Keep Cache Replacement Simple in Peer-Assisted VoD Systems

Jiahua Wu, Baochun Li
Department of Electrical and Computer Engineering
University of Toronto
{*jiahua, bli*}*@eecg.toronto.edu*

***Abstract*—Peer-assisted Video-on-Demand (VoD) systems have not only received substantial recent research attention, but also been implemented and deployed with success in large-scale real-world streaming systems, such as PPLive [1]. Peer-assisted Video-on-Demand systems are designed to take full advantage of peer upload bandwidth contributions with a *cache* on each peer. Since the size of such a cache on each peer is limited, it is imperative that an appropriate cache replacement algorithm is designed. There exists a tremendous level of flexibility in the design space of such cache replacement algorithms, including the simplest alternatives such as Least Recently Used (LRU). Which algorithm is the *best* to minimize server bandwidth costs, so that when peers need a media segment, it is most likely available from caches of other peers? Such a question, however, is arguably non-trivial to answer, as both the demand and supply of media segments are stochastic in nature. In this paper, we seek to construct an analytical framework based on optimal control theory and dynamic programming, to help us form an in-depth understanding of *optimal* strategies to design cache replacement algorithms. With such analytical insights, we have shown with extensive simulations that, the performance margin enjoyed by optimal strategies over the simplest algorithms is *not* substantial, when it comes to reducing server bandwidth costs. In most cases, the simplest choices are good enough as cache replacement algorithms in peer-assisted VoD systems.**

I. Introduction

Peer-assisted Video-on-Demand (VoD) systems have not only received substantial recent research attention [2], but also been implemented and deployed with success in large-scale real-world streaming systems, such as PPLive [1]. The essential advantage of peer-assisted VoD systems is to take full advantage of peer upload bandwidth contributions, using a pre-determined and fixed amount of disk space on each peer as a *cache* of recently downloaded media segments.

In the PPLive VoD system [1], for example, the pre-determined size of such a media cache may be in the order of 1 GB. As peers download media segments from servers or other peers for on-demand playback, they are stored in the cache to potentially serve other peers, hence reducing bandwidth costs on servers. Peers in PPLive, however, do not choose to *actively prefetch* media segments that it has not requested for playback. One of the reasons against the use of active prefetching in PPLive is that downloading activities on most real-world DSL peers may affect upload bandwidth contributions [1].

As the size of such a cache on each peer is limited, it is imperative that an appropriate cache replacement algorithm is designed. Such an algorithm is critically important, as it determines the availability of media segments in the entire system. Such segment availability affects server bandwidth costs, as peers will download segments directly from servers if they become unavailable from other peers.

Consider peer-assisted VoD systems with a limited pool of *passive* media cache space, where no active prefetching is used and the media cache content is not actively managed by centrally controlled servers. We have good news and bad news in this case. The good news is that there exists a tremendous level of flexibility in the design space of such cache replacement algorithms, including the simplest alternatives such as Least Recently Used (LRU) and Least Frequently Used (LFU). Which algorithm is the best to minimize server bandwidth costs, so that when peers need a media segment, it is most likely available from caches of other peers? The bad news is that such a question is arguably non-trivial to answer, as both the demand and supply of media segments are stochastic in nature in the system.

In this paper, despite the challenges imposed by both stochastic demand and stochastic supplies of media segments, we seek to construct a tractable analytical framework based on optimal control theory and dynamic programming, to help us form an in-depth understanding of *optimal* strategies to design cache replacement algorithms. With complementary simulation studies, we show that the performance margin enjoyed by a practical algorithm design of optimal strategies over the simplest algorithms is *not* substantial, with respect to reducing server bandwidth costs. In most cases, the simplest choices are good enough as cache replacement algorithms in peer-assisted VoD systems. To our knowledge (with detailed justifications in the next section), this work represents the first attempt to analytically consider *passive* peer caching in peer-assisted VoD systems, derive optimal solutions of cache replacement algorithms, and reach the conclusion that they are only marginally (3% – 10%) better than the simplest algorithms.

The remainder of this paper is organized as follows. In Sec. II, we discuss the originality of our work in the context of related work. Sec. III describes the basic concepts in the analytical model. In Sec. IV, we develop an analytical framework and derive optimal solutions. We propose our optimal cache replacement algorithm in Sec. V. In Sec. VI, we resort to simulations to compare the performance of the optimal algorithm with the simplest alternatives, and make the observation that simple solutions are good enough. Finally, Sec. VII concludes the paper.

This work was supported in part by Bell Canada through its Bell University Laboratories R&D program.

II. Related Work

The application of cache management was comprehensively investigated in the VoD services over cable networks [3]–[5]. In cable networks, content subscribers appear to be more cooperative because the facilities, such as home gateways or set-top boxes, are under the control of a network or content provider. Comprehensive theoretical analyses on peer-assisted VoD services in cable networks were conducted in [4], [5]. However, the emphasis of our paper differs from those work in that we focus on the cache replacement algorithm in a system where the cache is passively, rather than actively, managed. Allen *et al.* [3] did a trace-driven evaluation in cache replacement algorithms on cable networks. Although similar conclusions were drawn, our paper distinguishes from [3] in that [3] completely relies on simulation, rather than theoretical aspects considered in our work.

Recently, in the Internet, there also emerged a number of application-level tree-based or mesh-based P2P VoD protocols. However, only simple cache replacement algorithms were utilized in these proposals, *e.g.*, LRU [6] or simply replacing pieces earlier than the current file position [7]. Parvez *et al.* [8] did a comprehensive theoretical analysis on the performance of on-demand stored media content delivery using BitTorrent-like protocols. However, the local cache of the peers were assumed to be unlimited in their analysis. To the best of our knowledge, this paper presents the first theoretical analysis on peer-assisted VoD systems considering passive peer caching.

III. System Model

In this section, we present our theoretical model for the cache replacement problem in P2P VoD systems. The probability distribution of segment popularity is known *a priori* in the systems we investigate. Let there be N peers and a dedicated server in the system. The server stores a repository of media files to be streamed on demand with a constant bit rate. The length of the files may vary, but each of them is composed of a number of fixed-length segments and the total number of segments stored in the server is of size M. The access probability of segment i is assumed to be f_i. The sum of access probability f_i satisfies the condition $\sum_{i=1}^{M} f_i = 1$. Without loss of generality, we assume $f_i \geq f_j, \forall i < j$.

Each peer in the system maintains a limited size of local cache, which is able to store up to B segments. We assume that peers in the system are homogeneous, which indicates that all the peers have the same upload capacity and local cache size. We analyze the system performance within the context of time intervals. The duration of a time interval corresponds to the amount of time for a peer to upload a single segment.

By making the aforementioned assumptions, the behavior of the peers can be characterized in a time-slotted fashion. Let vector $\mathbf{X}(t) = (x_1(t), x_2(t), \ldots, x_M(t))^T$ denote the state of the peers' cache in the system. Each entry $x_i(t)$ stands for the number of peers that hold segment i in its local cache at time interval t. The local cache of all peers will eventually be completely filled as time progresses. Without loss of generality, we assume the cache of all the peers are filled up at time interval 0. The sum of all the entries in the state vector is equal to the storage space from all the peers, namely $\sum_{i=1}^{M} x_i(t) = BN$, $\forall t$.

At the beginning of every time slot, each peer will request a segment it will playback in the near future from the peers that have this specific segment in their local cache. Once this request is satisfied by one of the target peers, the peer will download the entire segment during this time slot. The remaining unsatisfied requests are redirected to the dedicated server and all requests sent to the server are assumed to be satisfied within the time slot. At each time slot, it is natural for a peer to receive several requests from other peers, but it is able to respond to only one request at each slot and it selects the request uniformly at random.

After introducing the working mechanism of the system, we can now model it in a more rigorous way. At each time slot, the requests sent from the peers can be characterized by a random vector $\mathbf{W}(t) = (w_1(t), w_2(t), \ldots, w_M(t))^T$, where $w_i(t)$ denotes the number of requests for segment i at time slot t. In the problem with the knowledge of segment popularity, the random vector $\mathbf{W}(t)$ follows a multinomial distribution with parameters N and $(f_1, f_2, \ldots, f_M)$.

At the same time, each peer downloads a new segment and replaces one stored in its local cache in every time slot. Like the random demand vector, the segments being replaced can also be described by a control vector $\mathbf{U}(t) = (u_1(t), u_2(t), \ldots, u_M(t))^T$, where $u_i(t)$ denotes the number of segment i being replaced at time slot t. This control vector is the key parameter in our system, since the selection of segments to be replaced in each time slot is directly determined by the cache replacement algorithm. It is easy to see that the control vector should obey the following rules:

$$\sum_{i=1}^{M} u_i(t) = N, \qquad \forall t \tag{1}$$

$$0 \leq u_i(t) \leq x_i(t), i = 1, 2, \ldots, M, \quad \forall t \tag{2}$$

$$u_i(t) \in \mathbb{N}, i = 1, 2, \ldots, M, \qquad \forall t \tag{3}$$

However, the control vector is a macro-parameter of the system and it may sometimes cause a conflict for its allocation to individual peers.

IV. Analysis of Cache Replacement Algorithms

We now proceed to derive the optimal cache replacement algorithm in the system with the knowledge of segment popularity. The state that the system evolves to after the tth time interval is

$$\mathbf{X}(t+1) = \mathbf{X}(t) + \mathbf{W}(t) - \mathbf{U}(t) \tag{4}$$

where $\mathbf{X}(0)$ is the initial state of the system. This state is determined by the time that the cache of all the peers are filled up. Equation (4) characterizes the state evolution of our system.

The purpose of a cache replacement algorithm is to vary the availability of different segments so as to fully utilize the storage and upload resources from peers. In each time slot, we are willing to see that most of the requests are able

to be satisfied by other peers, rather than directly from the server. The optimality of the cache replacement algorithm can be measured by the percentage of requests satisfied by the server directly. The mathematical formulation of this problem is as follows:

$$\min_{\mathbf{U}(t)} E\{\sum_{t=0}^{T-1} \mathbf{1}^T \cdot \max\left(\mathbf{0}, \mathbf{W}(t) - \mathbf{P}(t)\mathbf{X}(t)\right)\} \qquad (5)$$

where $\mathbf{1}$ is a $M \times 1$ vector with all the entries equal to 1. $\mathbf{P}(t) = \text{diag}(\bar{p}_1(t), \bar{p}_2(t), \ldots, \bar{p}_M(t))$ is a diagonal matrix. Each entry $\bar{p}_i(t)$ on the diagonal indicates the average probability of the peers currently holding segment i to respond to the requests for this segment. To rigorously define $\bar{p}_i(t)$, we first define $\mathcal{A}_i = \{i \mid \text{peer } j \text{ has segment } i \text{ in its local cache}\}$, the set of peers holding segment i in its local cache. Hence $\bar{p}_i(t) = \sum_{j \in \mathcal{A}_i} p_{ij}(t)/x_i$, where $p_{ij}(t)$ denotes the probability of peer j selecting a request for segment i at time interval t.

In order to investigate the properties of the serving probability matrix $\mathbf{P}(t)$, we first derive the value of $p_{ij}(t)$. Denote K_j and N_j, the number of requests for segment i that peer j receives and the total number of requests that peer j receives, respectively. $\mathcal{B}_j$ indicates the set of segments that peer j has in its local cache.

$$\begin{aligned} & P\{\text{peer } j \text{ selects a request for segment } i\} \\ = & \sum_{n=0}^{N-1} \sum_{k=0}^{n} \frac{k}{n} \cdot P\{K_j = k \mid N_j = n\} \cdot P\{N_j = n\} \\ = & \sum_{n=0}^{N-1} \frac{1}{n} P\{N_j = n\} \cdot E\{K_j \mid N_j = n\} \\ = & \frac{f_i}{\sum_{k \in \mathcal{B}_j} f_k} \end{aligned}$$

We then have

$$\bar{p}_i(t) = \sum_{j \in \mathcal{A}_i} p_{ij}(t)/x_i = \frac{1}{x_i} \sum_{j \in \mathcal{A}_i} \left(\frac{f_i}{\sum_{k \in \mathcal{B}_j} f_k}\right) \qquad (6)$$

Since $f_i \geq f_j, \ \forall i < j$, then we have $\bar{p}_i(t) \in [\frac{f_i}{B \cdot f_1}, \frac{f_i}{B \cdot f_M}]$. The serving probability matrix $\mathbf{P}(t)$ is an important parameter, since it reflects the peer selection algorithm, segment selection algorithm and some other mechanisms in the system design.

$\mathbf{W}(t)$ in Equation (5) represents the random demand for each segment at time interval t, while $\mathbf{P}(t)\mathbf{X}(t)$ stands for the average supply from the peers. Though the contribution peers make to serve others at each time interval is random as well, we take its mean to approximate the random supply in the objective function. A cache replacement algorithm is regarded as optimal if it minimizes the burden of the server, which in this case is equivalent to minimizing the sum of the difference over T time slots.

This is a stochastic optimal control problem, and we resort to Dynamic Programming (DP) algorithms [9] to solve it. First, we need to define some notations. Let

$$\begin{aligned} g_T(\mathbf{X}(T)) &= 0 \\ g_t(\mathbf{X}(t), \mathbf{W}(t)) &= \mathbf{1}^T \cdot \max\left(\mathbf{0}, \mathbf{W}(t) - \mathbf{P}(t)\mathbf{X}(t)\right), \\ & t = 0, 1, \ldots, T-1 \end{aligned}$$

The DP algorithm for our problem is illustrated as follows. For every initial state $\mathbf{X}(0)$, the optimal cost $J^*(\mathbf{X}(0))$ of the problem is equal to $J_0(\mathbf{X}(0))$, given by the last step of the following algorithm, which proceeds backwards in time from period $T-1$ to period 0:

$$\begin{aligned} J_T(\mathbf{X}(T)) &= g_T(\mathbf{X}(T)), \\ J_t(\mathbf{X}(t)) &= \min_{\mathbf{U}(t) \in \mathcal{U}(\mathbf{X}(t))} E\{g_t(\mathbf{X}(t), \mathbf{W}(t)) + \\ & \qquad J_{t+1}(\mathbf{X}(t+1))\}, \\ & \qquad t = 0, 1, \ldots, T-1 \end{aligned} \qquad (7)$$

where the expectation is taken with respect to the probability distribution of $\mathbf{W}(t)$. $\mathcal{U}(\mathbf{X}(t))$ is the set of $\mathbf{U}(t)$ satisfying Equations (1) – (3). If $\mathbf{U}^*(t) = \mu_t^*(\mathbf{X}(t))$ minimizes the right side of Equation (7) for each $\mathbf{X}(t)$ and t, the policy $\pi^* = \{\mu_0^*, \ldots, \mu_{T-1}^*\}$ is considered to be optimal.

However, it is impossible to obtain an optimal policy in the closed form for our problem, and a numerical solution is therefore necessary. The computational requirements to obtain this solution are overwhelming, as they increase exponentially as the size of the problem increases, commonly referred to as "the curse of dimensionality" [9]. Fortunately, for this specific problem, it is possible to suppress the temporal component of the model and determine a one time interval optimum that is also optimal for the problem with the planning horizon T, when T is large enough.

Theorem 1: A myopic policy (one time interval optimum) is optimal for the problem described by Equations (1) – (5) with a sufficiently large planning horizon T.

Due to space constraints, the details of the proof are omitted, however, the basic idea is the following. In the steady state, the control vector is able to compensate the impact of the random demand from peers, which leaves the same system state at the end of each time interval. Thus, one time interval optimum is also optimal for time slots in the steady state. For a sufficiently large planning horizon T, the impact of the steady state is dominant, and thus the announced results follow.

Theorem 1 tells us that an optimal policy in any time interval t is optimal over the planning horizon T. It remains to derive a one time interval optimum for our problem. To this end, consider the time interval $T-1$. It should be mentioned that the replacement of content occurs at the end of each time slot, thus in order to minimize the cost function at time interval $T-1$, we should optimize the control vector $\mathbf{U}(t-2)$. If we denote $x^+ \triangleq \max\{x, 0\}$, the cost function at the $T-1$th time interval $J_{T-1}(\mathbf{X}(T-1))$ is equal to

$$\begin{aligned} \min_{\mathbf{U}(T-2)} \quad & E\{\mathbf{1}^T(\mathbf{W}(T-1) \\ & -\mathbf{P}(T-1)\mathbf{X}(T-1))^+ + J_T(\mathbf{X}(T))\}. \end{aligned}$$

Plugging (4) into the cost function and replacing $J_T(\mathbf{X}(T))$ with 0, $J_{T-1}(\mathbf{X}(T-1))$ can be expressed as follows:

$$\min_{\mathbf{U}(T-2)} \quad E\{\mathbf{1}^T(\mathbf{W}(T-1)-\mathbf{P}(T-1)(\mathbf{X}(T-2) +\mathbf{W}(T-2)-\mathbf{U}(T-2)))^+\}.$$

Changing the order of expectation and summation in the last formula, the optimal control problem at time interval $T-1$ can be obtained:

$$\min_{\mathbf{U}(T-2)} \quad \mathbf{1}^T\{(\mathbf{I}-\mathbf{P}(T-1))\overline{\mathbf{W}}-\mathbf{P}(T-1)\mathbf{X}(T-2) +\mathbf{P}(T-1)\mathbf{U}(T-2)\}^+$$

subject to the constraints

$$\mathbf{1}^T\cdot\mathbf{U}(T-2)=N$$
$$\mathbf{0}\preceq\mathbf{U}(T-2)\preceq\mathbf{X}(T-2)$$

where $\mathbf{I}$ is an $M\times M$ identity matrix and $\overline{\mathbf{W}}=\{\bar{w}_1,\ldots,\bar{w}_M\}$ is the expected value of the random demand vector $\mathbf{W}$, which is equal to $\{f_1N,f_2N,\ldots,f_MN\}^T$. We can introduce an $M\times 1$ vector $\mathbf{K}$, which satisfies the following constraints:

$$\mathbf{K}\succeq\mathbf{0} \tag{8}$$

$$(\mathbf{I}-\mathbf{P}(t+1))\overline{\mathbf{W}}-\mathbf{P}(t+1)\mathbf{X}(t)+\mathbf{P}(t+1)\mathbf{U}(t)\preceq\mathbf{K} \tag{9}$$

Thus, the optimal cache replacement policy $\mathbf{U}^*(t)$ at each time slot t can be concluded by the following theorem.

Theorem 2: The optimal cache replacement strategy is $\pi^*=\{\mathbf{U}^*(0),\mathbf{U}^*(1),\ldots,\mathbf{U}^*(T-1)\}$. For each $\mathbf{U}^*(t)$, it is the solution to the following mixed-integer programming problem:

$$\min_{\mathbf{K},\mathbf{U}(t)} \quad \mathbf{1}^T\cdot\mathbf{K} \tag{10}$$

subject to the constraints (1) – (3), (8) and (9).

V. Practical Cache Replacement Algorithm Design

We now begin to design practical cache replacement algorithms, based on insights from our theoretical analysis derived in Sec. IV. Such practical algorithms can serve as a "benchmark" of the *best possible* algorithms that may be realizable, to which simpler alternative heuristics can be compared.

Our basic idea is to develop the algorithm based on the optimal cache replacement strategy, mixed-integer optimization problem (10), derived in Sec. IV. However, there exist three challenges in utilizing this optimization formulation. *First*, the mixed-integer optimization problem is NP-hard in general [10]. In other words, it is unlikely to be solved in polynomial time. *Second*, future knowledge of the serving probability matrix is needed to derive the optimal solution. *Third*, even if we can obtain optimal integral solutions, there might exist problems to allocate the control vector to individual peers.

Approximation algorithms should be designed to address these challenges. For the mixed-integer optimization problem in the first challenge, it can be relaxed to a corresponding linear programming problem while discarding the integer constraints. In fact, this relaxation does not undermine the optimality of the solution. To respond to the second challenge and the lack of future knowledge, we can simply use the serving probability matrix that is available at the current time. With respect to the third challenge, the conflict of allocation of the control vector can be solved by the following probabilistic replacement strategy.

Rather than selecting the segment for replacing peer by peer, we can set the probability of replacement for segment i to be $q_i(t)=u_i^*(t)/N$ and distribute these probabilities to all peers. Each peer makes its decision according to the replacement probability distribution individually. The benefit of this scheme is two fold. First, by letting the peers make their own decisions individually, the control overhead is quite low in our algorithm. The control messages only occur when the optimization is re-solved. Second, this algorithm copes better with the situation that peers replace segments asynchronously. In the asynchronous case, it is not feasible to allocate the control vector to individual peers in a centralized manner.

With these measures, we can solve the optimization problem as a linear programming problem in a centralized fashion, known to be solvable in polynomial time.

VI. Performance Evaluation

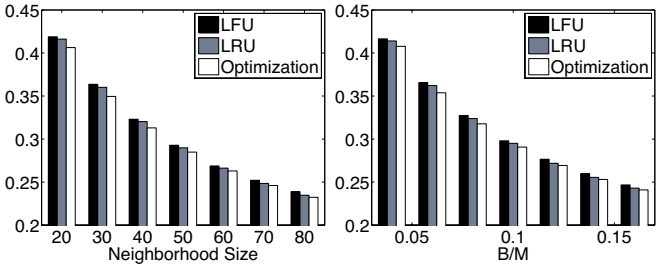
In this section, via complementary simulation studies, we evaluate two commonly used candidates: Least Recently Used (LRU) and Least Frequently Used (LFU), in comparison with the optimization-based algorithm that we have designed, arguably the best achievable strategy in practice.

Unless otherwise specified, we simulate systems that consist of random topologies of 10000 peers, with each peer connected with k neighbors on average. Recall that B and M denote the number of segments a peer is able to store in its local cache and the total number of segments in the system, respectively. B/M, the local cache size from each peer over the total number of segments in the system, is set to be 10% in most simulations.

Since our consistent objective is to mitigate the server bandwidth cost, the main performance metric of concern in our comparisons is the average server load over the total demand, *i.e.,* the portion of requests served by the server. Unless otherwise specified, the Y-axis in the subsequent figures indicate the average server load over the total demand. We further assume the segment popularity follows a stretched exponential distribution (SE), which was found in recent work to be the access pattern for most media workloads in Internet [11]. Hence, it is an appropriate approximation of the segment popularity distribution in our simulation.

A. Effects of Neighborhood Sizes and Cache Sizes

Fig. 1 shows the effects of neighborhood sizes and cache sizes on the server load reduction. We observe that the server load decreases dramatically with the increase of the neighborhood size and cache size. The curves in these two figures show similar trends, since the increase of the neighborhood size is equivalent to the increase of the cache size per peer.



(a) Various Neighborhood Sizes (b) Various Cache Sizes

Fig. 1. Average server load over the total demand under various neighborhood sizes and cache sizes, in large systems with 10000 peers. We set B/M to 10% in (a) and k to 50 in (b).

Further, it is most revealing that the differences among the three cache replacement strategies are notably small. Though our optimization based algorithm shows consistently better performance than much simpler heuristics, the improvement is insignificant. The differences are slightly more pronounced in small caches. As the cache size increases, there is little, if any, difference when the three algorithms are compared with one another.

B. Effects of System Sizes

After investigating large systems with 10000 peers, we are now interested on the effects of varying system sizes on the performance of cache replacement algorithms. Fig. 2 presents the average server load over the total demand while varying the system size from 1000 peers to 10000 peers. From Fig. 2, we can clearly see that the average server load over the total demand remains constant with the change of the system size in all these three algorithms. This demonstrates a superior degree of scalability of peer-assisted VoD systems, where new peers contribute more caching storage space to the system. The bad news is that, even with such stellar scalability, a linear increase of server bandwidth costs need to be incurred to accommodate a linear increase of the system scale, and using even the practically optimal cache replacement algorithm does not make a significant difference.

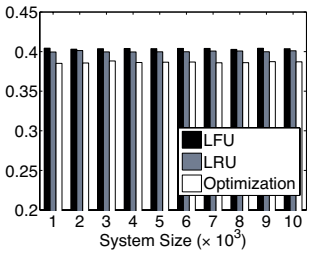


Fig. 2. Average server load over the total demand under various system sizes. We set B/M to 5% and k to 50.

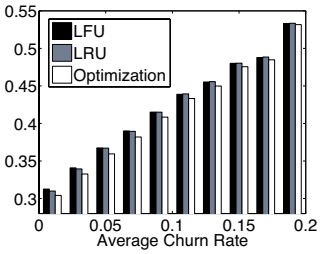


Fig. 3. Average server load over the total demand under various churn rates, in large systems with 10000 peers initially. We set B/M to 10% and k to 50.

C. Effects of Peer Churn

Finally, although all previous analyses and simulations are performed in the static setting, we are also interested in the system performance under the effects of peer churn, including both peer arrivals and departures. The performance of all three algorithms in terms of the average server load over the total demand is shown in Fig. 3. The average server load over the total demand increases dramatically as the churn rate increases. The performance of cache replacement algorithms are similar to that in the static case when the churn rate is small. However, in the case of high churn rates, there exist no substantial performance gain when the optimal algorithm is used rather than its simpler alternatives. When the peer lifetime is short with a high churn rate, a substantial percentage of peers are not able to fill up their local cache before they leave the system, making it futile to optimize the design of the cache replacement algorithm.

VII. Conclusion

Real-world peer-assisted VoD systems commonly use peer caches that are limited in size and passively managed. What, then, is the *best possible* way to manage these peer caches? It is non-trivial to answer this question, since the passively managed nature of peer caching makes both demand and supply of media segments stochastic in nature. In this paper, we seek to answer this question with rigorous theoretical analysis and complementary simulation studies. Using control theory and dynamic programming, we construct a tractable analytical framework and derive the optimal strategy in systems with the knowledge of segment popularity. With our analysis, we are able to gain new insights on how an optimal algorithm should be designed; with our simulations, we are able to compare the performance of our optimal algorithm with the simplest heuristics, and draw the surprising conclusion that simple solutions perform as well as the optimal algorithm, with very insignificant differences.

References

[1] 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 *Proc. of ACM SIGCOMM*, Aug. 2008.

[2] C. Huang, J. Li, and K. W. Ross, "Can Internet Video-on-Demand Be Profitable?" in *Proc. of ACM SIGCOMM*, 2007, pp. 133–144.

[3] M. S. Allen, B. Y. Zhao, and R. Wolski, "Deploying Video-on-Demand Services on Cable Networks," in *Proc. of International Conference on Distributed Computing Systems*, Jun. 2007.

[4] K. Suh, C. Diot, J. Kurose, L. Massoulie, C. Neumann, D. Towsley, and M. Varvello, "Push-to-Peer Video-on-Demand System: Design and Evaluation," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 9, pp. 1706–1716, Dec. 2007.

[5] Y. Boufkhad, F. Mathieu, F. de Mongolfier, D. Perino, and L. Viennot, "Achievable Catalog Size in Peer-to-Peer Video-on-Demand Systems," in *Proc. of International Workshop on Peer-to-Peer Systems*, Feb. 2008.

[6] Y. He and Y. Liu, "Supporting VCR in Peer-to-Peer Video-On-Demand," in *Proc. of IEEE International Conference on Network Protocols*, Oct. 2007.

[7] Y. R. Choe, D. L. Schuff, J. M. Dyaberi, and V. S. Pai, "Improving VoD Server Efficiency with BitTorrent," in *Proc. of ACM Multimedia 2007*, 2007, pp. 117–126.

[8] N. Parvez, C. Williamson, A. Mahanti, and N. Carlsson, "Analysis of BitTorrent-like Protocols for On-demand Stored Media Streaming," in *Proc. of ACM SIGMETRICS*, Jun. 2008.

[9] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 2nd ed. Belmont, Massachusetts: Athana Scientific, 2000, vol. I.

[10] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Courier Dover Publications, 1998.

[11] L. Guo, E. Tan, S. Chen, Z. Xiao, and X. Zhang, "The Stretched Exponential Distribution of Internet Media Access Patterns," in *Proc. of ACM Symposium on Principles of Distributed Computing*, Aug. 2008.