This paper was presented as part of the main technical program at IEEE INFOCOM 2011

Exploring the Optimal Replication Strategy in P2P-VoD Systems: Characterization and Evaluation

Weijie Wu John C.S. Lui
Computer Science & Engineering Department
The Chinese University of Hong Kong
Email: {wjwu,cslui}@cse.cuhk.edu.hk

Abstract— **Content providers of P2P-Video-on-Demand (P2P-VoD) services aim to provide a high quality, scalable service to users, and at the same time, operate the system with a manageable operating cost. Given the volume-based charging model by ISPs, it is to the best interest of the P2P-VoD content providers to reduce peers' access to the content server so as to reduce the operating cost. In this paper, we address an important open problem: what is the "*optimal replication ratio*" in a P2P-VoD system such that peers will receive service from each other and at the same time, reduce the traffic to the content server. We address two fundamental problems: (1) what is the optimal replication ratio of a movie given its popularity, and (2) how to achieve the optimal ratios in a distributed and dynamic fashion. We formally show how movie popularities can impact server's workload, and formulate the video replication as an optimization problem. We show that the conventional wisdom of using the proportional replication strategy is *non-optimal*, and expand the design space to both *passive replacement policy* and *active push policy* to achieve the optimal replication ratios. We consider practical implementation issues, evaluate the performance of P2P-VoD systems and show that our algorithms can greatly reduce server's workload and improve streaming quality.**

I. Introduction

Adapting the peer-to-peer (P2P) technology into video-on-demand (VoD) service has received a lot of attention in the past few years. There are a number of successful P2P-VoD systems like PPLive, PPStream and UUSee. The key advantage of using this technology is that the system can utilize peers' resources to satisfy other peers' viewing requirement, and thereby achieve a more scalable system as compared to the traditional client-server architecture. The main difference between P2P-VoD systems and P2P live streaming systems is that peers in P2P-VoD service need to contribute a much larger local cache space[1] as well as their upload bandwidth. For example, in PPLive system [5], each peer needs to dedicate 1 GB of local storage for a scalable P2P-VoD service. This local storage is for caching some previously watched movies so that in case a new peer arrives, this peer does not need to download from the VoD server but rather, obtains the data directly from other peers. This unique feature not only frees up the server to serve other peers, but more importantly, reduces the upload bandwidth traffic of P2P-VoD content providers. Since content providers usually pay ISPs based on the *volume-based charging* scheme, it is to the best interest of a P2P-VoD content provider to reduce the server's upload traffic, and a proper caching policy can achieve such goal. Given the distributed cache spaces of all peers, we aim to address how to utilize them efficiently so as to reduce the operating cost of the P2P-VoD content provider. Specifically, given the caching and upload capacities, what is the "*optimal caching policy*" so as to minimize the traffic to the P2P-VoD server?

Unlike the P2P file sharing applications, to support peers in a P2P-VoD service, one has to guarantee (a) a low start-up latency in accessing the movies; (b) supporting the required playback-rate for all peers which want to watch the movie. Therefore, a good replication policy should duplicate enough replicas so as to support the required workload. To illustrate, consider a P2P-VoD system with 100 cache units (i.e., each cache unit can store one movie). There are two movies M_1, M_2 with the same viewing popularity, i.e., each with 50 viewers. Assume a replication policy keeps 99 replicas for M_1 but only one replica for M_2. Then, requests for M_2 will fail to find enough peers' support and have to go to the server, yet there are excessive replicas of M_1 and some peers which cache M_1 will not be using their upload bandwidth efficiently.

Characterizing the optimal caching policy is challenging, especially when peers have different upload capacities and movies have different viewing popularities. These factors make designing the optimal replication policy an open problem so far. In this paper, we address the following problems:

- Given movies' viewing popularities, what is the *optimal* ratio of total storage space that should be dedicated to each movie such that when a new peer arrives, it is most likely to be served by other peers in the system?
- Given the system dynamics (e.g., peers' churn), how to design distributed and adaptive algorithms to achieve the optimal replication ratios?

Previous work [5] and conventional wisdom suggested to use the *proportional replication strategy*, i.e., replicating movies proportionally to their popularities. However, authors in [13] reported a contradicting result that the proportional replication may have poor performance, especially for unpopular movies. In this work, we give the formal justifications on the above contradiction and provide insights to characterize the optimal replication ratios. These insights provide us with the principles in designing practical and effective replication algorithms. The contributions of our paper are:

[1] In this paper, local cache space refers to physical memory or storage space that is dedicated to cache any multimedia data.

978-1-4244-9921-2/11/$26.00 ©2011 IEEE

- We show that proportional replication is not the optimal strategy. Instead, one should replicate movies proportionally to the *deficit bandwidth* (we will formally define this concept in Sec. II-B). The optimal replication strategy should be more greedy for unpopular movies.
- We expand the design space by considering both *passive* and *active* replication. We propose a passive replacement algorithm to decide which movie to purge when a local storage is full; we also propose an active replication algorithm to aggressively push data to peers so as to achieve the desired replication ratios. We evaluate our proposed algorithms and show significant improvement of video quality and large reduction on server's workload.

This is the outline of our paper. In Section II we present a mathematical model for P2P-VoD system and give an optimization framework for the replication problem. In Section III, we discuss the optimal replication ratios in software based systems. In Section IV, we present our passive replacement and active push algorithm to achieve the optimal replication ratios. In Section V, we consider several important practical issues in implementation and present the performance results of our algorithms. Section VI states the related work and Section VII concludes.

II. Mathematical Model

Let $\mathcal{M} = \{M_1, \ldots, M_K\}$ be the set of movies which are stored in $\mathcal{S}$, the logical server of the P2P-VoD system[2]. Peers which want to watch movie M_k can access the data from (1) $\mathcal{S}$, or (2) any other peer which is watching M_k, or (3) peers which have watched and cached M_k.

A P2P-VoD content provider aims to minimize, or at least to reduce, the upload bandwidth traffic of $\mathcal{S}$ so as to reduce its operating cost. The upload consumption of $\mathcal{S}$ is affected by (1) the peer scheduling policies, and (2) movie replication strategies. For the ease of presentation, we decompose these two design problems and use a realistic and practical peer scheduling strategy: a peer first requests data from other peers, and only when other peers cannot supply sufficient bandwidth, this peer requests data from $\mathcal{S}$. In here, we focus on the movie replication strategy and show how it can significantly impact the upload bandwidth consumption of $\mathcal{S}$.

In what follows, we first state the general P2P-VoD setting, then we give a general model to show how replication strategy can affect the upload bandwidth consumption of $\mathcal{S}$.

A. Peer Type and Peer Scheduling Policy

In general, there are two types of peers:

- *Active peers*: an active peer is one which is currently watching a movie and at the same time, provides upload service to other peers. The data that this peer uploads may come from the movie it is currently watching, or the data it has stored in its local cache.
- *Inactive peers*: an inactive peer is one which is currently *not* watching any movie, but it stays *online* in a P2P system, and this peer contributes its upload bandwidth to other peers. This type of peers can represent those online set-top boxes which have built-in P2P functionalities.

We differentiate these two types of peers because their distribution in a P2P-VoD system will have a major impact on replication strategies.

A peer scheduling strategy includes two design issues: (1) from a downloader's viewpoint, it needs to decide which peers to request the data from; (2) from an uploader's viewpoint, it needs to decide which peers to upload the data to when it receives multiple requests.

Let us first describe how an active peer downloads a movie. Suppose a peer wants to watch movie M_k. In general, its request can be supported by three different sources: (a) the server $\mathcal{S}$; (b) active peers which are currently watching the same movie M_k, and we denote these as the *concurrent peers*; (c) peers which are not watching movie M_k but have stored M_k at their local cache, and we denote these as the *replication peers*. Note that a replication peer may be an active peer watching another movie; or an inactive peer which has movie M_k in its cache. Lastly, in order to have a short start-up latency and good streaming quality, each peer needs to download the movie at its playback rate r.

We consider the following practical peer scheduling policy: a peer first seeks help from concurrent peers; if the playback rate cannot be satisfied, the peer seeks help from replication peers; if the playback rate still cannot be satisfied, the server $\mathcal{S}$ will upload the data to this peer.

We would like to comment on the rationale of this scheduling policy. We let $\mathcal{S}$ be the last requesting target of a peer in order to reduce the upload consumption of $\mathcal{S}$ so as to reduce the operating cost of the P2P-VoD system. Note that, in real systems, a peer may cache only *part* but not the whole movie (details are discussed in Section V). Therefore, requesting from a replication peer may result in frequent rescheduling: a peer P_i that requests data from a replication peer P_j needs to look up other peers for support when P_i's viewing part is not cached in P_j; however, it is more likely that P_i can download from one particular concurrent peer P_k throughout the watching period. This is why we prefer concurrent peers over replication peers to support the viewing requirement.

One interesting question is how should a peer seek help from concurrent peers? Note that only those concurrent peers which arrived *earlier* than this peer can assist in data upload. We use the "*sequential-search-for-help*" principle: suppose there are n_k concurrent peers watching movie M_k, and these concurrent peers are denoted as P_1, P_2, ..., P_{n_k}, with P_1 having the earliest arrival time. In here, P_1 cannot get any download from other concurrent peers but instead, may ask replication peers or $\mathcal{S}$ for help, while the only concurrent peer that P_2 can seek service from is P_1. In general, peer P_i will first seek help from P_{i-1}; if the request cannot be satisfied, P_i will send request to peers P_{i-2}, P_{i-3}, ..., in a sequential manner, until it receives data to satisfy its playback rate r.

From an uploader's viewpoint, it may receive multiple download requests simultaneously. In our work, an active

[2]A logical server may consist of many physical servers which process any request from peers in a P2P-VoD system.

peer will serve its concurrent peers with a higher priority. Only when its upload bandwidth is not fully consumed by concurrent peers, the active peer will provide the remaining upload bandwidth to other peers. For an inactive peer, it will upload to those peers watching any movie that is stored in its local cache. The assumption is not only for mathematical tractability, but this simple scheduling rule can also maximize the number of peers to receive data at the playback rate r.

B. Model of Movie Popularity on Server's Workload

Let us illustrate how the movie popularity can affect the server's upload bandwidth. First, peer P_i's download rate can be supported partly by concurrent peers, partly by replication peers, and partly by the server $\mathcal{S}$. Let $\widetilde{d}_i^c$ and $\widetilde{d}_i^r$ be the total download rates supported by concurrent peers and replication peers, respectively. If $\widetilde{d}_i^c + \widetilde{d}_i^r$ is below the playback rate r, the server needs to fill up the gap $r - \widetilde{d}_i^c - \widetilde{d}_i^r$ so as to satisfy the QoS guarantee. For the ease of presentation, we denote $\widetilde{d}_i^s = r - \widetilde{d}_i^c - \widetilde{d}_i^r$ $(\forall i \in \mathcal{N}_\mathcal{A})$ as the filling upload rate from the server to P_i, where $\mathcal{N}_\mathcal{A}$ is the set of all active peers.

Denote $\mathcal{N}_k$ as the set of active peers which are currently watching movie M_k. Obviously, we have $\mathcal{N}_k \subseteq \mathcal{N}_\mathcal{A}$. The expected upload bandwidth consumption of the server $\mathcal{S}$ is denoted as U, which can be expressed as:

$$U = E\left(\sum_{i\in\mathcal{N}_A} \widetilde{d}_i^s\right) = E\left(\sum_{k\in\mathcal{M}}\sum_{i\in\mathcal{N}_k} \widetilde{d}_i^s\right). \tag{1}$$

Based on the peer scheduling policy above, we have:

Lemma *1:* In a large scale P2P-VoD system, the expected upload consumption at the server is approximately

$$U \approx \sum_{k\in\mathcal{M}} \left[E\left(\sum_{i\in\mathcal{N}_k}(r - \widetilde{d}_i^c)\right) - \mathcal{R}_k\right]^+, \tag{2}$$

where $\mathcal{R}_k$ is the maximal upload bandwidth from all replication peers that can contribute to M_k, or $\sum_{i\in\mathcal{N}_k} d_i^r \le \mathcal{R}_k$; and $(x)^+ = \max(0, x)$.

Due to the page limit, we omit the proofs of lemmas and propositions in this paper. Interested readers may refer to our technical report [16] for details.

Let us define the following important notation.

Definition *1:* For movie M_k, we define *deficit bandwidth* as

$$\widetilde{\mathcal{D}}_k(n_k) = \sum_{i\in\mathcal{N}_k}(r - \widetilde{d}_i^c) = \sum_{i\in\mathcal{N}_k}(\widetilde{d}_i^r + \widetilde{d}_i^s), \tag{3}$$

and let

$$\mathcal{D}_k(n_k) = E\left[\widetilde{\mathcal{D}}_k(n_k)\right] \tag{4}$$

be the *expected deficit bandwidth* of having n_k concurrent peers in $\mathcal{N}_k$, i.e., the bandwidth gap between the total required playback rates and the total download rates that can be supported by n_k concurrent peers. We can now rewrite Eq. (2) and directly obtain the following lemma:

Lemma *2:* Given the number of concurrent peers in each movie, the expected upload consumption at the server is

$$U = \sum_{k\in\mathcal{M}} (\mathcal{D}_k(n_k) - \mathcal{R}_k)^+. \tag{5}$$

Our objective is to find a replication strategy, which determines $\mathcal{R}_k$, such that U, the average upload bandwidth consumption of the server $\mathcal{S}$, can be minimized. Thus, we want to explore how the popularity of a movie may affect $\mathcal{D}_k(n_k)$, which in turn will affect the choice of replication. In what follows, we state an important lemma:

Lemma *3:* Deficit bandwidth can be iteratively calculated by

$$\widetilde{\mathcal{D}}_k(n_k) = n_k r - \sum_{i\in\mathcal{N}_k} \widetilde{d}_i^c, \tag{6}$$

$$\widetilde{d}_i^c = \begin{cases} 0, & i = 1; \\ \min\left(\sum_{j=1}^{i-1}(\widetilde{u}_j - \widetilde{d}_j^c), r\right), & i \ge 2, \end{cases} \tag{7}$$

where $\widetilde{u}_j$ denotes the maximal upload rate of P_j.

Let us use an example to illustrate the above framework in calculating d_i^c and $\widetilde{\mathcal{D}}_k(n_k)$. Assume six peers are watching movie M_k. These peers are $P_1,\ldots,P_6$, with P_1 arriving the earliest. We fix the movie playback rate at $r = 500$ kbps. If we know their upload capacities $\widetilde{u}_i$ as illustrated in Tab. I, we can calculate $\widetilde{d}_1^c, \ldots, \widetilde{d}_6^c$ respectively, and by summing up all $(r - \widetilde{d}_i^c)$ we can get the deficit bandwidth, or $\widetilde{\mathcal{D}}_k(6) = 1000$ kbps. This $\widetilde{\mathcal{D}}_k(6)$ represents the bandwidth required to support all these six peers.

	P_1	P_2	P_3	P_4	P_5	P_6
u_i (kbps)	500	800	200	800	300	1000
d_i^c (kbps)	0	500	600	400	600	500
$\widetilde{\mathcal{D}}_k(6)$ (kbps)	1000					

TABLE I: Example for calculating $\widetilde{d}_i^k$ and $\widetilde{\mathcal{D}}_k(n_k)$

In summary, given each peer's upload capacity (or $\widetilde{u}_i$), one can determine $\widetilde{d}_i^c$ and $\widetilde{\mathcal{D}}_k(n_k)$. Furthermore, given the upload capacity distribution of $\mathcal{N}_k$, we may determine $\mathcal{D}_k(n_k)$ by taking the expectation. Based on this framework, we can show how movie's popularity may impact the upload bandwidth consumption of the server $\mathcal{S}$. In Section III, we will show some interesting implications.

C. Model of Replication on Server's Workload

Let us show how replication can help to reduce the server's workload. Intuitively, without using replication (or when $\widetilde{d}_i^r = 0$), $\mathcal{D}_k(n_k)$ will be the upload bandwidth consumption of server $\mathcal{S}$ for movie M_k. Replication helps to reduce server's upload bandwidth consumption since peers could use replicas to partially support the deficit bandwidth $\mathcal{D}_k(n_k)$. For the ease of presentation, we have the following assumptions:

- The system contains K movies and N peers, of which αN are active peers. An active peer watches movie $M_k \in \{M_1, ..., M_K\}$ with probability ρ_k, with $\sum_{k=1}^K \rho_k = 1$.
- Each peer, either active or inactive, caches one movie that it has watched before. An active peer can also upload the movie it is currently watching to other peers[3].

[3]Note that the typical cache size of a peer is 1 GB (e.g., in PPLive), and typical movie size is about 500 MB. Therefore, we assume that besides the currently watching movie, an active peer can cache one extra movie so as to cope with typical settings of real systems.

- Peers' scheduling strategy follows the assumptions in Section II-A.

To model the distribution of number of active peers in each movie, one can use a closed queueing network to represent the P2P-VoD system. Let random variable $\widetilde{n}_k$ denote the number of peers watching movie M_k. Similar to the work in [14], we can express the probability for movie M_k having n_k viewers, for $k = 1, 2, \ldots, K$, as:

$$P(\widetilde{n}_1 = n_1, \ldots, \widetilde{n}_K = n_K) = (\alpha N)! \frac{\rho_1^{n_1}}{n_1!} \ldots \frac{\rho_K^{n_K}}{n_K!}, \quad (8)$$

with $\sum_{k=1}^{K} n_k = \alpha N$. Here, ρ_i implies the relative popularity of movie M_i. Based on this queueing model, we have the following proposition:

Proposition *1:* The average upload consumption of $\mathcal{S}$ is

$$\overline{U}(\mathbf{q}) = \sum_{\sum n_k = \alpha N} \left\{ (\alpha N)! \frac{\rho_1^{n_1}}{n_1!} \ldots \frac{\rho_K^{n_K}}{n_K!} \times \sum_{k=1}^{K} \left[\mathcal{D}_k(n_k) - q_k \sum_{l=1}^{K} (C_l^u(n_l) + \mathcal{D}_l(n_l) - n_l r) - q_k C_{\mathcal{I}}^u((1-\alpha)N) \right]^+ \right\}, \quad (9)$$

where q_k is the fraction of cache space dedicated to M_k, $C_k^u(n_k) = E\left(\sum_{i \in \mathcal{N}_k} \widetilde{u}_i\right)$, $C_{\mathcal{I}}^u(n_I) = E\left(\sum_{i \in \mathcal{N}_{\mathcal{I}}} \widetilde{u}_i\right)$, and $\mathcal{N}_{\mathcal{I}}$ is the set of inactive peers.

Now the model of the *optimal replication strategy* can be formulated as follows. We seek to find a set of replication ratios $\mathbf{q}$ which satisfies:

$$\begin{aligned} \min_{\mathbf{q}} \quad & \overline{U}(\mathbf{q}) \\ \text{subject to} \quad & \mathbf{q}^T \cdot \mathbf{1} = 1. \end{aligned} \quad (10)$$

In other words, we want to determine the number of replicas for each movie $M_k \in \mathcal{M}$, so that the average upload bandwidth consumption of $\mathcal{S}$ can be minimized.

In general it is difficult to get a close form solution for this optimization problem. However, as we will show in the following section, when we consider some common deployment cases, one can get some interesting implications and insights in designing replication strategies.

III. Optimal Replication under Software Deployment

There are two typical deployment scenarios for P2P-VoD services: software and set-top box deployment. In software based systems like PPLive, most users contribute their upload bandwidth to others peers only when they are watching a movie; while in set-box deployment, those set-top boxes are "*online*" even when users are not watching any movie.

We consider both deployment cases. However, due to page limit, in this section we focus on software deployment; interested readers may refer to our technical report [16] for analysis and design principles in set-top box deployment.

We define two operating modes based on the average upload capacity $\overline{u}$ and the playback rate r: we call the system operates at *deficit mode* when $\overline{u} < r$, or *surplus mode* when $\overline{u} > r$. In addition, we further divide surplus mode into two modes: (a) *slightly-surplus mode*, i.e., $\overline{u} \gtrsim r$, and (b) *highly-surplus mode*, i.e., $\overline{u} >> r$.

We like to note that the deficit mode is *not* a rational operating point for a software P2P-VoD system. If the system operates in deficit mode, the server's total upload bandwidth consumption is at least $N(r - \overline{u})$, so that the system cannot be scalable with huge number of viewers. The only way to reduce the workload in $\mathcal{S}$ is to lower the playback rate r so that the system can be in the surplus mode. On the other hand, if the system operates in the highly-surplus mode, concurrent peers can support each other with ease, so that the abundance of peers' upload bandwidth makes replication unnecessary. This abundance in resource will motivate the P2P-VoD system designer to increase the playback rate r so as to improve video quality,which eventually brings the system to the slightly-surplus mode. To summarize, the only *interesting case* for the software-based P2P-VoD deployment is when the system is operating at the slightly-surplus mode. As we will show, designing an optimal replication strategy is a non-trivial task.

Under the software deployment scenario, by setting $\alpha = 1$ in Eq. (9), we have the following proposition:

Proposition *2:* In software deployment, the average upload bandwidth consumption at the server $\mathcal{S}$ is:

$$\overline{U}(\mathbf{q}) = \sum_{\sum n_k = N} \left\{ N! \frac{\rho_1^{n_1}}{n_1!} \ldots \frac{\rho_K^{n_K}}{n_K!} \cdot \sum_{k=1}^{K} \left[\mathcal{D}_k(n_k) - q_k \sum_{l=1}^{K} (C_l^u(n_l) + \mathcal{D}_l(n_l) - n_l r) \right]^+ \right\}. \quad (11)$$

To minimize $\overline{U}$, the decisions on q_k are highly dependent on the shape and characteristics of $\mathcal{D}_k(n_k)$. We consider some typical distributions to represent peers' upload capacity so as to gain a deeper understanding. In addition, we also use numerical and fitting methods to show the characteristics of $\mathcal{D}_k(n_k)$. In here, we present one typical example.

Example 1 (Exploring the characteristics of $\mathcal{D}_k(n_k)$): Assume n_k peers are watching movie M_k, and the movie playback rate is $r = 500$ kbps. According to [4], we set a realistic upload capacity distribution as illustrated in Tab. II. We plot the deficit workload in Fig. 1 when varying $n_k \in [1, 100]$. It shows that $\mathcal{D}_k(n_k)$ is *concave* in n_k and can be fitted into the form $\mathcal{D}_k(n_k) = \mathcal{D}_0 - Ae^{-\lambda(n_k - n_0)}$.

Due to page limit, we omit other cases which we have studied, but we state the following observation:

bandwidth (kbps)	768	384	256	128
share	50%	30%	5%	15%

TABLE II: Peers' upload capacity distribution

Observation 1: *$\mathcal{D}_k(n_k)$ is a concave function on movie popularity n_k, and it converges to a fixed value when n_k approaches infinity.*

Physical meaning: The concavity and convergence features imply that for unpopular movies, they have a much higher marginal increase on the deficit bandwidth $\mathcal{D}_k(n_k)$ while for popular movies, the marginal increase on $\mathcal{D}_k(n_k)$ is minimal.

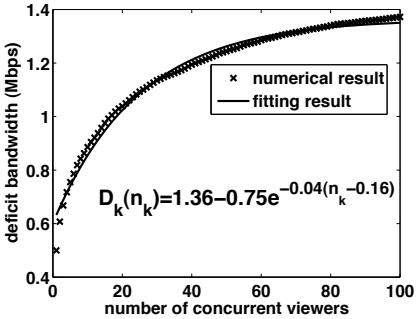


Fig. 1: Deficit bandwidth $\mathcal{D}_k$ vs. n_k

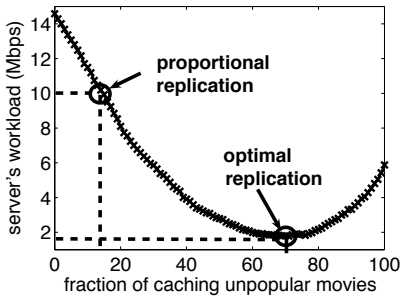


Fig. 2: Replication impact on server's workload in a software-deployed system, where $r = 500$ kbps.

An intuitive explanation of this feature is that peers within a larger set can cooperate more effectively and thus achieve higher utilization of peers' upload bandwidth. Using law of large number on Eq. (7), for large enough i, we have $\sum_{j=1}^{i-1} \widetilde{u}_j \approx (i-1)\overline{u}$ with *high* probability. Furthermore, if we assume $\overline{u} > r$ and $i > r/(\overline{u}-r)+1$, then we have

$$\widetilde{d}_i^c \simeq \min\left((i-1)\overline{u} - \sum_{j=1}^{i-1} \widetilde{d}_j^c, r\right) \geq \min\left((i-1)(\overline{u}-r), r\right) = r,$$

which means that any late arriving peer can receive sufficient download from concurrent peers and will not incur any bandwidth consumption on $\mathcal{S}$.

The above analysis shows the reason why deficit bandwidth $\mathcal{D}_k(n_k)$ is sub-linear. This also implies that the conventional wisdom of using the proportional replication strategy, or caching the more popular movies [5], [17], [18], is *sub-optimal*. Looking at Fig. 1, we see that a popular movie with 25 viewers needs nearly the same deficit bandwidth as a movie with 100 viewers, while an unpopular movie of five viewers needs more than half of the deficit bandwidth as compared with a popular movie with 100 viewers.

Now the impact of movie popularities on server's workload is clear: without replication, the server's workload for movie M_k is simply $\mathcal{D}_k(n_k)$. The goal of replication is to reduce the workload on $\mathcal{S}$. The non-linearity of $\mathcal{D}_k(n_k)$ implies one should be *greedy* in replicating unpopular movies. In the following, we use examples to justify our argument, and at the same time, determine the optimal replication ratios $\mathbf{q}$.

Example 2 (Sub-optimality of the proportional replication strategy): Assume that the system consists of $N = 600$ active peers and $K = 25$ movies. Movie M_1 - M_5 are popular movies while movie M_6 - M_{25} are unpopular. Peers choose each of the popular movies with a probability of $\rho_{pop} = 1/6$, or each of the unpopular movies with a probability of $\rho_{unpop} = 1/120$. The playback rate is set at $r = 500$ kbps, Peers' upload capacity follows Tab. II.

Denote q_{unpop} as the fraction of cache space (which can store 600 movies) that is dedicated to cache any unpopular movie (M_6 - M_{25}) while $1 - q_{unpop}$ is the fraction of cache space of caching popular movies (M_1 - M_5). We develop a simulator to explore the replication strategy. In particular, we vary the fraction of cache space that is used to store unpopular movies from 0% (i.e., all cache space is used to store popular movies) to 100% (i.e., all cache space is used to store unpopular movies), and evaluate the system performance. Fig. 2 illustrates the upload bandwidth consumption on the server when we vary q_{unpop}.

If the system only replicates popular movies, then all deficit bandwidth $\mathcal{D}_k$ of unpopular movies will contribute to the server's workload. This corresponds to the first point from the left (about 15 Mbps) in Fig. 2. Similarly, if all replications are for unpopular ones, the server's workload is around 6 Mbps and this corresponds to the last point at the right of the figure. Since $5/6$ fraction of the requests are for popular movies, using the *proportional replication strategy*, only $1/6$ or 17% of the cache space should be dedicated to store the unpopular movies. From the figure, we see that the workload on the server will be around 10 Mbps. However, the *optimal replication fraction* for unpopular movie should be around 70% where the server's upload bandwidth consumption is less than 2 Mbps. This simple example validates the following observation:

Observation 2: *In a software-based system, proportional replication is far from optimal and incurs a higher server's workload as compared with the optimal replication strategy.*

Example 3 (Exploring optimal replication ratio): We consider a P2P-VoD system with 700 peers and 35 movies. M_1 - M_5 are popular movies with $\rho_{pop} = 1/7$; M_6 - M_{15} are moderately popular movies with $\rho_{mid} = 1/70$; while M_{16} - M_{35} are unpopular movies with $\rho_{unpop} = 1/140$. We simulate this system and explore the server's workload at each possible integral fraction combination $\mathbf{q} \in \{(q_{pop}, q_{mid}, q_{unpop}) : q_{pop} + q_{mid} + q_{unpop} = 100\%\}$. Using exhaustive search, the popularity fractions and optimal replication fractions of different movies are shown in Tab. III. Obviously, the optimal replication strategy should be "aggressive" in caching unpopular movies.

	popular	moderate	unpopular
proportional fraction	71%	14%	14%
optimal replication fraction	18%	32%	50%
deficit bandwidth fraction ($\hat{q}_k$)	21%	30%	49%

TABLE III: Optimal replication ratios vs. popularity fractions

It is mathematically difficult to derive the optimal ratios of replication from the optimization problem, and one cannot afford to use exhaustive search for large scale P2P-VoD sys-

tems. However, we obtain an interesting finding: the optimal ratio for replication for movie M_k should be close to the fraction of deficit bandwidth $\mathcal{D}_k(n_k)$ for movie M_k among the total deficit bandwidth, i.e., let $\hat{q_k} = \frac{\mathcal{D}_k(n_k)}{\sum_{i=1}^{K} \mathcal{D}_i(n_i)}$, then we have $q_k^{opt} \simeq \hat{q_k}$. For instance, in the above example $\hat{q}_{pop} = 21\%, \hat{q}_{mid} = 30\%$ and $\hat{q}_{unpop} = 49\%$, which are near to the optimal ratios for replication. This example reveals the following observation:

Observation 3: *Proportional deficit bandwidth replication is a good estimate to the optimal replication ratios.*

Due to page limit, we omit examples using other typical settings, but they all strongly support our observations. We like to point out that there is an intuitive explanation to the above resource allocation strategy: without replication, $\mathcal{D}_k(n_k)$ is exactly the server's bandwidth consumption for movie M_k, thus, allocating remaining bandwidth in this ratio is the most efficient method to balance each movie's request load to the server $\mathcal{S}$. This idea leads to the replication algorithm which we will discuss in the next section.

IV. Algorithms to Control Replication Ratios

Let us present several algorithms to achieve the optimal replication ratio **q** in a P2P-VoD system. Note that a P2P-VoD is with high churn, therefore, the replication algorithms need to be robust and efficient in minimizing the server's workload.

In general, there are two ways to control or adjust the replication ratios:

- **Passive adjustment:** peers adjust the replication ratio in the P2P-VoD system using replacement algorithms, i.e., when a peer's local cache is full, it decides which movie(s) to purge so as to accommodate a new movie that can reduce the server's workload;
- **Active adjustment:** the server $\mathcal{S}$ and all peers actively push out (or upload) data for which the replicas are not at the desired replication ratios.

Passive Adjustment via Replacement Algorithm: When the cache space of a peer is full, a peer has to decide which movie(s) to purge from the cache space such that new movies could be cached. Since we want to explore the impact of movie popularities to the server's workload, we consider replication algorithms that replace the whole movie, instead of partial replacement. Nevertheless, generalization can be easily made.

Previous work [5] suggested a weight-based evaluation scheme, or replicating proportionally to a movie's popularity. Our previous examples already show the sub-optimality of such replication algorithm and one should replicate movie proportionally to its *deficit bandwidth*. This leads to the following replacement algorithm, which is illustrated in Algorithm 1.

The main idea behind this algorithm is to keep the number of replicas proportional to the deficit bandwidth. We define *satisfaction index* (SI) to express the extent that a movie has enough replicas. When $SI_k < 1$, it means that the number of replicas for movie M_k is below the desired replication ratio. On the other hand, if $SI_k > 1$, the number of replicas of M_k exceeds the desired ratio. Therefore, in each round, we purge the movie that has the highest SI from the local cache.

Algorithm 1 Replacement Algorithm

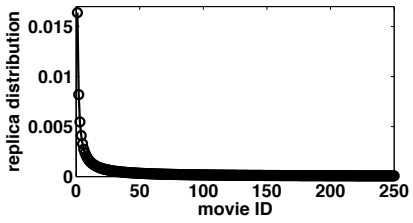
```
for any peer that starts to watch a new movie do
  if this movie is already stored in its local cache then
    do not replace any movie;
  else
    for each movie M_i in its local cache do
      calculate the expected number of caches for M_i:
          N_i^exp = [D_i(n_i)N] / sum_k D_k(n_k);
      calculate the current number of caches for M_i:
      N_i^cur;
      calculate the satisfaction index for M_i:
          SI_i = N_i^cur / N_i^exp;
    end for
    replace the movie with the highest satisfaction index.
  end if
end for
```

Algorithm 2 Push Algorithm

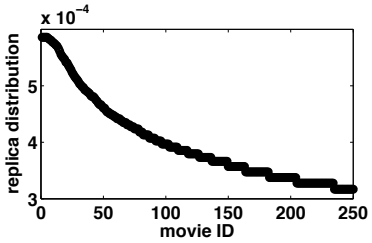
```
while system is in an idle time slot do
  find out a list of replication peer candidates which will
  most likely be inactive at the publishing instance;
  estimate the total volume available for push
  calculate the total volume desired for coming-to-hot
  movies:
          sum_k D_k(n_k) × (movie length)
  if total available volume is larger than desired then
    assign each movie M_i with the volume it desires;
  else
    for each movie M_i do
      assign volume
          D_i(n_i) / sum_k D_k(n_k) × (total available volume)
      to the list of replication peer candidates;
    end for
  end if
end while
```

We like to mention that although requiring global information, this algorithm is efficient in practice. Each peer reports to the tracker the current cached movies, and the tracker can calculate the satisfaction indices and broadcast to all peers. These can be done in a proper frequency (e.g., once in five minutes), and thus the overhead can be minimal. In Sec. V, we also evaluate the impact of delay in broadcasting SI.

***Active Adjustment via Push Strategy*:** Although our proposed replacement algorithm can reduce the server's workload, other factors still constrain the system performance. For instance, the server's workload may dramatically increase when a popular movie is newly released, since there will be a sudden surge of demand but only few peers in the system have cached this movie. By simply relying on the replacement algorithm, the system will take a long time to converge to the optimal replication ratios. To overcome this limitation, one can take a proactive approach to push out a movie if we know that it will be a popular object. This leads to the push algorithm as

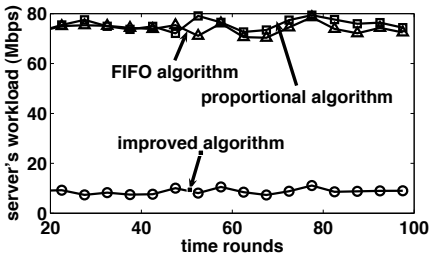


(a) proportional distribution

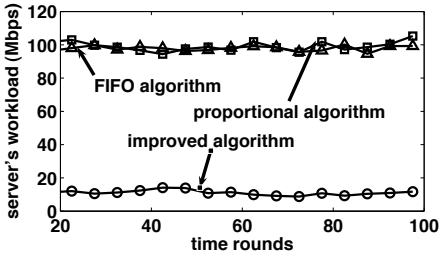


(b) optimal distribution

Fig. 3: Proportional vs. optimal replica distribution



(a) $\gamma = 1$



(b) $\gamma = 1.5$

Fig. 4: Comparison of replacement algorithms

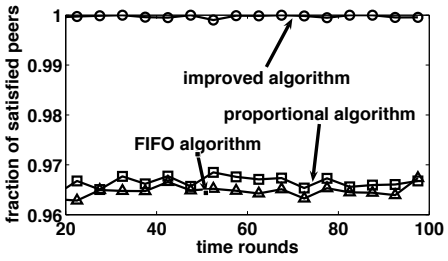


Fig. 5: Impact of replication algorithm on satisfied peers

depicted in Algorithm 2. In essence, the push algorithm utilizes the excessive bandwidth in idle time slots to push some data for future use so server's workload could be "*flattened*" over time. Note that there is a daily periodicity of user behavior [5], leading it possible to predict the peak hours and popularity dynamics (in particular, for TV series) so as to implement the algorithm in practice. We will illustrate the effect of the algorithm in the following section.

V. Performance Evaluation

A. Performance of Replacement Algorithm

We carry out extensive simulation to validate our models and evaluate the performance of our proposed replacement algorithm. For a software-based P2P-VoD system where $\alpha = 1$, we use the following as inputs to our simulator: (a) The system contains $N = 10,000$ active peers. Peer's upload capacity distribution follows Tab. II, with average of 531 kbps. (b) The system provides $K = 250$ kinds of movies. Each movie is with the playback rate of $r = 500$ kbps. (c) Movie popularity follows a Zipf distribution [9] with parameter γ. (d) At each round, an active peer requests a movie. Active peers switch movie channel every round.

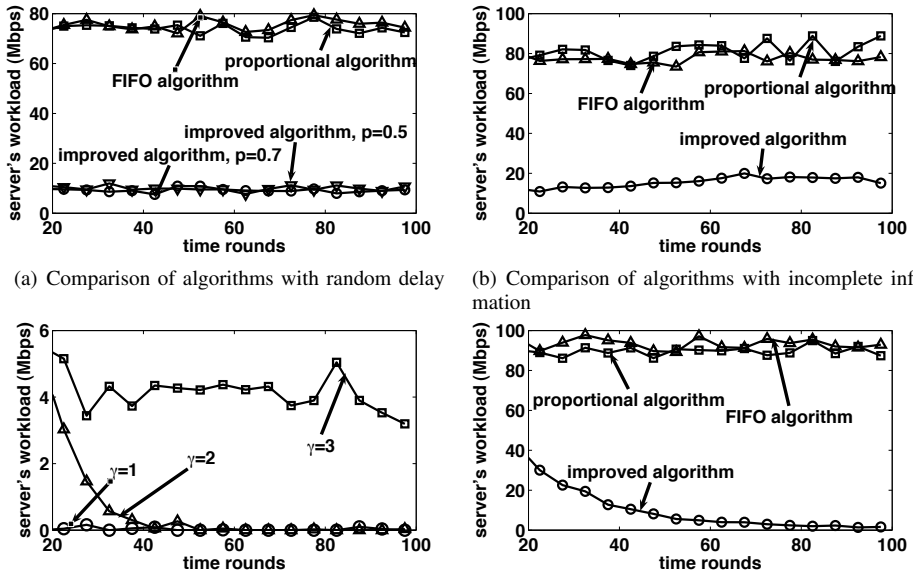
The movie playback rate and average upload capacity we set are based on realistic settings like PPLive. The tracker in the system records the current cache distribution in the system and that all peers can use this information to do replacement. We compare server's workload using the following replacement algorithms: (a) *Our proposed algorithm*, which keeps replicas proportional to deficit bandwidth; (b) *Proportional algorithm*, which keeps the replicas proportional to movie popularity; (c) *First-in-First-out (FIFO) algorithm*, which keeps the newly cached movie and purges the oldest one.

Fig. 3 illustrates the comparison of the proportional replica distribution vs. the optimal replica distribution (both normalized to 1). We set the Zipf distribution parameter $\gamma = 1$ and arrange the movies in a decreasing order of popularity. Our proposed algorithm keeps fewer popular movies than the proportional distribution and is more greedy in replicating unpopular movies. Thus, the optimal replica distribution is "*flatter*" than the proportional distribution.

Fig. 4 compares the server's workload using different replacement algorithms. We record the server's load after round 20 since we are interested in the steady state performance.

In Fig. 4(a) where $\gamma = 1$, the server's average workload is around 9 Mbps using our proposed replacement algorithm, but is 75 Mbps using the proportional algorithm or 74 Mbps using the FIFO algorithm. Hence, our proposed push algorithm reduces the server's workload by a *factor of eight*. When movies' popularities have high variability, the performance of our replacement algorithm shows even higher workload reduction. Fig. 4(b) illustrates this result with $\gamma = 1.5$, or higher variance in movies' popularity. In the steady state, the server's average workload is around 10, 103 and 101 Mbps using our proposed replacement algorithm, proportional algorithm and FIFO algorithm, respectively.

Another important performance measure is to evaluate the number of peers which can watch movie at the normal playback rate r (which we call *satisfied peers*) when given a finite server upload capacity. We set the simulation environment as the previous case and the server upload capacity is restricted at



(a) Comparison of algorithms with random delay

(b) Comparison of algorithms with incomplete information

(c) Improved algorithm with multiple cache spaces

(d) Comparison of algorithms with peer departure

Fig. 6: Practical issues

10 Mbps. Fig. 5 shows the comparison on fraction of satisfied peers using different replacement algorithms. From the figure, our proposed replacement algorithm can ensure the maximum ratio of peers that can watch the movie at the playback rate, achieving approximately 100% of satisfied peers, while the ratio is around 96% when using the proportional or FIFO algorithm. These results validate that our proposed replacement algorithm can reduce the server's workload and at the same time, improve streaming quality.

Practical issues: let us comment on some practical implementational issues and the adaptiveness of our algorithms.

First, we consider the possible delay when a peer receives information from the tracker. We assume that a peer has probability p to successfully update SI when doing replacement. If this is delayed, a peer will do replacement based on SI it received previously. In Fig. 6(a), we show that our algorithm is robust to this delay: when $p = 0.7$ or 0.5, the server's workload is much less than proportional or FIFO algorithms.

Secondly, we consider the case that some peers fail to report its cached movie to the tracker, and thus the replacement is based on incomplete information. Fig. 6(b) shows the comparison of server's workload with 10% failure to report. The server's workload is 16 Mbps when using our proposed algorithm, but will be 80 Mbps or 77 Mbps with proportional or FIFO algorithm. Compared with Fig. 4(a), our proposed algorithm is still robust with incomplete information.

Thirdly, when there are more cache space, the server's workload can be further reduced. In Fig. 6(c), we allow each peer to store up to three movies it has watched. The server's average workload is near zero when the variance of movie popularity is not large (e.g., $\gamma = 1$ or 2), and is only around 4 Mbps even if the movie popularities vary a lot (e.g., $\gamma = 3$).

Lastly, peers may not watch the entire movie but may leave after watching *part* of the movie. This behavior can lead it a more challenging task in supporting latter parts of movies. Theoretically one can use "chunk level" replication, but it comes with an overhead. In here, we do a minor modification on our proposed algorithm, i.e., divide a movie into several segments (we use three as default) and separately replicate these segments based on individual deficit workload. In Fig. 6(d), we assume that among all peers that watch the first part of any movie M_k, 80% of them also watch the second segment, and only 50% watch the third segment. It shows that our proposed algorithm is adaptive to this system, while the proportional or FIFO algorithm performs poorly.

B. Performance of Push Algorithm

In here, we show that the proposed push algorithm can significantly improve the system performance, particularly for a set-top box deployed system. The key idea is to push some data to some inactive peers in advance such that the server's workload could be flattened down when publishing a new popular movie. The simulation setting is similar to the previous one but the differences are: (a) There are 5,000 active peers and 5,000 inactive peers. (b) Playback rate of a movie is doubled to $r = 1.0$ Mbps. (c) Initially there are $K = 249$ kinds of movies $\{M_1,..,M_{249}\}$ with a Zipf popularity distribution ($\gamma = 1$). At the beginning of round t_p, a new popular movie M_{250} is published. The popularities of these 250 movies follow a new Zipf distribution and M_{250} is the most popular one.

Movie M_1 - M_{249} can be stored in local cache of active and inactive peers. In a traditional system, before round t_p, movie M_{250} will not be in any local cache since no peer requests this unpublished movie. Hence, the system will have a very heavy workload at the publishing instance.

With the push strategy, the new movie M_{250} can be pushed into inactive peers' cache before being published. We divide a day into 24 rounds (1 hour/round), of which we allow 18 rounds in advance to push a new movie before the publication.

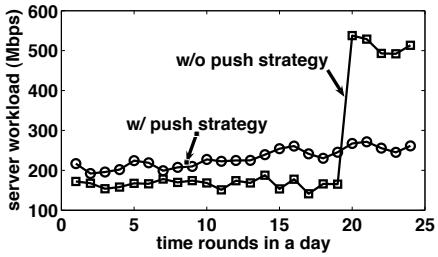


Fig. 7: Performance improvement of using the push strategy

To avoid introducing much overhead, in each round we only push the data of this new movie into forty inactive peers. At $t_p = 19$, the new movie is published. After the publishing instance of the new movie, we do not push the data any more.

Fig. 7 illustrates the improvement using the push strategy. With a limited overhead, this strategy can significantly decrease the server's workload at the publishing instance of a new movie. In other words, this allows the system to "prepare" the data ahead of the flash crowd event. In comparison, the system without using the push strategy will experience heavy workload not only at the time of the new movie publication, but also in the following rounds after publishing the movie, as it is slow to adjust to the optimal replication ratio.

VI. Related Work

Replication has long been an effective strategy to improve performance in P2P systems. Existing literatures [3], [6], [7], [12] discussed replication replacement strategies to optimize file availability, minimize traveling distance or maximize search effectiveness. They vary in objectives and hence lead to different conclusions. Recent popularity of P2P-VoD systems leads to the rethinking of replication. Previous replication algorithms are not suitable due to a very different objective: to minimize the server's workload. Authors in [4] showed the benefit of a peer-assisted VoD system by utilizing peers' upload bandwidth, but peers only redistribute content to concurrent peers and hence there is no real replication. Huang *et al.* [5] discussed the design and implementation of a P2P-VoD system and showed replication can reduce server's workload; but the proportional replication policy was reported to have poor performance for unpopular movies in [13]. Similar proportional replications were also implemented in [17], [18]. Poon *et al.* [8] reformulated the replication problem in terms of minimizing server's workload. Cheng *et al.* [2] proposed and evaluated a heuristic algorithm of lazy replication. These two works were based on simulations but did not provide theoretical analysis. Tan *et al.* [11] discussed content placement for a P2P-VoD system. Wu *et al.* [15] used dynamic programming to derive the optimal replacement strategy for P2P-VoD system. These two works assumed homogenous upload capacity, which is different from our model which considers peer heterogeneity. Besides, our model differentiates concurrent peers and replication peers which is different from [15]. Suh *et al.* [10] considered push-to-peer scheme for a set of always-online homogeneous peers, where the content server only pushes out data and the peers only pull contents. This differs from our scheme where we let heterogeneous peers to push out data proactively. Chen *et al.* [1] considered pre-fetching strategy for a peer-assisted IPTV system based on a *structured IPTV platform*, which is different from unstructured P2P systems.

VII. Conclusion

Movie replication is important in the design of P2P-VoD systems. However, it remains an open problem to answer what the optimal replication policy should be so as to minimize the server's workload. In this paper, we present mathematical models and formulate an optimization framework to understand the impact of movies' popularities on server's workload, and reveal important principles in designing optimal replication algorithms. We show that conventional proportional replication strategy is far from optimal; rather, one should be more "*aggressive*" to replicate unpopular movies. Furthermore, to operate the system at the optimal point, we propose both passive replacement and active push strategies. We discuss practical implementational issues and validate via extensive simulations to show that we achieve high QoS guarantee in streaming and at the same time, reduce workload at the server.

References

[1] 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 System*, 15(1):19–32, 2009.

[2] B. Cheng, L. Stein, H. Jin, and Z. Zhang. A framework for lazy replication in p2p vod. In *Proc. of NOSSDAV*, 2008.

[3] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proc. of ACM SIGCOMM*, 2002.

[4] C. Huang, J. Li, and K. W. Ross. Can internet video-on-demand be profitable? In *Proc. of ACM SIGCOMM*, 2007.

[5] Y. Huang, T. Z. Fu, D.-M. Chiu, J. C. S. Lui, and C. Huang. Challenges, design and analysis of a large-scale P2P-VoD system. In *Proc. of ACM SIGCOMM*, 2008.

[6] J. Kangasharju, K. Ross, and D. Turner. Optimizing file availability in peer-to-peer content distribution. In *Proc. of IEEE INFOCOM*, 2007.

[7] W. Lin, C. Ye, and D.-M. Chiu. Decentralized replication algorithms for improving file availability in P2P networks. In *Proc. of IWQoS*, 2007.

[8] W. Poon, J. Lee, and D.-M. Chiu. Comparison of data replication strategies for peer-to-peer video streaming. In *Proc. of ICICS*, 2005.

[9] T. Qiu, Z. Ge, S. Lee, J. Wang, Q. Zhao, and J. Xu. Modeling channel popularity dynamics in a large iptv system. In *Proc. of ACM SIGMETRICS*, 2009.

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

[11] B. Tan and L. Massoulie. Brief announcement: adaptive content placement for peer-to-peer video-on-demand systems. In *Proc. of ACM PODC*, 2010.

[12] S. Tewari and L. Kleinrock. Proportional replication in peer-to-peer networks. In *Proc. of IEEE INFOCOM*, 2006.

[13] K. Wang and C. Lin. Insight into the P2P-VoD system: Performance modeling and analysis. In *Proc. of ICCCN*, 2009.

[14] D. Wu, Y. Liu, and K. Ross. Queuing network models for multi-channel P2P live streaming systems. In *Proc. of IEEE INFOCOM*, 2009.

[15] J. Wu and B. Li. Keep cache replacement simple in peer-assisted vod systems. In *Proc. of IEEE INFOCOM mini conference*, 2009.

[16] W. Wu and J. C. S. Lui. Exploring the optimal replication strategy in P2P-VoD systems: characterization and evaluation, available at http://www.cse.cuhk.edu.hk/~cslui/infocom_2011_TR.pdf.

[17] L. Ying and A. Basu. pcVOD: Internet peer-to-peer video-on-demand with storage caching on peers. In *Proc. of DMS*, Banff, Canada, 2005.

[18] W.-P. Yiu, X. Jin, and S.-H. Chan. Vmesh: Distributed segment storage for peer-to-peer interactive video streaming. *IEEE JSAC*, 25(9):1717–1731, 2007.