completion time for the biased cache distribution and the uniform distribution. We see that the gain is largest when the aggregated upload bandwidth is less than 1.5 times the playback rate, a common working point for VoD swarms [7]. Figure 10(b) and Figure 10(c) show the relative improvement in the playback continuity across the first 10% and 50% of the video for the biased distribution in comparison to uniform as a function of the availability of the seeds and the number of seeds, respectively. The gain is most drastic when seeds are rarely available, i.e., very active in their own swarm or serving other videos,



(a) Ratio of aggregated upload bandwidth and playback rate



(b) Availability of seeds (c) Number of seeds

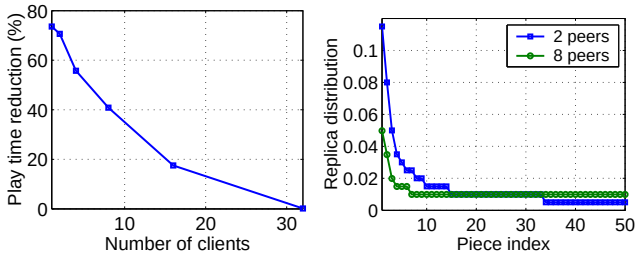
Figure 10: Performance comparison between biased and uniform cache design in PlanetLab experiments when varying (a) the ratio of upload bandwidth to playback rate; (b) availability of seeding peers; (c) number of seeding peers.

or when the number of peers actively seeding the video is minimal. In more realistic systems, where the availability of niche content can significantly vary (e.g., due to peer upload heterogeneity), we expect our cache design to have even larger impact.

5. MULTIPLE CLIENTS

We now explore the the impact of the size of the swarm on the cache policy. Our interest lies in videos that have high likelihood to be actively downloaded by multiple clients. That is, medium to large distribution swarms. We study the impact of the swarm size on the cache distribution using simulations (see Section 4.2). To derive the distribution of the cache, we develop a simple on-line algorithm that adapts the cache during successive transmissions of the video, where the playback times of the video pieces are monitored over time. The on-line algorithm does not require knowledge about node availability and the swarming mechanisms, which are often complex and co-dependant in large-scale distribution swarms [9,16], and hence is applicable for general VoD swarming systems. The basic idea of the algorithm is to attempt and shift replicas from pieces that have high continuity to pieces that have low continuity, which will have the affect of equaling continuity across all pieces. Further details are given in Section 5.1.

In the default setting, we consider a distribution swarm with multiple peers downloading video A and 40 peers who are not actively downloading A, but have space in their NCC to store it. Each such peer offers availability



(a) Playback improvement (b) Cache distribution

Figure 11: Benefit of the biased distribution in a multiple client setting.

of $P_a = 0.1$, and the aggregated cache is two times the video size. The peers interested in A arrive according to a poisson process. Figure 11(a) shows the relative improvement in playback completion time across the first 50% of the video for the biased cache distribution computed using the on-line algorithm in comparison to a uniform distribution. Figure 11(b) shows the biased cache distribution for swarms with 2 and 8 downloading peers. We see that the distribution of the cache plays a less significant role as the size of the swarm grows. Thus, the swarm becomes 'self-supporting'. The results are similar when considering a more general setting with two classes of peers: peers who serve pieces from their NCC, and peers who serve pieces from their PCC and typically offer high availability,e.g., 0.8 in our setting. We also compare the biased caching distribution in Section 4.1.3 based on the (weighted) average of peer availability to that derived using the on-line algorithm and observe similar distributions (up to 10% difference in placement) and playback times for small swarms, demonstrating the usefulness of the biased cache distribution for small swarms.

The results in Figure 11 can be explained by noting that the downloading peers are more aggressive in prioritizing the swarm than those acting as seeds, and thus piece availability is dominated by the downloading peers. Since the downloading peers are at some midpoint in the video playback, this naturally creates a higher density of earlier pieces in the swarm. Next, we study how adjustments in the hybrid piece selection policy (e.g., the fraction of time an RF-like request is made versus an EF-like request) affect the desirable distribution of pieces. We observe that a policy that prioritizes in-order delivery too aggressively can lead to a playback continuity that flattens out towards the end of the video [9]. In this case, the on-line algorithm yields a back-weighted cache distribution, which significantly improves the playback completion time, e.g., by 55% for a 64-peer swarm. Similarly, the cache distribution can compensate for piece selection policies that prioritizes rarest-first delivery too aggressively by weighting the front more heavily. In summary, we observe that the cache distribution flattens out as the swarm size increases, and that it can lessen the performance impact resulting from varying the EF/RF mixture in the hybrid piece selection policy.

5.1 On-line Caching Strategy

A drawback of our model-based caching policy is the required knowledge about node availability (P_a) and the swarming mechanisms in use. We thus present an adaptive algorithm that determines the cache distribution. The algorithm optimizes the playback performance, while adapting to variations in the availability of peers. Recall from Section 4.1.3 that our primary goal is to minimize the video's playback completion time: $\min_{\{r_i\}} E[t_{m-1}]$ subject to a constraint on the total cache size, $\sum_{i=0}^{m-1} r_i \leq R$, and a constraint on the number of replicas of a piece, $1 \leq r_i \leq N$.

We develop an algorithm that adapts the distribution of the cache during successive transmissions of the video, where the playback times of the video pieces are monitored over time. We let $\hat{t}_i$ represent the measured playback time of piece i and $\hat{w}_i = \hat{t}_{i+1} - \hat{t}_i$ represent piece i's playback time relative to that of piece $i-1$. We set our objective to minimize $\hat{t}_{m-1} = \sum_{i=0}^{m-1} \hat{w}_i$. The basic idea of the algorithm is to shift replicas from pieces that have high continuity to pieces that have low continuity. To this end, we order the pieces in ascending order according to their playback completion times $\hat{w}_i$. Assuming the aggregate size of the cache must stay fixed, whenever we find $\hat{w}_i \ll \hat{w}_j$ then we attempt to shift a replica of piece j to piece i, which can potentially decrease the playback completion time of the entire video.

We associate a timeout value for each pair of pieces participating in a replica shift to avoid frequent moves of the same pieces. We let $\{h_{i,j}\}$ represent the number of iterations of the algorithm for which a replica shift between j and i is prohibited, and decrease all entries $\{h_{i,j} > 0\}$ in every iteration. We restart the sampling for the piece playback times $\hat{t}_i$ every time the cache allocation is shifted for a piece. In addition, we periodically update $\hat{t}_i$ to their real measured values using a weighted moving average to allow the algorithm to track system dynamics. At initialization, we set the cache distribution to a uniform distribution $r_i = R/N$. The algorithm returns a new replica distribution candidate $\{\tilde{r}_i\}$ if one exists. The details of the cache optimization algorithm are given in Algorithm 1. Our algorithm can be easily extended to optimize the playback performance for skip-missing piece by measuring the continuity rate of pieces, and adapting the cache distribution to equalize the continuity across all pieces.

We compare the cache distribution based on the on-line algorithm to that based on our model for multiple simulation settings. We set the frequency with which

to invoke the replica shift to 50 successive downloads to accurately estimate peers' availability. We observe similarity in the distributions and the corresponding results for small swarms of three or less clients, namely, up to 10% difference in piece placement and 6% difference in the resulting playback performance. Typically, it takes the algorithm less than 200 iterations to compute the cache distribution.

Algorithm 1 On-line Cache Optimization

Given $\{\hat{t}_i\}$
Sort the pieces $\{i\}_{i=0}^{m-1}$ in ascending order according to their playback completion time
$x_k \leftarrow$ the piece that has the kth smallest playback completion time
if $\exists i,j$ such that $j = \arg\min_k\{x_k\}$ and $i = \arg\max_k\{x_k\}$ and $\hat{w}_{x_j} - \hat{w}_{x_i} > th$ and $h_{i,j} = 0$ and $h_{j,i} = 0$ and $r_j > 1$ and $r_i < N$ **then**
  $\{\tilde{r}_i\} \leftarrow \{r_i\}$
  $\tilde{r}_k \leftarrow \tilde{r}_k + 1$
  $\tilde{r}_j \leftarrow \tilde{r}_j - 1$
  $h_{i,j} \leftarrow m$
  return $\{\tilde{r}_i\}$
end if
return null

6. INTER-VIDEO DISTRIBUTION

Given a system with multiple videos of different popularity, how should peers share their bandwidth and cache resources across the various videos? To address this question we develop a model to characterize multi-file P2P VoD systems. We then use the model to derive an optimal cache allocation strategy across multiple videos, justifying the system's bias toward niche content in the form of the NCC.

More formally, we consider a P2P VoD system serving a set of videos $\mathcal{V}$. A video $i \in \mathcal{V}$ has size l_i and is requested at a rate of λ_i, with larger λ_i indicating a more popular video. The system has a total cache capacity of Z. The cache capacity allocated to video i is z_i. Different weight values $\{\alpha_i\}$ are assigned to the video files to indicate how important they are. For example, the weight of a video can be proportional to its popularity $\alpha_i = \lambda_i$, so that the service can be biased towards popular videos at the expense of niche ones. Observe that the heavy tail of demand for niche content means that their performance cannot be ignored. Alternatively, the weight can be constant $\alpha_i = \alpha$, so that all videos are treated equally.

6.1 Cross-content Optimization

Our primary goal is to minimize the playback completion time for a video request within our system, as described in Section 3.1. When considering all videos, we can specify our global objective as:

$$\min_{\{z_i\}} G = \sum_i \alpha_i T_i$$

$$\text{subject to } \sum_i z_i = Z$$

where α_i is the weight of video i and T_i is playback completion time of video i.

We make several simplification assumptions in our analysis, as follows. First, we use a parameter β to capture the ratio of the sequential progress of a video download, the ability to acquire pieces sequentially, and the download progress, the rate at which pieces are obtained. It has been empirically shown that for popular swarms, a BitTorrent-like protocol with in-order piece selection and FCFS upload queues can achieve near optimal download progress as well as ideal sequential progress [20], i.e., β close to 1. For unpopular content, β can be computed from our model in Section 4.1. We empirically observe that β decreases with the swarm size and therefore assume $\beta_i \geq \beta_j$ for $\lambda_i \leq \lambda_j$. The playback completion time can thus be expressed as $T_i = \max\{\frac{l_i}{s}, \beta_i T_i^{(d)}\}$, where $T_i^{(d)}$ is the download time of video i and $\frac{l_i}{s}$ is the video length. Second, we assume that each unit of storage is associated with a unit of bandwidth. This allows us to transfer the optimization problem into a bandwidth allocation problem.

We build on the results in [13,21] to develop a simple fluid model for the download time in a multi-file P2P VoD system with dedicated seeding peers. The seeds represent the service contribution of the NCC. We give a detailed description of the model in the Appendix. To keep the exposition simple, we consider here the case where peers do not abort the download and stop seeding after download is complete, which is a conservative assumption from provisioning perspective. Based on the download time expression in Eq. (27) in the Appendix, the playback completion time of video i in steady-state can be written as:

$$T_i = \max\left\{\frac{l_i}{s}, \frac{\beta_i}{\eta_i}\left(\frac{l_i}{\mu} - \frac{z_i}{\lambda_i \mu}\right)\right\} \qquad (14)$$

where z_i is the now the dedicated seed bandwidth assigned to video i and η_i is the effectiveness of swarming, that is, the probability that the upload capacity

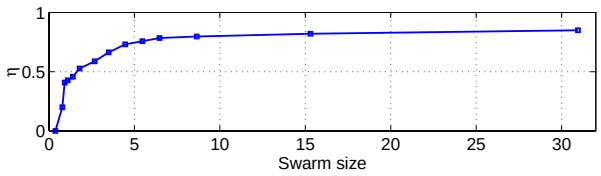


Figure 12: The effectiveness of swarming as function of swarm size.

of peers is fully utilized. We see that the reduction in download time due to the dedicated seed bandwidth $\frac{z_i}{\lambda_i \mu}$ decreases linearly with the arrival rate λ_i. In accordance with other P2P system studies [13, 16], we assume that $s \geq \mu$. That is, the peers' upload bandwidth is most certainly the constraint. Thus, the bandwidth allocated to video i should satisfy

$$z_i \leq \lambda_i l_i \left(1 - \frac{\eta_i \mu}{s\beta_i}\right). \quad (15)$$

We base the preference in bandwidth capacity assignment on the marginal utility of each video, the marginal improvement in the system-wide performance upon unit increase in capacity:

$$-\frac{dG}{dz_i} = \frac{\alpha_i \beta_i}{\eta_i \mu \lambda_i}. \quad (16)$$

To solve the optimization problem, we first allocate the capacity to v_1 until z_{v_1} is saturated. The residual capacity is then allocated to v_2, and the process repeats recursively. Hence,

$$z_{v_i} = \min \left\{ \tilde{z}_i, Z - \sum_{j=1}^{i-1} z_{v_j} \right\} \quad (17)$$

where $\tilde{z}_i = \lambda_i l_i \left(1 - \frac{\eta_i \mu}{s\beta_i}\right)$. This result indicates that the capacity allocated to a video should be proportional to the video popularity times $1 - \frac{\eta_i \mu}{s\beta_i}$, a factor that is inversely correlated with η. Thus, the capacity allocation per peer is higher for small swarms than that of a large swarm.

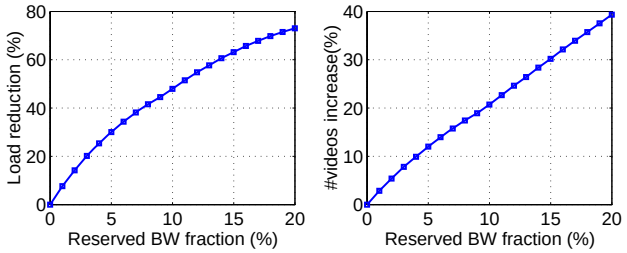
To determine the marginal utility, we observe that peers become more busy uploading as the swarm size increases [20]. For example, Figure 12 shows the measured value of η as function of swarm size in the experimental environment in Section 4.2. Thus, $\eta_i \leq \eta_j$ for $\lambda_i \leq \lambda_j$. This result combined with Eq. (16) allows us to determine preference in bandwidth capacity assignment: the capacity should be allocated according to videos popularity, *least popular videos first*. This allocation holds for both proportional video weights and uniform weights. The primary difference with respect to proportional is that the performance of relatively popular swarms is essentially independent of the allocation strategy. This occurs because once a swarm acquires few peers, the swarming effectiveness η increases very slowly with additional peers, so that the marginal utility of popular videos is essentially the same. In summary, the above analysis shows that the allocation of storage within the NCC should be proportionally to the the video popularity, but with a bias in favor of niche content. The bias is needed to compensate for the low swarming efficiency of small swarms.

We use a similar approach to demonstrate that reserving resources for niche content has small effect on the performance of popular swarms served from the PCC. Here, we assume that peers become seeds once they complete the download, serving the video for an average time of time $1/\gamma$. Based on the download time expression in Eq. (26) in the Appendix, we have

$$T_i = \max \left\{ \frac{l_i}{s}, \frac{\beta_i}{\eta_i} \left(\frac{l_i}{\mu} - \frac{1}{\gamma} \right) \right\} \quad (18)$$

This result indicates that if users keep videos in their PCC for relatively long times, the common case in our system, the seed residence time $1/\gamma$ is likely to be larger than the latency of uploading the entire file l_i/μ, so that the download time is not constrained by the peers' upload bandwidth. Hence, we can conclude that popular swarms have more than enough capacity that a small percentage of upload capacity can be sacrificed without noticeable effect on the performance.

6.2 Numerical Results



(a) Server load (b) Catalog size Availability

Figure 13: Performance improvement of our system in comparison to a traditional P2P VoD system in terms of (a) server bandwidth savings (b) catalog size increase.

To understand how efficient is an NCC-enabled system design, we calculate the system-wide performance numerically based on the equations provided in Section 6.1. We assume that we have a central server that can complement video streaming when the P2P network cannot satisfy a request. We focus on two performance measures: the server workload and the supported video catalog size, where a video is included in the catalog if its playback time is close to its original length l/s.

We consider a system with a content library of 1000 videos, each of which encoded at rate of $s = 1000$Kbps, and 1000 peers. The video popularity is modeled according to the Zipf's distribution with parameter $\theta = 0.75$. The Zipf distribution allows us to capture the long-tail phenomena typical in video rental services [4] and Web-based video services with user generated content [10]. Video i is requested at rate $\lambda_i = 1.25\frac{K}{i^\theta}$, where $K = \sum_{i=1}^{1000} 1/i^\theta$. We assume that on average a peer stays as a seed after completing the video download for $1/\gamma = 1.66$ hours. The swarming effectiveness and β

are derived empirically using experimentation (see Section 4.2).

We compare the server load and the supported catalog size for two designs: traditional P2P system (baseline), where peers use their entire upload bandwidth to serve all the video requests, and NCC-enabled, where a fraction $1 - f_a$ of a peer's bandwidth is dedicated to serving the 'top' videos and the remaining fraction f_a allocated to the rest of the videos according to the policy described in Section 6.1. The 'top' videos are the popular, overprovisioned videos in the system, i.e., those that do not suffer noticeable playback time decrease due to the reduction in the peers' upload bandwidth. For example, for $f_a = 0.2$, we have 250 top videos. Figure 13(a) shows the savings in server load of our system in comparison to baseline as a function of the fraction of peer's upload bandwidth reserved for niche content f_a. We see that the gain of an NCC-enabled system increases with the reserved bandwidth; the savings can go up to 70%. Intuitively, the improvement stems from the overprovisioning for popular content, while less popular content experiences low service (download) rates due to low swarming effectiveness and thus needs to rely more heavily on the central server to cope with the demand. Figure 13(b) shows the improvement in the supported catalog size for our system in comparison to baseline. As shown, the improvement increases linearly with the fraction of reserved bandwidth. While the baseline can support 650 videos, our architecture can reach 920 videos with $f_a = 0.2$, allowing nearly full access to the video catalog, as desired.

7. RELATED WORK

P2P VoD has attracted great research interests in recent years. Compared to traditional client-server VoD systems [3], a P2P-based VoD solution is less costly and more scalable [15, 15]. In the context of provider managed P2P VoD systems, there are few related papers. Janardhan *et al.* [18] proposed a P2P architecture for set-top boxes. Allen *et al.* [3] empirically evaluated the the benefits of using the storage of set-top boxes in a P2P fashion using a trace-driven approach. Suh *et al.* [24] designed an architecture for pushing content into boxes, and analyzed optimal placement strategies. Empirical evaluation of such an architecture for IPTV is provided in [12] and analysis of the achievable catalog size for such an architecture is given in [7]. Our work differs from these studies in that we focus on efficient delivery of the long-tail of unpopular content, rather than the popular videos.

In addition, a lot of recent work focuses on the design and enhancement of BitTorrent-like swarming protocols, designed originally for downloading systems [8,21], for live streaming systems [28, 29], and more recently for stored media streaming systems [5,20,25]. However, most of this work focuses on large-scale swarms, paying little attention to the behavior of small swarms. Furthermore, most of these works focus on single-file service models. Recently, there have been few papers that analyze multi-channel P2P streaming systems [26,27]. Hoverer, these studies are not directly applicable to stored media streaming where users viewing videos are unsynchronized.

The idea of using caching to support large user base and large catalogs is not new. In fact, is has been suggested that proxy caching [11] and prefix-caching [23] can significantly improve the performance of traditional client-server VoD systems. The key difference between these proposals and our system is that we consider a distribution swarm with out-of-order delivery of video pieces, while the previous proposals use unicast streaming for video delivery and hence rely on in-order delivery of video pieces.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a provider-managed P2P VoD framework for efficient delivery of niche content. By reserving small portions of peers' resources and using novel, weighed caching techniques, providers can support much larger catalogs for VoD. We demonstrated through analytical analysis, simulations and experiments on planetlab that our system design can provide short startup delays and high streaming quality for niche content.

As part of future work, we intend to systematically explore how to determine and adapt the barrier between NCC and PCC to optimize system-wide performance. We also intend to address video popularity dynamics. As a video becomes more popular, the swarm becomes more reliable, the video more prevalent in the PCC, and hence presumably it can be thinned from the NCC. In contrast, as the video becomes less popular, and clients begin to flash the video from their PCC, the NCC capacity may need to be increased to compensate for the video deletions. We intend to explore on-line algorithms that adapt the cache so that the number of copies of a video across both NCC and PCC never falls below a desired level. Finally, we intend to explore how to support video seeking operations of users, such as fast-forwarding. Our initial approach is to use anchor-points [16]. When a user attempts to go to a particular position in the video, if the piece for that position is missing then the closest anchor point is used instead. To improve the startup delay after such a jump, we intent to apply the biased distribution in between these points.

APPENDIX

A. CONTINUITY ANALYSIS

Simplification of continuity expression in Eq. (3): Using algebraic manipulation, we can rewrite the expression for $p(i,k)$ given by Eq. (1) and (2) as

$$p(i,k) = 1 - (1 - p(i,\tilde{k}-1))(1 - F(i)S(i,\tilde{k}))$$

where $\tilde{k} \triangleq \max(k, i+d)$. From the above equation we can see that $p(i,k)$ has linear recurrence relation on k. Applying recursive substitutions on $p(i,k)$, we get

$$p(i,k) = 1 - \prod_{j=0}^{\tilde{k}} (1 - F(i)S(i,\tilde{k}-j)) \tag{19}$$

The simplified representation follows immediately from Eq. (19) since $p_i = p(i,i+d)$. □

Derivation of piece continuity $p_s(i)$ in Eq. (4): Let $c_i(k)$ be the probability that playback is continuous ($k=0$) or held at piece i for k rounds ($k>0$), and $p_s(i)$ be the probability that piece i is available when playback advances to position i. Then, we can express $p_s(i)$ as

$$p_s(i) = \sum_{k=0}^{2m-1} c_{s-1}(k) p_{s-1}(i,k)$$

$$p_s(i,k) = 1 - (1 - p_s(i)) \prod_{j=0}^{k} (1 - F(i)S_s(i,j))$$

$$S_s(i,k) = \prod_{j=s}^{i-1} 1 - \frac{F(j)}{p_a N}(1 - p_s(j,k-1))$$

where $p_s(i,k)$ is the probability that a peer has piece i when the time spent on playback of s is $k+1$ rounds, $S_s(i,k)$ is the probability that piece i is the most needed piece available for download, and $F(i)$ is the probability that at least one peer with piece i, as defined in Eq. (2).

The probability piece i is available when playback is at position 0, $p_0(i)$, is the probability to acquire piece i within d startup rounds. Based on Eq. (3), we have that $p_0(i) = 1 - \prod_{j=0}^{d-1}(1 - F(i)S(i,j))$. The expression for $p_s(i,k)$ and that for $S_s(i,k)$ can be derived in a similar way to that used to derive the playback continuity expression $p(i,k)$ in Eq. (1) and $S(i,k)$ in Eq.(2), respectively, given an additional assumption that playback is held at s while i is being fetched.

PROPOSITION A.1. *Given a uniform piece distribution and a startup delay of zero, the playback continuity function p_i is monotonic increasing*

$$p_i < p_{i+1} \tag{20}$$

PROOF. We prove the proposition by showing that:

$$p(i+1,k+1) \geq p(i,k) \qquad \forall i,k : k \leq i.$$

The claim above immediately yields the proposition since $p_i = p(i,i)$ for a zero startup delay. We prove the claim using induction on i. First, we consider the base case. From Eq. (3), we have

$$p(1,1) = 1 - (1 - F(1)S(1,0)(1 - F(1)) \geq S(1,0),$$

where the inequity holds since $F(i)$ is a probability (i.e., $F(i) \leq 1$). From Eq. 19, we also have

$$S(1,0) = 1 - \frac{F(1)}{p_a N}(1 - p(0,0)) \geq p(0,0),$$

where the inequity holds since $p(0,0)$ is a probability. Now, suppose the claim holds for i. From Eq. (3), we have that for $k \leq i$:

$$\begin{aligned} p(i+1,k+1) &= 1 - \prod_{j=0}^{k+1}(1 - F(i+1)S(i+1,j)) \\ &\geq 1 - \prod_{j=0}^{k}(1 - F(i+1)S(i+1,j+1)) \\ &= 1 - \prod_{j=0}^{k}(1 - F(i)S(i+1,j+1)). \end{aligned}$$

The second step holds since $F(i+1)$ and $p(i+1,0)$ are probabilities. The last step holds since $F(i+1) = F(i)$ when the replica distribution is unifrom. Based on the expression for $S(i,k)$ in Eq (2) and the induction hypothesis, it is easy to see that $S(i+1,k+1) \geq S(i,k)$, $\forall k : k \leq i$. Substituting the inequality for $S(i+1,k+1)$ in the expression above, we get

$$p(i+1,k+1) \geq 1 - \prod_{j=0}^{k}(1 - F(i)S(i,j)) = p(i,k).$$

□

Observe that the proof above is valid for a cache distribution function that is monotonic non-decreasing, i.e., $\forall i : r_i \leq r_{i+1}$. Hence, proposition A.2 holds for these cache distributions as well.

PROPOSITION A.2. *The playback continuity function p_i is bounded by*

$$(1 - F(i))^{i+d+1} \leq 1 - p_i \leq (1 - F(i)S(i))^{i+d+1}$$

where

$$S(i) = \frac{1}{i+d+1} \sum_{k=0}^{i+d} \prod_{j=\max(0,k-d)}^{i-1} 1 - \frac{F(j)}{p_a N}.$$

PROOF. Since $p(i,k)$ is a probability (in particular $0 \leq p(i,k) \leq 1$), we can bound $s(i,k)$ in Eq. (2) by

$$\prod_{j=\max(0,k-d)}^{i-1} 1 - \frac{F(j)}{p_a N} \leq s(i,k) \leq 1.$$

Substituting the upper bound on $s(i,k)$ into (3), we get the upper bound on p_i. Substituting the lower bound into (3) and then applying holder's inequality, we get the lower bound on p_i. □

B. MODELING DOWNLOAD TIME

We build upon the models in [13, 21] to develop a model for the download time in a VoD system using a BitTorrent-like protocol. Similarly to [13], our model captures the the impact of the service capacity of dedicated seeders on the performance, i.e., the impact of the service capacity provided by the peer's NCC on the download time. We extend the single-file models in [13, 21] to account for a system serving multiple videos with different popularity. The set of videos being served is denoted by $\{1, \cdots, |V|\}$. For video i, we use the notations in Table 1, as well as the following ones

- $x_i(t)$ number of peers who are downlaoding video i at time t. $\overline{x_i}$ is the equilibrium value of $x_i(t)$.
- $y_i(t)$ The number of peers who are seeding video i after downloading it. $\overline{y_i}$ is the equilibrium value of $y_i(t)$.
- z_i The dedicated seed bandwidth assigned to video i.
- γ The rate at which peers abort seeding the recently downloaded video.
- θ The rate at which clients abort the download.
- η_i The effectiveness of swarming

In the fluid model, the evolution of the number of peers with video i is given by:

$$\frac{dx_i(t)}{dt} = \lambda_i - \theta x(t) - \frac{\min\{cx_i(t), \mu x_i(t)\eta_i + \mu y_i(t) + z_i\}}{l_i} \tag{21}$$

$$\frac{dy_i(t)}{dt} = \frac{\min\{cx_i(t), \mu x_i(t)\eta_i + \mu y_i(t) + z_i\}}{l_i} - \gamma y_i(t). \tag{22}$$

The number of peers and seeds in steady state can be computed by letting $\frac{dx_i(t)}{dt} = 0$, $\frac{dy_i(t)}{dt} = 0$. When the downloading bandwidth is the constraint, i.e., $c\overline{x_i} \leq \mu\overline{x_i}\eta_i + \mu\overline{y_i} + z_i$, we have

$$\overline{x} = \frac{\lambda}{\theta + c} \tag{23}$$

When the uploading bandwidth is the constraint, i.e., $c\overline{x_i} > \mu\overline{x_i}\eta_i + \mu\overline{y_i} + z_i$, we have

$$\overline{x} = \frac{\lambda}{\nu_i\left(1 + \frac{\theta}{\nu_i}\right)} - \frac{z_i}{\eta_i\mu\left(1 + \frac{\theta}{\nu_i}\right)} \tag{24}$$

where $\frac{1}{\nu_i} = \frac{1}{\eta_i}\left(\frac{l_i}{\mu} - \frac{1}{\gamma}\right)$

To calculate the average download time of video i, we can use Little's law, as follows

$$\frac{\lambda_i - \theta\overline{x}}{\lambda}\overline{x} = (\lambda_i - \theta\overline{x})T_i \tag{25}$$

Using eqs. (23) and (24) we can express the average download time of video i as

$$T_i = \max\left\{\frac{l_i}{\theta + c}, \frac{1}{\nu_i\left(1 + \frac{\theta}{\nu_i}\right)}\left(1 - \frac{z_i\nu_i}{\lambda\eta_i\mu}\right)\right\} \tag{26}$$

Assuming that peers do not seed the recently downloaded video ($\frac{1}{\gamma} \to 0$) and do not abort the download ($\theta \to 0$), we can simplify the average download time to

$$T_i = \max\left\{\frac{l_i}{c}, \frac{l_i}{\eta_i}\left(\frac{1}{\mu} - \frac{z_i}{\lambda_i l_i \mu}\right)\right\} \tag{27}$$

.

C. REFERENCES

[1] Nada home page: http://www.nanodatacenters.eu/.

[2] Tivo home page: http://www.tivo.com/.

[3] M. S. Allen, B. Y. Zhao, and R.Wolski. Deploying video-on-demand services on cable networks. In *ICDCS*, Toronto, Canada, June 2007.

[4] C. Anderson. *The Long Tail: Why the Future of Business is Sellecing Less of More*. Hyperion, July 2006.

[5] S. Annapureddy and S. Guha. Exploring VoD in P2P swarming systems. In *IEEE INFOCOM*, Anchorage, Alaska , USA, May 2007.

[6] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and improving a bittorrent networks performance mechanisms. In *IEEE INFOCOM*, Barcelona, Spain, April 2006.

[7] Y. Boufkhad, F. Mathieu, F. de Montgolfier, D. D. Perino, and L. Viennot. Achievable catalog size in peer-to-peer video-on-demand systems. In *IPTPS*, Tampa Bay, FL, USA, February 2008.

[8] C. Bram. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer Systems*, May 2003.

[9] N. Carlsson and D. L. Eager. Peer-assisted on-demand streaming of stored media using bittorrent-like protocols. In *IFIP/TC6 Networking*, pages 570–581, Atlanta, GA, USA, May 2007.

[10] M. Cha, H. Kwak, P. Rodriguez, Y. Y. Ahn, and S. Moon. I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system. In *ACM IMC*, New York, NY, USA, October 2007.

[11] Y. Chae, K. Guo, M. M. Buddhikot, S. Suri, and E. W. Zegura. Silo, rainbow, and caching token: schemes for scalable, fault tolerant stream caching. *IEEE JSAC*, 20(7):1328– 1344, 2002.

[12] Y. F. Chen, Y. Huang, R. Jana, H. Jiang, M. Rabinovich, J. Rahe, B. Wei, and Z. Xiao. Towards capacity and profit optimization of video-on-demand services in a peer-assisted iptv platform. *Multimedia Systems*, 15(1):19–32, 2009.

[13] S. Das, S. Tewari, and L. Kleinrock. The case for servers in a peer-to-peer world. In *IEEE ICC*, Washington, DC, USA, June 2006.

[14] G. Dn and N. Carlsson. Dynamic swarm management for improved bittorrent performance. In *IPTPS*, Boston, MA, USA, April 2009.

[15] C. Huang, J. Li, and K. W. Ross. Can internet video-on-demand be profitable? In *ACM SIGCOMM*, Kyoto, Japan, August 2007.

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

[17] K. W. Hwang, V. Misra, and D. Rubenstein. Stored media streaming in bittorrent-like p2p networks. Technical Report cucs-024-08, Columbia University, New York, NY, April 2008.

[18] V. Janardhan and H. Schulzrinne. Peer assisted vod for set-top box based ip network. In *P2P-TV*, New York, NY, USA, 2007.

[19] J. G. Luo, Q. Zhang, Y. Tang, and S. Q. Yang. A trace-driven approach to evaluate the scalability of p2p-based video-on-demand service. *IEEE TPDS*, 20(1):59–70, 2009.

[20] N. Parvez, C. Williamson, A. Mahanti, and N. Carlsson. Analysis of bittorrent-like protocols for on-demand stored media streaming. In *ACM SIGMETRICS*, volume 36, pages 301–312, Annapolis, MD, USA, 2008.

[21] D. Qiu and R. Srikant. Modeling and performance analysis of bittorrent-like peer-to-peer networks. In *ACM SIGCOMM*, Portland, OR, USA, August 2004.

[22] P. Rodriguez, S. Annapureddy, and C. Gkantsidis. Providing video-on-demand using peer-to-peer networks. In *IPTV Workshop, WWW*, May 2006.

[23] S. Sen, J. Rexford, and D. Towsley. Proxy prex caching for multimedia streams. In *IEEE INFOCOM*, New York, NY, USA, March 1999.

[24] K. Suh, C. Diot, J. Kurose, L. Massoulie, C. Neumann, D. F. Towsley, and M. Varvello. Push-to-peer video-on-demand system: Design and evaluation. *IEEE JSAC*, 25(9):1706–1716, 2007.

[25] A. Vlavianos, M. Iliofotou, and M. Faloutsos. Bitos: Enhancing bittorrent for supporting streaming applications. In *IEEE INFOCOM*, Barcelona, Spain, April 2006.

[26] C. Wu, B. Li, and S. Zhao. Multi-channel live p2p streaming: Refocusing on servers. In *IEEE INFOCOM*, Phoenix, AZ, USA, Feb 2008.

[27] D. Wu, Y. Liu, and K. Ross. Queuing network models for multi-channel p2p live streaming systems. In *IEEE INFOCOM*, Rio De Zaneiro, Brazil, April 2009.

[28] X. Zhang, J. Liu, B. Li, and T. P. Yum. Coolstreaming/donet: A data-driven overlay network for peer-to-peer live media streaming. In *IEEE INFOCOM*, Miami, FL, USA, April 2005.

[29] Y. Zhou, D. M. Chiu, and J. C. S. Lui. A simple model for analyzing p2p streaming protocols. In *IEEE ICNP*, Beijing, China, October 2007.